# Proteus: Control and Management System

Robert M. Haralick, Yung-Hsi Yao, Linda G. Shapiro, Ihsin T. Phillips, Arun K. Somani,
Jenq-Neng Hwang, Mike Harrington, Craig Wittenbrink, Chung-Ho Chen, Xufei Liu, and Su Chen
Dept. of Electrical Engineering, FT-10
University of Washington
Seattle, WA 98195

## Abstract

*The Proteus is a highly parallel MIMD, multiple instruction, multiple-data machine, optimized for large granularity tasks. The system is designed to use 256 to 1,024 RISC processors. Computer vision algorithms consists of sub-algorithms, which can be executed in parallel. Advanced software system for partitioning, scheduling, development, and execution of tasks can utilize this fact in a data flow programming paradigm to increase throughput. This Paper describes how these modules interact with each others such that permits the efficient control of large grained parallelism without having to handle the general concurrency problem.*

## 1 Introduction

Computer vision has been regarded as one of the most complex and computationally intensive problems. It also transcends a wide range of representations and forms of processing which requires flexibility. The quantity of data processed in a machine vision system suggests that it seeks to have a higher input data rate. High efficiency for computationally intensive activities suggests the algorithm driven systolic network. Flexibility suggests reconfigurability. The combination of the two suggests a data flow architecture.

However, the kinds of data units that are processed in computer vision change as the processes proceed from low to high level. This suggests that instead of thinking that the architecture processes small data units such as a pixel, we can visualize an architecture which processes more complex data units such as the image, digital arc, sets, and relations.

In factory applications of computer vision, the same vision algorithm is applied repeatedly to a succession of images. The input is not an image, but a sequence of images. The output to each image processing operation likewise is a sequence of images. The processor, instead of only processing one simple operation such as an add or multiply on the primitive data unit now must perform an arbitrarily complex sequence of operations on the large data unit. The code run on the processor now does not have to be the kinds of specialized code used for vector processor, pipelines and systolic arrays, or digital signal processors. Rather the code can be the same kind of code written in languages such as C or Ada and which can be tested on standard workstations.

For high input data rate and simple algorithms, such an architecture runs in a single program multiple data stream mode. For low input data rate and highly complex algorithms, such an architecture can reconfigure itself to function in a pipeline network or full multiple instruction multiple data stream mode. We call this architecture the Proteus data flow architecture.

High-level programs for Proteus are written in the INSIGHT programming language [8][9]. Insight is dataflow language rather than a sequential language. This means that the program can translate into a graph structure, where each node is an task to be performed, and each arc is data variable. This graph structure lends itself to a parallel computation system, since when a node's input data is ready, it may perform its computation without regard to any other node's state of computation. Thus, when a node in the graph has completed its computation on an image, it may pass its data to next node(s) and may immediately begin computation on the next image in the sequence when the previous node(s) finishes its computation. In this manner, two types of parallelism are permissible: (1) separate parts of the algorithm may be completed on the same image simultaneously on different processors and (2) pipelining of the sequence of the images through the processor may occur. The INSIGHT program describes the flow of a sequence of images, and other data structures, and their resultant data structures through the Proteus. Each process of the network performs one or more operations on its input image(s) and/or structure(s) to produce output image(s) and/ or other data structure(s). The most important aspect of the INSIGHT language is that it expresses relationships, not commands. The relationships dictate a graph structure that defines the flow of data through the system. Figure 3 illustrates the graph structure for a program given in Figure 4.

The input to the INSIGHT program shown in Figure 4 is a 256x256 gray scale image G0, and the output is a 256x256 binary image B4. Intermediate gray scale images G1, G2, and G3 and intermediate binary images B1, B2, and B3 are also produced during execution of the program. The first relation says that gray scale image G0 is to be thresholded using threshold T1 (a constant), and the result is to become binary image

B1. The second relation says that G0 is also to be the input to a morphological closing operation [4]. with a structuring element that is a box (rectangle) of dimension 5 x 5, with the result becoming gray scale image G1. The third relation specifies the production of another binary image B2 that is the result of performing an opening in G1, subtracting the opening from G1 itself and thresholding the result of the subtraction. The other relations can be analyzed in a similar fashion. A high level data flow language called INSIGHT is used to specify the relation in the Proteus network. The INSIGHT program can be translated into a graph structure, where each node is an task to be performed, and each arc is a data variable.

The parallelism of computer vision algorithms can be exploited through an advanced software system for partitioning, scheduling, development, and execution a data flow programming paradigm. System software includes a translator for the INSIGHT language, a parallel debugger, and high level simulators, and Proteus Control and Management system (PCMS).

In section II of the paper we describe the Proteus Hardware Hardware. In section III we describe the Proteus software System.

## 2 Proteus Hardware System

Proteus is a high-performance MIMD machine. The whole hardware system for Proteus architecture contains a Host as front end processor and 8 to 32 processing groups connected in a hypercube structure with each node of the hypercube representing a group. A top level view of the system is shown in Figure 1. The external input is received on 32 parallel channels which are equally distributed to the enhanced hypercube nodes. The Host works as user interface and file server for back end processors. Algorithms are developed, mapped, and down-load through the Host. The final output generated by the number-crunch processing elements is also collected by the Host. We use Sun Sparc 10 as the Host and a single board SUN SparcEngine 1E with the VMEbus and Ethernet interfaces as Group Controller. All of them operate under UNIX operating system environment. Each group communicates with the host through the Ethernet link.

Data communication between groups is through a circuit switched enhanced hypercube connection. An enhanced hypercube contains two links in any one dimension of a regular hypercube, as shown in Figure 2. A centralized algorithm at the host may route any arbitrary permutation [3]. The 32 groups in a full scale system can thus communicate with each other in an arbitrary permutation for rapid exchange of data. By not buffering the data at the intermediate nodes, the transmission across the diameter of the hypercube are negligible.

Each group contains nine processor clusters, one generalized communication interface board (GCI), one group controller and a crossbar switch. Each group has its own local VME bus backplane which is accessible to all of the clusters, GCI, and the group controller.

Data communication within a group is through a circuit switched cross-bar connection. The GCI board of each group provides the external input interface and supports the communication interface between within-group and outside-group data exchanges.

The GCI board has an Intel 80960CA as its controller. It provides high speed serial communications capabilities to enable efficient transfer of data. Each board contains four frame buffers. Each with a HotRod receiver and transmitter capable of transferring data concurrently at 250MBaud. The HotRod receiver and transmitter are GaAs devices that provide synchronous point-to-point serial communications. The frame buffers are used to facilitate the synchronization of potentially asynchronous external (to the Group) data sources. Each frame buffer contains 256KByte RAM. Typical operation would be to receive asynchronous data into the frame buffer and then re-transmit it with the system in a synchronized fashion.

The basic function of the crossbar switch is to enable the switching of the high speed serial data streams. Each Cluster in the same Group has a HotRod transmitter that is connected to an input of the crossbar switch. Similarly each cluster also contains a HotRod receiver that is connected to an output from the crossbar switch. Each of the four frame buffers of the GCI board has a HotRod transmitter connected to a crossbar switch input and a HotRod receiver connected to a crossbar switch output. The crossbar switch is programmed by the 80960CA on the GCI board.

The cluster architecture is a shared memory multiprocessor system. Communication within a cluster is through shared memory. Each cluster consists of four Intel i860 as Processing Elements (Pixel Processors or PP), one Intel 80960CA as cluster controller, 1 Mbytes SRAM dual port memory and a shared memory with 8 MBytes of DRAM upgradeable to 32 MBytes. The Intel i860 is a 40MHz/MIPS processors with 8k Bytes on chip data cache, and 4k Bytes on chip instruction cache. Each PP has 1 Mbytes of external cache. The cache can also be configured as local memory under software control. The cluster controller is responsible to scheduling task for each PP, controls the synchronization among four PE, manages the cluster communication with group controller, and controls the dual channel DMA of the communication unit.

Figure 2 shows Proteus with 4 cube, and the extra links connecting all nodes in the vertical dimension. The links marked a, b, c, and d are the high speed serial links input and output for one group. The e link is the additional link which allows full permutation capability. The exploded view of the group contains the Unix board group controller (GC), the clusters (C0 to C8), and the communication interface or crossbar (xBar). Clusters are connected by crossbar to each other and to the enhanced hypercube. I/O from external sources is fed through the I/O buffer marked as IB. An exploded view of a single cluster is shown, and consists of the cluster control processor (CCP), the shared memory (SM), the I/O buffer and memory (I/O DPM), and the RISC processors (or pixel processors, PP). Pixel processors in a cluster share memory

and a serial I/O link. External caches and control processors help ease contention and multiprocessing performance degradation.

Currently, a prototype group with 8 clusters and one GCI board has been implemented. In addition, a emulator board has been built to emulate the external input source. The emulator has almost identical configuration as cluster board. Instead of having four pixel processors, 32 Mbyte RAM is installed. First, GCI receives data from emulator board, stores it into frame buffer and then re-transmit it to proper cluster from the frame buffer.

# 3 Proteus Software System

Figure 5 illustrates a top level view showing how the software relates to the Proteus hardware. whose processors are partitioned into groups and within groups partitioned into clusters. The system software consists of a INSIGHT translator runs on the Proteus host, Proteus Control and Management system (PCMS), which run on the group controller, cluster controllers, and GCI, and a simplified version of PCMS runs on the pixel processor.

## 3.1 INSIGHT Translator[7]

The INSIGHT translator converts an INSIGHT program into a sequence of tasks run on the Pixel Processors. The translator is responsible for parsing the algorithm, creating a network of data dependencies, and partitioning the work between the logical processors.

## 3.2 Communication System

The Proteus system consists of a large number of processors, running asynchronously with each other. This form of computing system places the communication architecture as a key component. The communications then become the arteries of a Proteus system.

The goals for the proteus communication system are:

- Be expandable to allow system changes to be as easily incorporated as possible.
- Low overhead – especially for the pixel processors.
- Common interface between processors for the application code, regardless of the actual physical means of communication.
- Reliable communications.
- Allows any given task on a given processor to send and receive messages on the same or others processor.

Message transmission is the transferring of data from the memory space of one processor to the memory space of another processor. In the case of a Cluster Controller (CC) and a Pixel Processor (PP) or from a CC to the Group Controller (GC) a change in the owner (which processor can write to the memory containing the message) processor.

Each GC can directly communicate with the Host and any of its CCs. The CC is capable of directly communicating with its Group Controller (GC) and each of its four PPs. If any processor sends a message to a processor that it is not directly connected to, the message is automatically forwarded through any needed intermediate processors until it arrives at the destination processor. Message forwarding through intermediate processors is invisible to the requesting software.

All transmission requests complete immediately upon being received by the communication software. The transmission requests are queued in communication software queues and are serviced in a first in, first out basis.

The communication software provides a reliable communication mechanism. All messages sent and received by the requesting software will be error free. The communication software will apply the error correction, request for re-transmission, and unrecoverable error logging. All unrecoverable errors will be logged to the Host computer.

### 3.2.1 Inter-Processor Communication

The proteus system uses several different mediums to transfer data between its components, i.e.:

1. Ethernet from the Host to each GC
2. VME within each Group – the GC, GCI and each CC can be bus masters to send message and interrupts.
3. Shared memory and interrupt lines within each cluster – the CC and each PPcan write to any shared memory location on the Cluster. Each PP has one interrupt line to its CC, and one from its CC.
4. High Speed serial – Each cluster in Proteus is a node on a network of high speed (250 Mbit/sec) lines. The interconnection is controlled by a cross bar switch in each Group. The input images to Proteus arrive via this network. The network is also used to transfer images between clusters.

There are a variety of communication paths on the various levels of processors in the Proteus system.

### 3.2.2 Message Format

Even though there are several different physical methods for the processors to communicate with each other, each message will have a standard format. The standard format will allow the using of the same source code as much as possible from one processor type to another.

The message is made up of a header section and the actual information carried DATA. The header contains the destination and sender addresses, the message type, the length of the data portion of the message, and a checksum for the header. Each parameter in the header is 32-bit word.

| Destination Address | |
|---|---|
| Source Address | |
| Message Type | Message Sub-Type |
| Length of Data body in bytes | |
| Spare | |
| Spare | |
| Spare | |
| Two's Complement Checksum | |
| Data<br>⋮ | |

This message format is maintained as the message passes through PCMS, even as the message is converted from an Ether packet to a VME data structure to a Cluster Shared Memory Structure.

Each node must have a unique address in this communication network. Addresses are used in all messages to identify the sender and the receiver. Each node uses its own Id for its address subfield. Thus, this address scheme is physical. The addressing scheme used provides:

- Unique address for each node (processor) in the Proteus system.
- Generic address mode, it is simple to parse an address to understand which processor is being addressed.
- Provides a broadcast address format that can be subdivided.
- Does not limit the number of processors that can be supported in a Proteus system.

**Address Subfields** The node addresses are like a Class B Internet address in layout. The main idea of the addressing is to take into account the hierarchical arrangement of a Proteus system and use that to our advantage. The format of a Proteus processor address is:

GROUP.CLUSTER.PIXEL

Each subfield (GROUP, CLUSTER, or PIXEL) partitions the address down to the hierarchy level for the processor being addressed.

The address is implemented as a 32 bit word. The word is broken into bit fields as follows. The least significant four bits will be used to represent PIXEL (allowing up to 16 PPs per cluster), the next least significant five bits are used to represent CLUSTER (allowing up to 32 clusters per group), and the next seven least significant bits are used to represent the GROUP (allowing up to 128 groups). In the most significant 16 bits, the four least significant bits are reserved for specifying a Host port number in future implementations. The remaining 12 bits are spare, and reserved for future use.

| Physical Layout of Address Word | | | | |
|---|---|---|---|---|
| Spare | Host Socket | GC | CC | PP |
| 12 bits | 4 bits | 7 bits | 5 bits | 4 bits |
| 31–20 | 19–16 | 15–9 | 8–4 | 3–0 |

**Message Type** This is used by the message system to determine which task on the TARGET processor is to receive the message. It optionally consists of two parts:

- Major Type - the general classification of the message (i.e. Debug) and is contained in the Most Significant two bytes of the TYPE.
- Minor or Sub-Type - a more specific classification of the message and is in the Least Significant two bytes of the TYPE.

## 3.3 Executive Scheduler

The Executive Scheduler is the software that resides on both the Group and Cluster controllers that decides which tasks should be executed next. The Executive Scheduler is an event driven non-preemptive round robin scheduler with four distinct task priority level. The Executive Scheduler is driven by both external and internal events. These events are listed below and grouped by internal (an event occurring on or by the local processor) and external (an event caused by a hardware fault or external processor action).

**Internal Events** An internal event is usually a software driven event. The list of internal events are:

- Explicit call to executive scheduler by a task – a task may call the ScheduleTask and ScheduleTaskPri routines to cause a task to be explicitly scheduled.
- Waited for resource availability—a task may wait on a region of memory to become available. When the memory waited on is freed by another task, the dynamic memory system causes an event to schedule the waiting task.
- Timer interval expiring—a task may request of the timer system to be notified when a given timer interval has expired. At the end of the specified interval the timer system causes an event that schedules the specified task.
- Reception of a specified message—a task may "wait" on a specific or general message type. When that specific message is received by the message system that meets the message type criteria the message system causes an event to schedule the specified task.

**External Events** An external event is usually caused by the action of another processor or an external hardware device.

- Interrupt from another processor—the processor hardware interrupt support is used to vector to an interrupt service routine that directly schedules a task. This is used to receive messages from the attached processors.
- Interrupt from timer chip—this interrupt is vectored to the clock interrupt routine to update the software time of the processor, and to declare an event if any timer interval request has come due.
- Interrupt from EOT hardware—at the End Of Transmission interval for the GCI system, and interrupt routine causes a task to become scheduled to perform the post GCI transfer processing.

### 3.4 Loader

According to the INSIGHT program, INSIGHT translator generates an assignment file. Each assignment file provides a description of the logical assignment of the data flow graph nodes to the logical processors. In addition, the file contains details of the list of input and output arcs for each data flow node as well as the number of buffers assigned to each arc.

According to the assignment file(s), a list of tasks is assigned to each physical processor. Each task contains the following attributes: Program_id, input_arcs, output_arcs, constant parameters. Each task has a unique task ID assigned by the loader. Each arc has a unique arc ID assigned by the loader.

The Host Loader packs all the schedule information needed by a cluster and sends them to each cluster controller through the messaging system. These schedule information are stored in the cluster controller database to control the allocation of memory and processing resources. The cluster controllers allocate physical buffers to each arc.

For each cluster, according to the tasks assigned to its subordinate pixel processors, the loader determines which programs should be loaded to that cluster and assigns memory space in shared memory to each program. Then, the host uses the socket facility of the Unix system to send a loading request message to each group via Ethernet. These request messages include two parts. The first part specifies that the action is loading. The second part is the full path name of the transfer request file. According to the received message, Group Controllers retrieve the transfer request files from disks. Each Group Controller reads in files specified in the request file, writes them to the VME buffer of the destination cluster(s) and requests the destination Cluster Controller to move them to the shared memory starting from address specified in the request file.

### 3.5 Debugger

The debugger interacts with the user through the host's window system. It has the capability to support the development of system and application programs. The debugger provides the user with the capabilities to control and monitor the execution of all the pixel processors in the Proteus system.

On the Host, the debugger is simply a front-end interface that interacts with the user, and manages the bulk of debugging information (e.g. symbol tables) that allows it to map symbolic information to physical addresses in the hardware. However, the memory accesses, and modification of processors' execution states must be done by system services provided by the cluster controllers and the pixel processors from the hardware side.

Debugger provides two levels of debugging capability. At the high level, the debugger allows the user to control execution by setting image watch-points (i.e. data breakpoints) in the INSIGHT data flow graph. When a pixel processor reachs an image watch-point, it will report back to user. At the low level, the subject language is the assembly language of the i860 pixel processor. The user may trace through the execution of a program by a pixel processor by setting code breakpoints inside the program and single-stepping through the program.

### 3.6 Pixel Processor Task Scheduling Modules

Pixel processor task scheduling modules are a collection of Cluster Controller tasks and subordinate routines that control scheduling of pixel processor activity. These tasks are run under the control of the Executive Scheduler of CC and active whenever the state of data buffer changes in the cluster. The modules are:

#### 3.6.1 Get Data

It is scheduled by the arrival of an image data. Image data are sent to the cluster either from an external source or from another cluster.

#### 3.6.2 PP-Finish

It is scheduled by completion of a pixel processor task. PP-Finish receives the Task Completion Record message from a pixel processor, it Updates Arc Status, and sends a job-completed message to invoke FindRunnableTask. PP-Finish reads the buffer pointers for the completed job from the Task Control Block for the pixel processor from which the message was received.

#### 3.6.3 PutData

It is scheduled by FindRunnableTask when consumers of buffers created by a completed task are located external to the cluster. PutData acts as a dummy pixel processor task, producing a TaskComplete Message when the message system indicates that the transfer has been completed.

#### 3.6.4 FindRunnableTask

It is scheduled by a change in buffer state. Buffer states change either when GetBuffer processes an image buffer, PutBuffer completes a buffer transfer, a pixel processor completes a task, or FindRunnableTask schedules a new job.

For each buffer, there is a state associated with it. Legal values for the state are:

1. ready and unconsumed: the buffer has valid new data but not all processors which require it have accepted it;
2. ready and consumed: the buffer has valid data which has already been used by the processors that require it and the new data which is to be loaded into the buffer is not yet ready; and
3. not ready: the buffer has no valid data in it.

The data buffer state changes whenever an image enters or leaves the cluster, or an intermediate PP task is allocated or completed.

A task can be executed when all of its input buffers are in the ready and unconsumed state and the output buffers in either not ready or ready and consumed state.

FindRunnableTask is invoked whenever any arc status changes. Arc status changes when GetBuffer completes, PutBuffer completes, when a pixel processor task completes, or when a pixel processor task is scheduled. FindRunnableTask manages processes that check arc status for:

1. Each job dependent on arcs created by the job that finished
2. Tasks dependent on arcs scheduled for creation on the same pixel processor, to pipeline workflow on a single pixel processor
3. Each job that creates arcs that were consumed by the job that finished
4. The job that just finished, in case resources had been updated while it was running that, combined with current resource updates, allow it to restart.

Decision points are based on arc status. The search algorithms in the informational cluster services must search the oldest buffers in the search area first, to preserve job sequence order.

The architecture for FindRunnableTask is functional in nature. The algorithm involves a breadth-first search for jobs that are logically dependent on the resources affected by the arc state change that invoked FindRunnableTask.

### 3.7 SendTCB

When a task becomes executable, the cluster controller transfers task scheduling information from the cluster to the pixel processor via a task control block. The task control block contains the identification tag, starting address of the program to be executed by the pixel processor, and pointers to the buffers that will be used for input and output.

Analysis by Levy [6] shows the i860 to be poor for operating system work due to its inefficient interrupt-handling capability. So extra care was taken in design to minimize interruption of the i860's processing. Each PP has its own task waiting queue in shared-memory. Whenever a task becomes executable, the cluster controller puts a task control block into a PP's task waiting queue and check whether that PP is idle. If it is, interrupting the PP to indicate the task waiting queue is not empty.

Whenever a PP finds its task waiting queue is not empty, removes the first entry from the queue and executes it.

When a PP finishes executing a task, a task completion message is sent back CC. According to the task completed message, CC updates the status of data buffers used by the task just completed, and schedules tasks which become executable due to buffer states changed. When a Pixel Processor completes a task, it consults their task waiting queue to determine their next task. If the queue is empty, the PP just puts itself to sleep. In this manner, busy processes do what they have to do without ever having to be interrupted once they begin their processing.

### 3.8 Monitor of Pixel Processor

The pixel processors are in either monitor or application mode. In the application mode, image processing programs are executed. In the monitor mode, it has to

- Handle interrupt from cluster controller.
- Read the task control block.
- Bind the data in task control block to program parameters.
- Set up the run-time environment for application programs.
- Flush the cache.
- Generate the task completed record and interrupts cluster controller.

When the execution of the application program is completed, the control of the CPU is switched back to the monitor program,

## 4 Summary

Proteus is a data flow architecture designed for computer vision applications where large granularity may be used. A prototype group with 8 clusters and GCI board has been implemented.

We have discussed how the design and implementation of system software which easily permits the efficient control of large grained parallelism without having to handle the general concurrency problem.

## References


[1] M. L. Campbell, "Static Allocation for a Data Flow Multiprocessor," Proceedings of the 1985 International Con- ference on Parallel Processing, 1985, pp. 511-517.

[2] S. B. Choi and A. K. Somani, "The Generalized Folding-Cube Network," NETWORKS, An International Journal, in press.

[3] S. B. Choi and A. K. Somani, "The Generalized Hyper-Cube," in Proceedings of ICPP-90, August 1990, pp. 372-375.

[4] R. M. Haralick, S. R. Sternberg, Y. Zhuang, "Image Analysis Using Mathematical Morphology," IEEE Transactions On Pattern Analysis and Machine Intelligence, Vol. PAM1-9, No. 4, July 1987.

[5] R. M. Haralick et. al., "Proteus: a reconfigurable computation network for computer vision" 11th IAPR International Conference on Pattern Recognition. August 1992.

[6] H. Levy, Personal Communication.

[7] C. F. Olson, "Translation of Machine Vision Programs to Reconfigurable Computational Network Control Codes" Master Thesis, University of Washington, Aug. 15, 1990.

[8] L. G. Shapiro, R. M. Haralick and M. Goulish, "INSIGHT: A Dataflow Language for Programming Vision Algorithms in a Reconfigurable Computational Network," International Journal of Artificial Intelligence and Pattern Recognition, Vol. 1, No. 3/4, 1987, pp. 335-350.

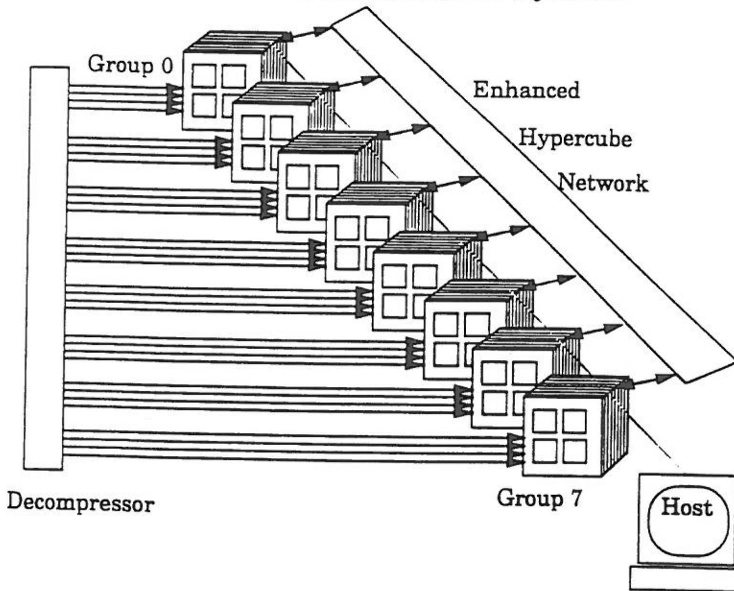
[9] L. G. Shapiro, "Programming Parallel Vision Algorithms: A Dataflow Language Approach," The International Journal for Supercomputer Applications, Vol. 2, No. 4, 1989, pp. 29-44.

[10] A.K. Somani et. al., "Proteus System Architecture and Organization" in Proc. of 5th International Parallel Processing Symposium, Anaheim, California, pages 287 - 294.

[11] W.W. Wadge and E.A. Ashcroft, LUCID: The Dataflow Programming Language, Academic Press, London, 1984.

[12] C. Wittenbrink "Directed Data Cache for High Performance Morphological Image Processing," Masters Thesis, University of Washington Dept. of Electrical Engineering, Oct. 8, 1990.

**Automatic Classification System**




32 Parallel Channels for image transfer

8 Groups Expandable to 32

Each Group has up to 9 Clusters

Each Cluster has 4 Processors

3500 frames/second input

**Figure 1 A top level view of the Proteus architecture**

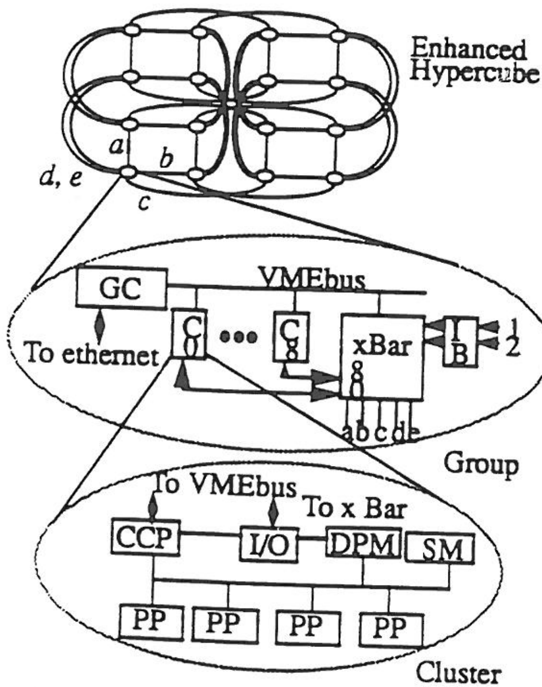



Figure 2. Exploded View of Proteus System

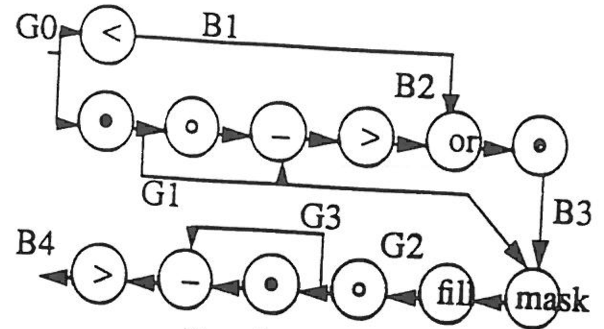



Figure 3. Insight Program

```
function Detect
    (integer array[512,512] G0;)
    (binary array[512,512] B4;)
where
    declare
        integer array[512,512] G1,G2,G3;
        binary array[512,512] B1,B2,B3;
        integer constant T1=195,T2=20,T3=25;
        integer constant W1=5,W2=15,W3=42,W4=126,W5=3;
    relations
        B1 = G0 < T1;
        G1 = G0 closedby box(W1,W1);
        B2 = (G1 - (G1 openedby box(W2,W2))) > T2;
        B3 = (B1 or B2) dilatedby box(W5,W5);
        G2 = fill(G1 maskedby B3);
        G3 = G2 openedby box(W3,W3);
        B4 = (G3 - (G3 closedby box(W4,W4))) > T3;
endwhere
```

Figure 4. Detect Program

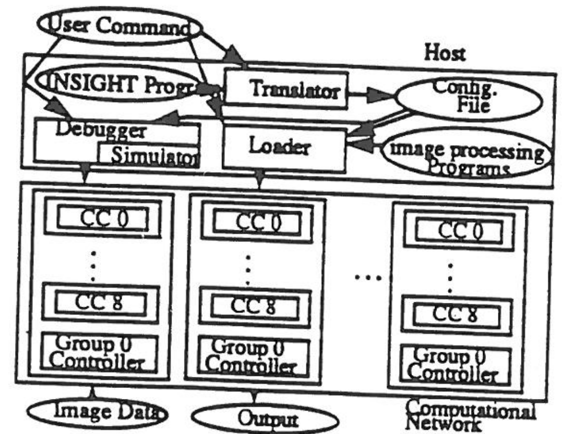



Figure 5. Software View