JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING 6, 598–622 (1989)

A Model for an Intelligent Operating System for Executing Image Understanding Tasks on a Reconfigurable Parallel Architecture*

C. HENRY CHU

Center for Advanced Computer Studies, The University of Southwestern Louisiana, Lafayette, Louisiana 70504

EDWARD J. DELP, LEAH H. JAMIESON, HOWARD JAY SIEGEL, AND FRANCIS J. WEIL

School of Electrical Engineering, Purdue University, West Lafayette, Indiana 47907

AND

ANDREW B. WHINSTON

Graduate School of Business, Department of Computer Science and IC² Institute, University of Texas, Austin, Texas 78712

Received November 20, 1987

Parallel processing is one approach to achieving the large computational processing capabilities required by many real-time computing tasks. One of the problems that must be addressed in the use of reconfigurable multiprocessor systems is matching the architecture configuration to the algorithms to be executed. This paper presents a conceptual model that explores the potential of artificial intelligence tools, specifically expert systems, to design an Intelligent Operating System for multiprocessor systems. The target task is the implementation of image understanding systems on multiprocessor architectures. PASM is used as an example multiprocessor. The Intelligent Operating System concepts developed here could also be used to address other problems requiring real-time processing. An example image understanding task is presented to illustrate the concept of intelligent scheduling by the Intelligent Operating System. Also considered is the use of the conceptual model when developing an image understanding system in order to test different strategies for choosing algorithms, imposing execution order constraints, and integrating results from various algorithms.

* This research was supported by the Air Force Office of Scientific Research under Grant F49620-86-K-0006 and by the Supercomputing Research Center, Lanham, MD.

0743-7315/89 $3.00

1. INTRODUCTION

A new approach to the implementation of image understanding systems on multiprocessor computer architectures is presented. In the simplest descriptive form, an *image understanding system* (*IUS*) takes an image or a set of images from a group of sensors and produces a description of the scene. These systems are also characterized by the need to do a great deal of numeric and symbolic processing in real-time. This type of constraint typically requires the use of special purpose computing systems that can exploit the structure of the algorithms used. One approach to solving this problem is through the use of parallel processing.

The various types of processing required in an image understanding system can roughly be classified into three groups. The first group includes operations that transform an image into another image, such as edge detection where gray level discontinuities in the image are found and the results are represented as an edge map. This type of processing is numerical in nature and requires a processing system capable of fast numerical operations, some of which may be floating point. The second group includes quasi-symbolic computations where the results of numeric image processing, e.g., edges, textures, and features, are used to describe surfaces and shapes of objects in the scene. This level of processing consists of both numeric and symbolic types of operations. The third group comprises mainly symbolic processing used to produce the scene description. These various computations require a large amount of both raw computing power and flexibility of the computing system.

Because image understanding algorithms may have processing requirements that differ from one algorithm to another, it is most efficient to employ different modes of parallelism when image understanding systems are implemented on multiprocessor computer architectures [2, 16]. Window-based data-value independent image processing algorithms are, for example, most efficiently performed using SIMD parallelism where each processor has a local memory and there is only local communications between processors. Other SIMD algorithms, such as histogram algorithms, require global communications among all processors [19]. Contour tracing algorithms are examples of MIMD processes with variable communications patterns [12]. A *partitionable SIMD/MIMD system* is a parallel processing system which can be structured as one or more independent SIMD and/or MIMD machines (*partitions*) of various sizes (e.g., TRAC [18], PASM [19]). Such systems can support the full range of image understanding algorithms.

With the expected growth in multiprocessor computer systems, a key issue is the ability to provide a high-level operating system that is able to exploit fully the hardware architecture. One of the problems with using multiprocessor systems is how to "fit" the algorithms to the architecture, i.e., how to

structure a task for execution on a particular parallel architecture [1]. If the parallel system is reconfigurable there is the problem of choosing an effective system organization, i.e., to determine how the system is to be reconfigured for a given task or group of subtasks. This paper presents a conceptual model that explores the potential of artificial intelligence tools, specifically expert systems, to build cost-effective special purpose operating systems to control such reconfigurations.

The resulting operating system will consist of generalized routines, useful in all environments, and a specialization for a given multiprocessor architecture in the form of expert rules. The PASM [19] system, which permits dynamic reconfiguration, provides a multiprocessor model. The conditions under which a certain configuration would be appropriate are stated in the form of expert rules. The ultimate goal is to combine the reconfiguration expert system of the operating system with the problem solving component. As the image understanding task is processed, various numerical or symbolic processing steps are required. As processing progresses from one algorithm to the next, the new processing requirements are passed to the reconfiguration expert which then generates calls to the operating system routines to reconfigure the system.

Section 2 describes the overall model and discusses some of the issues involved in the development of this operating system. An expert systems approach is used in many components of the overall model; a new expert system language that has been developed is presented in Section 3. An example of executing an image understanding task is presented in Section 4 to illustrate the characteristics of the Intelligent Operating System (*IOS*). Section 5 explores the model further by considering the issues of a user interacting with the Intelligent Operating System to develop an image understanding system on a reconfigurable multiprocessor system.

2. System Model

The overall system model for executing an image understanding task is shown in Fig. 1, illustrating the interaction among the Image Understanding System, the Intelligent Operating System, and the Algorithm Database. An alternative view of this model is shown in Fig. 2, where the knowledge bases and the algorithm databases for each part of the system are grouped according to their levels of operation. It should be noted that there are situations where a human operator interacts with the Intelligent Operating System; one such situation is considered in more detail in Section 5.

The Image Understanding System determines what types of symbolic and numerical operations it wants to perform, and the results from these operations are used to determine what needs to be done next. The Image Under-

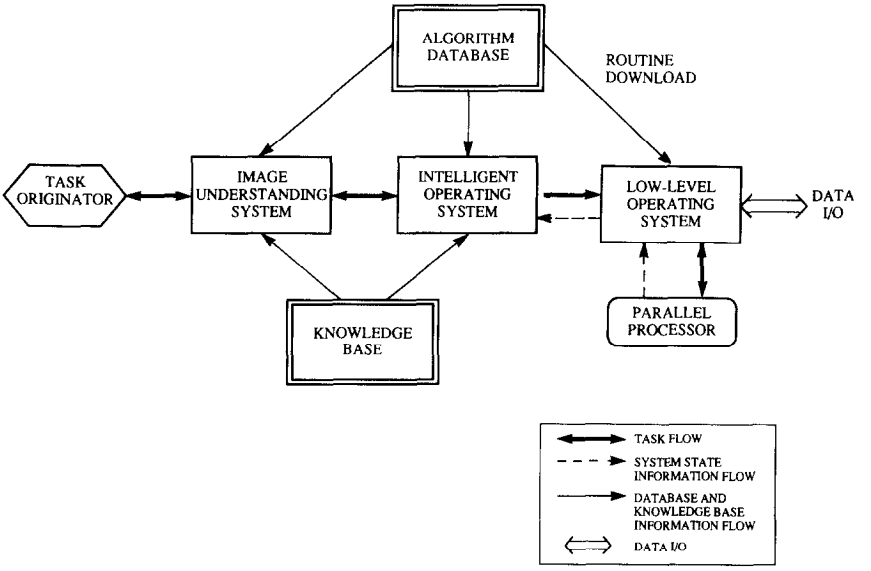

FIG. 1. Overall model of an intelligent operating system executing image understanding tasks on a reconfigurable parallel architecture.

standing System will also make decisions about the particular kinds of algorithms it wants to run. For instance, it will determine what types of intensity edge operators it wants to execute on the basis of the environmental conditions that the sensors are observing. The algorithms that the Image Understanding System can use are stored as the IUS Database part of the Algorithm Database (see Fig. 2).

The Intelligent Operating System component of the model incorporates concepts from the field of expert systems. This expert system will take requests from the Image Understanding System, e.g., "find edges using algorithms W or X and then trace the object contours using algorithms Y or Z." Information about how the algorithms can be mapped onto the multiprocessor architecture is stored in the IOS portion of the Algorithm Database. The expert operating system will then use this parallel implementation information to select from among alternative algorithm implementations and to determine the system configuration. As the particular image understanding task is running, the multiprocessor system will have to partition and reconfigure itself to accomplish all of the numeric and symbolic subtasks requested by the Image Understanding System.

Various scenarios could exist. One could arrive at a situation where the next step is "find the intensity edges in the image using the algorithm X." In the IOS Algorithm Database there may be many different parallel implementations for the algorithm X. These implementations may differ in their use

of system resources, placement of data results in the system memories, and/or execution speeds. The Intelligent Operating System must be able to examine the state of the system and choose the "best" parallel implementation of algorithm X in terms of system performance on the overall task. In doing this, the operating system can partition the multiprocessor such that several numeric and/or symbolic processes are running simultaneously in both SIMD and MIMD mode. The Intelligent Operating System interacts with the "native" Low-Level Operating System that exists on the multiprocessor architecture (see Figs. 1 and 2). This Low-Level Operating System is used to execute the actual system reconfiguration code.

Each new processing step in a task therefore cuts through the three levels shown in Fig. 2. The Image Understanding System (Level 1) generates an algorithm selection based on the knowledge of the task (Circle A) and information about each algorithm's image analysis performance characteristics (Circle B), e.g., how the algorithm will perform in the presence of noise. The algorithm selection is presented in the form of a data dependency graph for the Intelligent Operating System (Level 2). Circle D is the component of the Algorithm Database that is used by the Intelligent Operating System and contains information about the execution characteristics of different parallel implementations of the algorithms. Each entry in Circle B may have multiple entries in Circle D, corresponding to different implementations. The Intelligent Operating System uses this information in selecting each algorithm implementation.

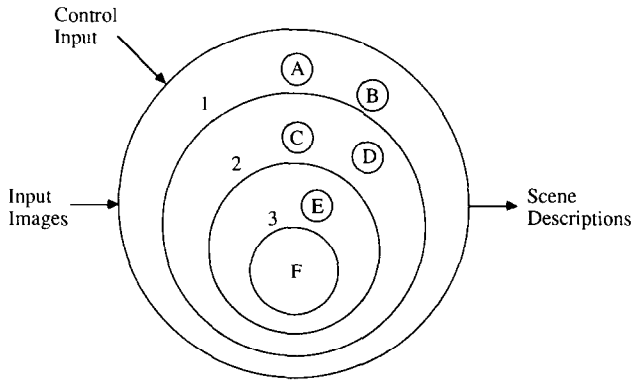

FIG. 2. Alternative view of the overall model with knowledge bases and algorithm databases grouped according to their levels of operation. Level 1, image understanding system (IUS); A, IUS knowledge base (task structure); B, IUS database (information of algorithm image analysis performance). Level 2, intelligent operating system (IOS); C, IOS knowledge base (reconfiguration and scheduling); D, IOS database (algorithm execution time as a function of resources). Level 3, low-level operating system; E, algorithm implementation encodings; F, reconfigurable parallel processing system.

Circle C represents the component of the Intelligent Operating System that provides the necessary information about the Reconfigurable Parallel Processing System to allow intelligent reconfiguration of resources for improved execution performance. This information includes knowledge of the system resources and their current status, and scheduling schemes. Decisions on system reconfiguration and the assignment of image analysis algorithms to partitions are then passed to the Low-level Operating System Routines. Circle E is the component of the Algorithm Database that is used by the Low-level Operating System Routines and contains the actual implementation codes for the algorithms. The three execution steps are therefore represented by Levels 1, 2, and 3; together Circles B, D, and E form the Algorithm Database shown in Fig. 1; Circles A and C form the Knowledge Base in Fig. 1.

There is a great deal of interaction among the Image Understanding System, the Intelligent Operating System, and the Algorithm Database. The Image Understanding System and the Algorithm Database could be extended to contain expert systems themselves. An important aspect of the model is that the Image Understanding System and the Intelligent Operating System are separate modules. Thus, despite the potential complexity of the complete system, there is a uniform, modular structure that allows incremental development of the various components. The strategies and overall structure of the Intelligent Operating System can be used in other application areas (such as speech understanding) by changing the Algorithm Database component. In the rest of this section, the major blocks of Fig. 1 are described in more detail.

2.1. *Image Understanding System*

An image understanding task is assumed to consist of many subtasks. The Image Understanding System contains information about which algorithms are used to perform a given subtask. Each subtask may be performed by more than one algorithm, where each algorithm has different image analysis performance characteristics which are stored as part of the algorithm in the Algorithm Database. The execution order of the subtasks may be represented as a data dependency graph, indicating which subtasks can be done simultaneously and which must be done sequentially with respect to the other subtasks. The exact structure and elements of the data dependency graph may vary during task execution on the basis of intermediate results that are derived. This data dependency graph is stored and maintained by the Image Understanding System.

An example of an image understanding task is shown in Fig. 3 to illustrate the types of data flow and control operations that are representative of these tasks. In particular, the execution time is nondeterministic when doing "edge linking" followed by "edge continuity checking." Also note that the processing has both a bottom–up and a top–down component. The top–down com-

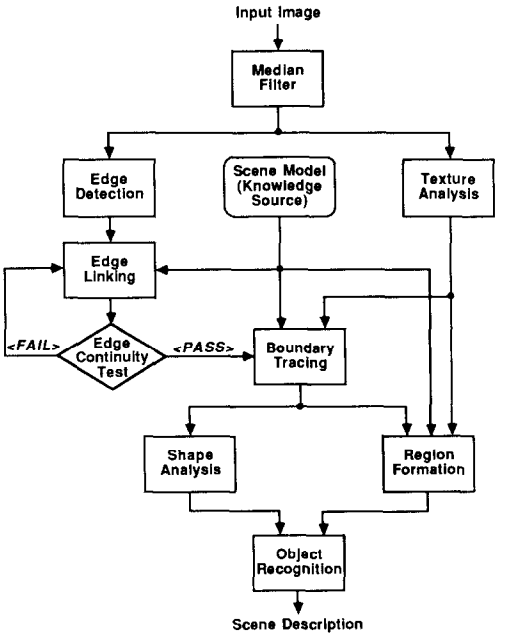

FIG. 3. A data dependency graph for a "typical" image understanding task scenario.

ponent (e.g., the use of a priori information) mainly consists of a scene model knowledge source that is used to drive the "edge linking," "boundary tracing," and "region formation" steps. The bottom–up component is used to drive the early vision steps of "median filtering," "texture analysis," and "edge detection." In the situation described in Fig. 3, it is important that the steps leading up to and including the "edge continuity test" be performed as quickly as possible, because the boundary tracing step requires this information before the rest of the processing can be completed. The Intelligent Operating System will have to recognize this and concentrate more system computation power to the steps leading up to the edge continuity step than to the texture analysis step. It should be noted that the type of processing occurring at the top of Fig. 3 is numeric, the type of processing occurring at the bottom is symbolic, and in between there is a mix of both.

2.2. *Algorithm Database*

The Algorithm Database contains the actual implementation codes and two levels of characteristics for each algorithm. The first (Circle B in Fig. 2) consists of algorithm performance. This contains image analysis information such as how a particular algorithm performs in the presence of noise. The

Image Understanding System interacts with the Algorithm Database to determine if a particular algorithm exists in the database and if any other algorithms that perform better in terms of their image analysis capabilities exist. The information the Image Understanding System uses to select an algorithm is based on image characteristics input with the image or derived during the execution of the task.

The next level (Circle D in Fig. 2) consists of information about parallel implementations of the algorithm. This will contain information such as how the input and output are distributed across the system's memories, expected execution time as a function of the number of processors used, and interprocessor network communication requirements. An algorithm may have multiple entries in the database for this characteristic level corresponding to the existence of several parallel implementations of that particular algorithm. The implementation that is most appropriate for performing a given subtask is determined by the types of processing that were done prior to the current step, the type of processing that is to be performed next, and the system constraints and resources available at that time. Table I shows an example of information about four alternative implementations of an edge detection algorithm [22].

2.3. *Intelligent Operating System*

The goal of a reconfigurable large-scale parallel processing system is to adapt the system state (machine configuration) to maximize some performance criteria. The performance criterion assumed here is execution speed of the total task. The objective is to use the Algorithm Database described in Section 2.2 to reconfigure system resources to maximize task execution speed or, equivalently, to minimize system response time. The factors that contribute to the response time for a given task include the execution time of the component image processing/analysis algorithms, the execution time of the Image Understanding System and Intelligent Operating System, and the time to reconfigure the state of the parallel processing system. In this subsection, the Intelligent Operating System and the target Reconfigurable Parallel Processing System are described.

Reconfigurable large-scale parallel processing systems can be constructed in different ways (e.g., TRAC [18], DCA [11]). A particular architecture with given reconfiguration parameters is being considered initially to make the overall model presented here tractable. For this purpose, PASM [19] is being used as the model of the Reconfigurable Parallel Processing System in Fig. 1. The overall model can be applied to other parallel systems.

The PASM design includes 1024 sophisticated processors in its computational engine and has many (e.g., 70) processors for operating system sup-

TABLE I
IMPLEMENTATION DATA FOR FOUR IMPLEMENTATIONS OF AN EDGE DETECTOR

Parameter	Implementation I1	Implementation I2
Mode	SIMD	MIMD
No. PEs	$\leq \frac{\text{No. pixels}}{64}$	No. pixels
Time	$\frac{6*\text{image\_size}}{\text{No. PEs}} + 4*\text{No. PEs}$	$\frac{4*\text{image\_size}}{\text{No. PEs}} + \frac{\text{image\_size}}{64}$
Input format	1 byte/pixel	1 byte/pixel
Output format	Edge list	Edge list
Input allocation	Regions	Regions
Output allocation	Regions	Regions
	I3	**I4**
Mode	SIMD	MIMD
No. PEs	$\leq \frac{\text{No. pixels}}{64}$	No. pixels
Time	$\frac{6*\text{image\_size}}{\text{No. PEs}}$	$\frac{4*\text{image\_size}}{\text{No. PEs}} + \frac{\text{image\_size}}{64}$
Input format	1 byte/pixel	1 byte/pixel
Output format	Binary image	Binary image
Input allocation	Regions	Regions
Output allocation	Regions	Regions

Note. The *Time* parameter is the expected execution time of the implementation in terms of the image size and the number of PEs allocated to the partition. The *Input allocation* and *Output allocation* parameters specify how the data is split up among the PEs when the algorithm begins and ends execution.

port (e.g., memory management, file directory maintenance for the multiple secondary storage devices, and SIMD control unit functions). A 30-processor prototype of PASM is currently operational [20]. The relevant features of PASM's computational engine needed as background for the following discussion include:

1. A system with 1024 processors is assumed.
2. All processors are the same (e.g., MC68000-family processors).
3. Each processor is paired with a memory module and I/O, forming a *processing element (PE)*. When one PE sends data to or requests data from another PE, the system is said to be operating in a *PE-to-PE* configuration (i.e., each processor has its own local memory). Network interfaces will also allow each processor to access another processor's memory module (almost)

directly. This mode of operation is referred to as the *processor-to-memory* configuration (i.e., the processors share a common set of memory modules).

4. The PEs in the system can be dynamically partitioned, under software control, into independent groups forming independent virtual machines of various sizes.

5. A multistage network is used to provide communications among the PEs. This network can be dynamically reconfigured under software control to be partitioned into independent subnetworks (to support independent virtual machines) and to perform a great variety of connection patterns, for both "local" and "global" communications.

6. The PEs in a virtual machine can operate in either SIMD or MIMD modes and can dynamically switch modes under software control.

One example of how the reconfiguration capabilities of PASM can be exploited is given in [12], where one approach to contour extraction in gray scale images is examined. A brief simplified summary is as follows. Each PE is assigned a checkerboard pattern subimage that is processed in three main phases: edge-guided thresholding, local contour tracing, and complete contour tracing. The edge-guided thresholding involves generating a Sobel image and using it with characteristics of the original image to select a threshold value. This phase is executed most efficiently in the SIMD mode, with the PE-to-PE configuration, and eight nearest-neighbor inter-PE network communication patterns. The local contour tracing involves each PE tracing contours in its subimage, both complete and partial (i.e., contours which span multiple subimages), and generating a symbolic representation of the contours. This phase is executed most efficiently in the MIMD mode with the PE-to-PE configuration (no inter-PE communications are required). Finally, the complete contour tracing phase combines the symbolic representations of partial contours that cross subimage boundaries to form complete contours. This is done most efficiently in the MIMD mode, with the processor-to-memory configuration and variable global access patterns from the processors to the memory modules.

The configuration of PASM at any given point of time, the status of any jobs executing or awaiting execution, and the memory contents determine the *system state.* The parameters in the state space include the number of virtual machines and the size of each (in terms of number of computational engine processors assigned to the virtual machine), the status of the algorithm executing on each virtual machine (e.g., execution time expended, amount of working memory consumed), the performance/system-requirements characteristics of all algorithms executing or awaiting execution (e.g., relationship of execution speed to number of processors used in the virtual machine, expected execution time, expected memory requirements, data allocation scheme among the processors of the virtual machine for both input

data and output data),[1] the processing mode (SIMD or MIMD) of each virtual machine (which can vary dynamically at execution time), and the interprocessor connectivity (interprocessor communication patterns) of each virtual machine (which can vary dynamically at execution time).

The Intelligent Operating System is responsible for keeping track of the system state. Most importantly, it determines many of the parameters, such as selecting which parallel implementation of an algorithm to use to perform a given subtask, scheduling algorithms for execution, choosing the size of the virtual machine for a given algorithm (how many PEs), and assigning algorithms to virtual machines (which PEs). (The Intelligent Operating System can modify the current allocation of resources to an algorithm being executed if it deems it appropriate for improved overall system performance of the complete task.) The Intelligent Operating System performs these functions using information from the Image Understanding System (data dependency graphs for subtasks, algorithms available to perform a given subtask), from the Algorithm Database (the algorithm system-requirements characteristics), from each virtual machine's master control unit (e.g., algorithm execution status, such as time and space consumed, expected time to completion, and any significant intermediate results of the computation), and from its own knowledge of the current system state.

In addition, the Intelligent Operating System has information about the execution characteristics of the Low-Level Operating System routines, allowing it to determine the time required to perform a system reconfiguration. The types of reconfiguration information that the Intelligent Operating System uses include how the PEs can be grouped (based on network topology) [9] and the time needed to migrate tasks to perform a reconfiguration [17]. The goal of the Intelligent Operating System is to assimilate all of this information and use it, whenever appropriate, to generate new system states that will optimize system performance of the task under execution. The resource management role of the Intelligent Operating System is a standard function of any operating system. However, on a reconfigurable parallel system, this job is significantly more involved than on a less flexible system. As described in the next section, an expert system is used to perform the decision-making necessary to select an algorithm implementation and assign resources on the basis of a diverse set of information. Hence the name *Intelligent Operating System*.

There are additional issues in reconfigurable parallel system design that can be incorporated into our model as extensions to the above functionality

[1] This information about data allocation is important when juxtaposing algorithms to perform a complete task; i.e., the output data allocation of one algorithm will become the input data allocation of another, and this may affect the choice of algorithms and/or the need to restructure the data.

requirements for the Intelligent Operating System. These include reconfiguring for fault tolerance, using data dependency graph look-ahead when scheduling algorithms to perform subtasks, assigning measures of relative importance to the speed of execution of different subtasks based on their practical importance in a real-time processing environment, and "concentrating" computational power to enhance the execution speed of a subtask of high importance.

3. Expert Systems

Expert systems are used in many components of the overall model: from performing image understanding routines to selecting algorithms and hardware configurations. Expert systems store, select, and process fragments of knowledge about a specific task in a reasoning process designed to arrive at an acceptable solution. These fragments of knowledge are represented as rules and facts that describe relationships between possible true states (or facts) and characteristics of the problem associated with these states. The capabilities of expert systems appear to be well matched to the types of decision making that must be performed in the model.

A new expert system language has been developed to provide efficient support for the diversified needs of the expert systems in our model, specifically, the ability to deal with both numeric and symbolic processing and to perform algorithm and hardware configuration selections. Knowledge fragments are grouped in terms of rule sets. A rule set consists of specific knowledge required to solve a particular type of problem. The syntax for defining this expert rules language, known as Rule Set Language (RSL), is given in Section 3.1. An example is given in Section 4 to demonstrate how RSL can be used to make a decision about the sequence of image analysis operations that should be performed. The segregation of knowledge into different rule sets lends itself naturally to parallel execution. Because the only possible interaction between rule sets is via the CONSULT command (see Section 3.2), different rule sets can be run concurrently.

3.1. *Rule Set Language Syntax*

Expert systems methods are exploited to represent knowledge of hardware reconfiguration and algorithm selection for image understanding. The expert system language, as described here, serves the purpose of presenting a prototype environment in which concepts and techniques of expert systems can be integrated into the framework of the Image Understanding System and the Intelligent Operating System. Hence, the specification of the syntax of an expert system that is oriented toward the type of problem solving in these tasks is necessary. In a sense, one can consider the language introduced here

as a tool for the development of an image understanding environment capable of capturing the knowledge of a human being in the identification and description of a scene and in the reconfiguration of computer hardware.

The language is basically a collection of syntactic entities called rule sets. A rule set consists of the specific knowledge required to solve a particular problem. Moreover, rule sets can be joined together implicitly in order to tackle problems that involve expertise from a number of areas. The syntax of a rule set is defined as

```
RULE SET ⟨rule-set-name⟩ READ ⟨read-codes⟩ WRITE ⟨write-codes⟩
        EXECUTE ⟨execute-codes⟩
INITIALIZATION ⟨command⟩
GOAL ⟨variable⟩ FOR ⟨variable conditions⟩ DO ⟨command⟩. . .
{RULE ⟨rule-name⟩ PRIORITY ⟨priority-level⟩ COST ⟨action-cost⟩
        READ ⟨read-codes⟩ WRITE ⟨write-codes⟩
IF ⟨condition⟩
        THEN ⟨command⟩
        USING ⟨decision variables⟩ COMMENT ⟨ascii characters⟩
        }
CONTEXT ⟨variables⟩. . .
```

There are four parts in a rule set declaration. The first part defines the name of the rule set ⟨rule-set-name⟩ and is a unique identification tag of a rule set for both internal operation and external inspection. This section also provides security control functions by allowing the creator of the rule set to specify the authority level for reading, writing, and executing the rule set. The ⟨read-codes⟩, ⟨write-codes⟩, and ⟨execute-codes⟩ are optional; omitting any of these implies public access is allowed for that operation on the rule set.

Unlike the first section, which is compulsory for all rule sets, the INITIALIZATION section is optional. If the INITIALIZATION is present, its commands are executed in sequence when the rule is invoked. These commands may be performed to establish initial variable values by assignment, to retrieve additional information, to interact with the end user, to perform computations for certain variables, and to establish communication links with other systems.

The mandatory GOAL section identifies the variable whose value will be inferred when the rule set is invoked. The optional FOR clause specifies conditions (e.g., integrity conditions) that must be satisfied when this goal variable's value has been determined in order for that value to be considered valid. The optional DO clause consists of a list of commands that will be automatically executed if the goal is met.

The last part of a rule set is the specification of individual rules. One or more RULE sections must be present. Each rule is given a unique name, an optional priority level, and an optional action cost. The function of the prior-

ity level is to provide a measure of importance or confidence of the rule, while the relative processing cost of the rule's action is reflected by the action cost.

A rule's IF clause consists of any permissible logical expression composed of one or more conditions connected by Boolean operators. A condition clause can be either an assertion (e.g., number-of-vertices < 10) or a query statement (e.g., whether an object-shape can be retrieved from object-library where object-name = car).

A rule's THEN clause consists of a sequence of commands which will be executed when the inference engine determines that the rule's premise is true. These commands can include not only assignment statements but also procedure invocation, rule set invocation, input statements, output statements, graphics, computations, and data retrieval. The USING clause identifies one of the variables whose value could be altered by the rule's action as the rule's decision variable. The inference engine examines a rule's decision variable in the course of inference in order to decide whether that rule is currently applicable as a candidate for backward chaining. The optional COMMENT clause contains text that explains the nature of the rule.

A rule set's optional CONTEXT section identifies those variables (in addition to decision variables) whose values will be preserved at points where backtracking could occur in an inference process. Implicit input rules exist for all condition variables in a rule set's rules.

3.2. *Rule Set Invocation*

A rule set can be invoked via the CONSULT command whose syntax is

```
CONSULT ⟨rule-set-name⟩ [TO SEEK ⟨decision-variable⟩. . .]
        [FOR ⟨condition⟩. . .] [DO
        ⟨command⟩. . .]
```

The optional SEEK clause is used if some decision variables other than the rule set's goal variable are desired. The optional FOR clause is used if some goal conditions other than those specified in the rule set's GOAL section are desired. The optional DO clause is used if some goal actions other than those stated in the rule set's GOAL section are desired. The CONSULT command utilizes the inference engine's backward chaining approach to inference.

A variation of the CONSULT command uses a forward chaining approach:

```
CONSULT ⟨rule-set-name⟩ TO TEST ⟨decision-variable⟩. . . [DO
        ⟨command⟩. . .]
```

Here the rule set is used to determine the value of the decision variable, as implied by the present context. The optional DO clause specifies commands

to be executed if the consultation resulted in a change in the decision variable's value. This generative use of a rule set can be used to test the present decision variable context.

4. Execution of an Image Understanding Task

A very simple example image understanding task is presented to illustrate the application of expert systems and the concept of intelligent scheduling on the parallel processor by the Intelligent Operating System. Most image understanding systems employ a top–down approach in the identification of individual objects and their spatial relationships in a given scene. In general, the process of object identification is a hypothesis-verifying process. Hypotheses concerning various properties of the object are set up in conjunction with the scenario given. A verifying procedure is then invoked to test the validity of the hypotheses. During the hypothesis-verifying process, new information may be aggregated into or deleted from the current hypothesis. More subhypotheses may also be generated for testing. Conventional programming environments are not appropriate for accomplishing this task in a cost effective manner. An example is presented below to illustrate how the RSL as described in the previous section is used to verify a simple hypothesis.

4.1. *Example Image Understanding Task*

This example is concerned with the identification of a cube in a three-dimensional space (see Fig. 4). Two rule sets are used: CUBE-HYPO-VERIFY and CAMERA-POSITION. The former rule set focuses on the verification of a cube in space on the basis of two camera images of the object. The latter is mainly concerned with the generation of two new camera positions on the basis of the given hypothesis of the shape of the object. What follows is a brief description of the two rule sets; the actual rule sets listing is in the Appendix. For simplicity, all priority, cost, read, write, and execute entries are left blank. However, this is not true in general; all these entries do bear some significance in the efficiency and security of the code.

In the INITIALIZATION section of CUBE-HYPO-VERIFY, CAMERA-POSITION is consulted to generate two new camera positions. Two camera images are then taken in these two positions and processed. The routine PREPROCESS will take in a camera image and produce a processed image that can be recognized and used by the system internally. The goal of CUBE-HYPO-VERIFY is to check whether the object is a cube or not. The decision variable used by CUBE-HYPO-VERIFY is CUBE-HYPO and can take on any of the values {true, false, nil}. If the goal is satisfied (i.e., CUBE-HYPO is either true or false), then the conclusion of this test (i.e., the object is a

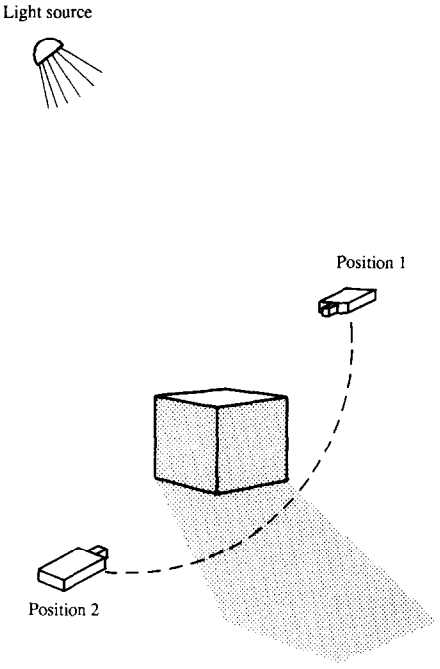

FIG. 4. A model of the sensor geometry for the image understanding task example in Section 4.

cube or not) is returned. The three rules in the RULE section use the certainty factors from the two camera positions to decide how to proceed.

The second rule set, CAMERA-POSITION, is called to determine the next two appropriate positions for the camera in verifying the shape of an object. In the INITIALIZATION section of CAMERA-POSITION, a routine called VERTEX-DETECTION is invoked to extract the number of vertices of the object. If the maximum number of vertices that the object can have from any angle of view is not known, then retrieve this piece of information from the knowledge base. POSITION1 and POSITION 2 are the two decision variables of CAMERA-POSITION and are unknown initially. This rule set is capable of dealing with a large number of object hypotheses; for illustrative purposes, only the first three rules that are concerned with cubic objects are shown.

The three rules in CUBE-HYPO-VERIFY and the first three rules in CAMERA-POSITION provide the knowledge which is sufficient, if not complete, in the identification of a cube in an unobstructed three-dimensional space. This example does not serve to reveal the technical implementation details of an image understanding system but rather to provide insights into

how an expert systems paradigm is aggregated into the design and construction of an image understanding system.

4.2. *The Intelligent Operating System*

The role of the Intelligent Operating System can be seen, in part, by further analysis of the execution of this example. Two of the routines called by the rule sets (PREPROCESS called from CUBE-HYPO-VERIFY and VERTEX-DETECTION called from CAMERA-POSITION) entail intensive processing. Routines such as these are therefore prime candidates for intelligent scheduling on the parallel processor by the operating system in order to achieve reduced execution times.

The routine PREPROCESS accepts as input a gray scale image and produces as output a symbolic description of that image. A typical algorithm for this task might be [14]

(1) Extract textural regions using a 3 by 3 Laplacian window.
(2) If every region can be described by its texture, proceed to the symbolic processing (step 4).
(3) While some unprocessed region has an area of greater than 50 pixels
  (3a) Compute the histogram of the region.
  (3b) If the histogram is unimodal
    (3b.1) If the area of the region is less than 1537, then go on to the next region.
    (3b.2) Do a window scanning histogram.
    (3b.3) If the histogram is still unimodal, then go on to the next region.
  (3c) Else the histogram has multiple peaks. Split the region into multiple new regions.
  (3d) If the region area is less than 8, ignore that region.
(4) Generate a symbolic description of the segments.

Many versions of each individual routine used in the task exist in the IOS portion of the Algorithm Database (Circle D of Fig. 2). For instance, there might be three versions of the Laplacian operator: two SIMD versions (one with data allocated by regions and one with data allocated by rows) and one MIMD version. Also of importance is the existence of decision, or branching, points in the algorithm. This makes the execution characteristics of the algorithm impossible to determine a priori. The algorithm for the VERTEX-DETECT has similar characteristics.

The Intelligent Operating System will have to make two types of decisions at Level 2 of Fig. 2. It will first have to determine an initial schedule and configuration for the parallel processor on the basis of fitting the available resources to the predicted runtimes and PE usages of the particular implementation of the algorithm. Second, the Intelligent Operating System must monitor the progress of the algorithm and update the schedule and configuration on the basis of actual execution times and decision point results. The overall execution time of the given task can therefore be reduced by allowing the Intelligent Operating System to make dynamic scheduling decisions.

This method will provide an obvious improvement over rigidly specifying the subtask list on the basis of general performance characteristics.

5. Algorithm Prototyping and System Prototyping Using the Model

There are two types of prototyping in which a user interacts with the image understanding environment. In *algorithm prototyping,* the user executes a particular algorithm using different sets of data to examine the performance of the algorithm. In this section, an *algorithm* refers to a task-independent routine; for example, the 3 by 3 Laplacian operator in the example of Section 4 is an algorithm. In *system prototyping,* the user can test different strategies for choosing algorithms, imposing execution order constraints, and integrating results from various algorithms. The output of system prototyping is typically task-specific and composed of algorithms. The task example in Section 2 shown in Fig. 3 is one such system. It is assumed throughout Section 5 that the IUS serves only as an interface between the user and the Intelligent Operating System, and, in some cases, between the user and the Algorithm Database (see Fig. 5).

5.1. *Algorithm Prototyping*

Before a new algorithm is tested, the implementation code (Circle E in Fig. 2) and information about parallel implementations (Circle D in Fig. 2)

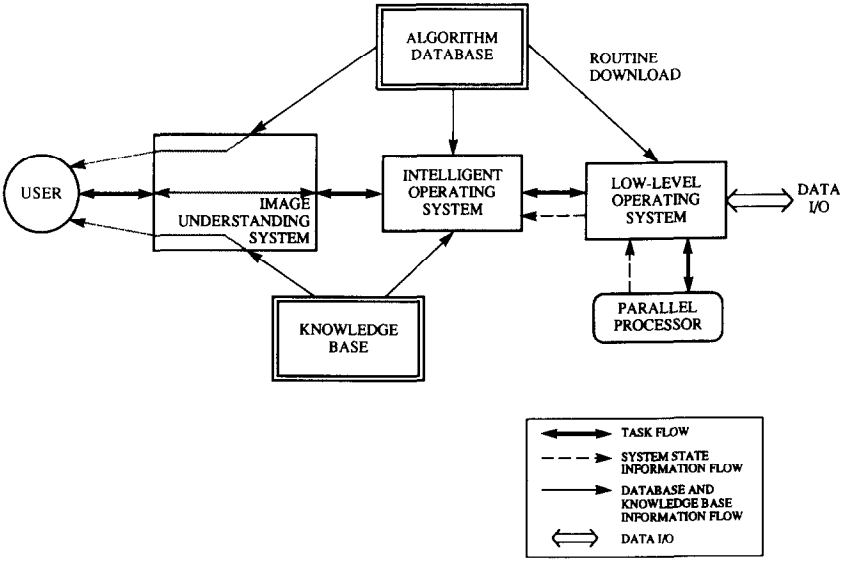

FIG. 5. Overall model of system prototyping and algorithm prototyping by a user.

of the algorithm must be supplied. One issue that arises is to determine when and to what extent in the prototyping process the Intelligent Operating System tools should assist the user in providing the information to the Algorithm Database.

Developing the implementation code, or programming a parallel computer, is not easy; a detailed discussion of the variety of approaches [3, 5–8, 10, 15] is beyond the scope of this paper. The current state in the development of parallelizing compilers does not allow programmers to be as removed from the hardware as in the case of serial computers [4, 13, 21]. For an entirely new algorithm, the parallel code in the Algorithm Database either is explicitly parallel code or is generated by a parallelizing compiler.

An alternative to generating the code and determining the parallel characteristics every time an algorithm is entered into the Algorithm Database is to exploit the knowledge the system possesses about parallel implementations of algorithms. This knowledge is in the form of the code and information about each algorithm stored in the Algorithm Database. An operation based on expert system concepts, referred to as *cloning,* allows a starting point for algorithm development. Through an interactive interface, the cloning process would ask the user to note which algorithm in the current database most closely resembles the algorithm that is to be added. The user would be prompted for further information required by the system about the algorithm. By using a set of features that characterizes parallel algorithms [8], the steps that the cloning process should take to make the necessary changes can be stated explicitly in rules. Hence, the cloning process can build a new entry in the Algorithm Database by modifying an existing one. Approaches to implementing this are currently under study.

5.2. *System Prototyping*

System prototyping is the process by which algorithms to perform an image understanding task for a particular situation are selected and the data flow among these algorithms is chosen. Consider the example previously discussed in Section 2.1 (Fig. 3). During the system prototyping process, the user can, for example, experiment with either feeding the image directly to texture analysis without median filtering or performing texture analysis after median filtering. The user can also choose an edge detector algorithm depending on the specific task situation. All these are facilitated by the model in that the Algorithm Database provides a set of tools for the user while the Intelligent Operating System frees the user from having to interact with the hardware directly. Since the user (at Level 1 in Fig. 2) does not have to choose the actual implementation for carrying out each subtask, one direct consequence is that an image understanding system developed under this model will be machine independent. The output of this rapid prototyping will be a final algorithm sequence for a particular situation that is stored as

part of the Image Understanding System so that it can be called upon in the future to execute a similar image understanding task.

6. Summary

The conceptual model that was described involves the following aspects of parallel processing, image understanding, and expert systems research:

—Design of an Intelligent Operating System for a reconfigurable parallel computer system and an Image Understanding System, focusing on incorporating intelligence in (1) the automatic selection of the algorithms to be used to perform an image understanding task and (2) the selection of appropriate architecture configurations for execution of the algorithms, with performance requirements driving both selection processes.

- Exploiting the flexibility of a reconfigurable parallel architecture by providing a very powerful Intelligent Operating System.
- Developing a database of information about image understanding algorithms, including information about both their image analysis properties (for algorithm selection) and their execution characteristics (for algorithm implementation and architecture configuration selection).
- Exploring the process by which algorithms are selected to accomplish an image understanding task, where the analysis "so far" is used in deciding what algorithms to use next.

—Use of expert systems in multiple areas of the Intelligent Operating System.

- Focusing on the use of intelligence in the operating system.
- Employing a uniform structure throughout the Intelligent Operating System, even though it makes decisions of many different types.
- Using expert systems in a control role for invoking other expert systems and the image analysis algorithms which are ultimately executed.
- Exploring the use of new expert systems development tools.

In summary, a model for an Intelligent Operating System that can make efficient use of reconfigurable parallel architectures has been presented. Characteristics of the Intelligent Operating System and the overall model have been illustrated by considering an image understanding task example and scenarios of a user interacting with the operating system. Current work involves the detailed implementations of this model, using the PASM simulators and prototype as validation tools [23]. The concepts underlying the methodologies employed by the Image Understanding System and Intelligent Operating System can be abstracted so that they can be incorporated into other reconfigurable large-scale parallel processing systems, as well as other problem domains.

APPENDIX

Rule Set Listing for the Image Understanding Task Example in Section 4

```
rule: set CUBE-HYPO-VERIFY read write execute

initialization
    consult CAMERA-POSITION to test POSITION1, POSITION2
    call MOVECAMERA(POSITION1)
    call IMAGE(POSITION1, IMAGE1)
    call MOVECAMERA(POSITION2)
    call IMAGE(POSITION2, IMAGE2)
    call PREPROCESS(IMAGE1, PRE-IMAGE1)
    call PREPROCESS(IMAGE2, PRE-IMAGE2)
    CUBE-HYPO = nil

goal CUBE-HYPO < > nil do OBJECT-SCENT(OBJECT,CUBE-HYPO)

{ rule CHV1 priority cost read write
  if (MATCH(PRE-IMAGE1, CUBE, CF1)) & (MATCH(PRE-IMAGE2, CUBE, CF2))
      (min(CF1,CF2) >= 0.7)
then
  (CF = min(CF1,CF2)
  (CUBE-HYPO = true)
using CUBE-HYPO
comment if the certainty factors that the object is a cube as
        estimated from the two preprocessed images are known and
        the minimum of which is larger than 0.7, then we conclude
        that the object is a cube with certainty factor equal to
        their minimum. }

{ rule CHV2 priority cost read write
  if ((MATCH(PRE-IMAGE1, CUBE, CF1)) & (CF1 > 0.2) & (CF1 < 0.7)) or
      ((MATCH(PRE-IMAGE2, CUBE, CF2) & (CF2 > 0.2) & (CF1 < 0.7))
THEN
then
  (consult CUBE-HYPO-VERIFY)
using CF1, CF2
comment if either or both of the certainty factors is between 0.2
        and 0.7, then recursively consult CUBE-HYPO-VERIFY for
        further clarification. }

{ rule CHV3 priority cost read write
  if (((MATCH(PRE-IMAGE1, CUBE, CF1)) & (CF1 <= 0.2)) or
      (((MATCH(PRE-IMAGE2, CUBE, CF2)) & (CF2 <= 0.2))
then
      (CUBE-HYPO = false)
      (CF = max(CF1, CF2))
using CUBE-HYPO
comment if either or both of the certainty factors is equal to or
        below 0.2, then we conclude that the object is not a cube
        with certainty factor equal to the maximum of the two
        factors. }
```

```
rule: set CAMERA-POSITION priority read write execute
initialization

    call VERTEX-DETECTION(OBJECT, NUM-OF-VERTICES)
    if OBJECT-MAX-VERTICES = nil then
        OBJECT-MAX-VERTICES = (retrieve MAX-VERTICES from
                    OBJECT-DES-FILE where SHAPE = OBJ-HYP)


goal POSITION1, POSITION2

{ rule CP1 priority cost read write
  if (OBJ-HYP = CUBE) & ((NUM-OF-VERTICES > 7) or
                (NUM-OF-VERTICES < 4))
then
  (POSITION1 = nil)
  (POSITION2 = nil)
using POSITION1, POSITION2
comment if the maximum number of vertices as detected is larger
        than 7 or less than 4 and the object-hypothesis is a
        cube, then no new camera positions is required for
        further clarification (i.e., the object is probably not
        a cube). }

{ rule CP2 priority cost read write
  if (OBJ-HYP = CUBE) & (NUM-OF-VERTICES >= 4) &
    (NUM-OF-VERTICES < 7)
then
    (POSITION = move(random-deg(POSITION), random-plane(POSITION)))
    (consult CAMERA-POSITION for OBJ-HYP = CUBE,
                    OBJECT-MAX-VERTICES = 7)
using POSITION
comment if the object-hypothesis is a cube and the maximum
        number of vertices detected is between 4 and 7, then
        randomly generate a new position for the camera and
        consult CAMERA-POSITION again for a better view. }
{ rule CP3 priority cost read write
  if (OBJ-HYP = CUBE) & (NUM-OF-VERTICES = 7)
  then
    (POSITION1 = POSITION)
    (POSITION2 = move(180, orthogonal))
using POSITION1, POSITION2
comment if the object-hypothesis is a cube and the maximum
        number of vertices detected is exactly 7 then the
        first camera position is the current position and
        the second camera position is the mirror image of
        the first position. }
```

ACKNOWLEDGMENTS

The Fig. 2 representation of the model was suggested by Thomas Schwederski. Preliminary versions of portions of this paper were presented at the IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management (1985) and the Second International Conference on Supercomputing (1987).

REFERENCES

1. Berman, F., and Snyder, L. On mapping parallel algorithms into parallel architectures. *J. Parallel Distrib. Comput.* **4,** 5 (Oct. 1987), 439–458.
2. Delp, E. J., Mudge, T. N., Siegel, L. J., and Siegel, H. J. Parallel processing for computer vision. *Robot Vision, Proc. of SPIE,* May 1982, Vol. 336, pp. 161–167.
3. Dietz, H., and Klappholz, D. Refined C: A sequential language for parallel programming. *1985 International Conf. on Parallel Processing,* Aug. 1985, pp. 442–449.
4. Duff, M. J. B. Parallel algorithms and their influence on the specification of application problems. In Preston, K., Jr., and Uhr, L. (Eds.). *Multicomputers and Image Processing Algorithms and Programs.* Academic Press, New York/London, 1982, pp. 261–274.
5. Gelernter, D. Domesticating parallelism. *Computer* **19** (Aug. 1986), 12–19.
6. Gemmar, P. Parallel solutions for conventional image algorithms. In Duff, M. J. B. (Ed.). *Intermediate-Level Image Processing.* Academic Press, New York/London, 1986, pp. 83–100.
7. Hummel, R. Connected component labelling in image processing with MIMD architectures. In Duff, M. J. B. (Ed.). *Intermediate-Level Image Processing.* Academic Press, New York/London, 1986, pp. 101–127.
8. Jamieson, L. H. Characterizing parallel algorithms. In Jamieson, L. H., Gannon, D. B., and Douglass, R. J. (Eds.). *The Characteristics of Parallel Algorithms.* MIT Press, Cambridge, MA, 1987, pp. 65–100.
9. Jeng, M., and Siegel, H. J. Dynamic partitioning in a class of parallel systems. *Eighth International Conf. on Distributed Computing Systems,* June 1988, pp. 33–40.
10. Jordan, H. F. The Force. In Jamieson, L. H., Gannon, D. B., and Douglass, R. J. (Eds.). *The Characteristics of Parallel Algorithms.* MIT Press, Cambridge, MA, 1987, pp. 395–436.
11. Kartashev, S. I., and Kartashev, S. P. A multicomputer system with dynamic architecture. *IEEE Trans. Comput.* **28** (Oct. 1979), 704–720.
12. Kuehn, J. T., Siegel, H. J., Tuomenoksa, D. L., and Adams, G. B., III. The use and design of PASM. In Levialdi, S. (Ed.). *Integrated Technology for Parallel Image Processing.* Academic Press, New York/London, 1985, pp. 133–152.
13. Lusk, E. L., and Overbeek, R. A. A minimalist approach to portable, parallel programming. In Jamieson, L. H., Gannon, D. B., and Douglass, R. J. (Eds.). *The Characteristics of Parallel Algorithms.* MIT Press, Cambridge, MA, 1987, pp. 351–362.
14. Ohta, Y. *Knowledge-Based Interpretation of Outdoor Natural Color Scenes.* Pitman, Marshfield, MA, 1985.
15. Potter, J. L. MPP architecture and programming. In Preston, K., Jr., and Uhr, L. (Eds.). *Multicomputers and Image Processing Algorithms and Programs.* Academic Press, New York/London, 1982, pp. 275–289.
16. Rice, T. A., and Jamieson, L. H. Parallel processing for computer vision. In Levialdi, S. (Ed.). *Integrated Technology for Parallel Image Processing.* Academic Press, New York/London, 1985, pp. 57–78.

17. Schwederski, T., Siegel, H. J., and Casavant, T. L. A model of task migration in partitionable parallel processing systems. *Frontiers '88: The Second Symp. on the Frontiers of Massively Parallel Computation,* Oct. 1988.

18. Sejnowski, M. C., Upchurch, E. T., Kapur, R. N., Charlu, D. P. S., and Lipovski, G. J. An overview of the Texas reconfigurable array computer. *National Computer Conf.,* June 1980, pp. 631–641.

19. Siegel, H. J., Siegel, L. J., Kemmerer, F., Mueller, P. T., Jr., Smalley, H. E., Jr., and Smith, S. D. PASM: A partitionable SIMD/MIMD system for image processing and pattern recognition. *IEEE Trans. Comput.* **30** (Dec. 1981), 934–947.

20. Siegel, H. J., Schwederski, T., Kuehn, J. T., and Davis, N. J., IV. An overview of the PASM parallel processing system. In Gajski, D. D., Milutinovic, V. M., Siegel, H. J., and Furht, B. P. (Eds.). *Computer Architecture.* IEEE Computer Society Press, Washington, DC, 1987, pp. 387–407.

21. Uhr, L. *Algorithm-Structured Computer Arrays and Networks.* Academic Press, New York/London, 1984.

22. Weil, F. J., Jamieson, L. H., and Delp, E. J. Some aspects of an image understanding database for an intelligent operating system. *1987 IEEE Workshop on Computer Architecture for Pattern Analysis and Machine Intelligence,* Oct. 1987, pp. 203–208.

23. Weil, F. J., Jamieson, L. H., and Delp, E. J. DISC: A method for dynamic intelligent scheduling and control of reconfigurable parallel architectures. Tech. Rep. TR-EE 89-8, Purdue Univ., School of Electrical Engineering, Feb. 1989.

CHEE-HUNG HENRY CHU received a B.S.E. (summa cum laude) degree in computer engineering and a M.S.E. degree in computer, information, and control engineering from the University of Michigan, Ann Arbor, in 1981 and 1982, respectively, and a Ph.D. in electrical engineering from Purdue University, West Lafayette, Indiana, in 1988. Dr. Chu is currently an assistant professor of computer engineering at the Center for Advanced Computer Studies of the University of Southwestern Louisiana, Lafayette. His current research interests include applying combinatorial optimization methods to signal processing and computer vision problems. He is a member of the IEEE, the Optical Society of America, the Association for Computing Machinery, Tau Beta Pi, and Eta Kappa Nu.

EDWARD J. DELP was born in Cincinnati, Ohio. He received B.S.E.E. (cum laude) and M.S. degrees from the University of Cincinnati and a Ph.D. from Purdue University, West Lafayette, Indiana. From 1980 to 1984, Dr. Delp was Assistant Professor of Electrical and Computer Engineering at The University of Michigan, Ann Arbor. Since August 1984, he has been Associate Professor of Electrical Engineering at Purdue University. His research interest lies in the broad area of communication and information theory, especially the transmission and processing of multidimensional signals. Dr. Delp is a member of Tau Beta Pi, Eta Kappa Nu, Phi Kappa Phi, Sigma Xi, the Optical Society of America, and the Pattern Recognition Society, and is a senior member of the IEEE. He is a member of the editorial board of the *International Journal of Cardiac Imaging.* He is also coeditor of the book *Digital Cardiac Imaging* published by Martinus Nijhoff.

LEAH H. JAMIESON received an S.B. degree in mathematics from MIT and M.A., M.S.E., and Ph.D. degrees in electrical engineering and computer science from Princeton University. She is currently Professor of Electrical Engineering at Purdue University. Her research interests include parallel algorithms; the application of parallel processing to speech, image, and signal processing; and speech recognition. She has co-edited the books *Algorithmically-Specialized Parallel Computers* (Academic Press, 1985) and *The Characteristics of Parallel Algorithms*

(MIT Press, 1987) and has been on the editorial board for the *IEEE Transactions on Acoustics, Speech, and Signal Processing,* the *Journal of Parallel and Distributed Computing,* and *The Journal of VLSI Signal Processing.* She is a member of the Advisory Committee for the NSF Division of Microelectronics Information Processing Systems.

H. J. SIEGEL received two B.S. degrees from the Massachusetts Institute of Technology (MIT), and M.A., M.S.E., and Ph.D. degrees from Princeton University. He is Professor and Director of the PASM Parallel Processing Project in the School of Electrical Engineering at Purdue University. He has coauthored over 120 technical papers, coedited four volumes, authored one book (*Interconnection Networks for Large-Scale Parallel Processing*), consulted, given tutorials, and prepared video tape courses, all on parallel processing networks and systems. He was General Chairman of the "Third International Conference on Distributed Computing Systems" (1982), Program Co-chairman of the "1983 International Conference on Parallel Processing," and General Chairman of the "Fifteenth Annual International Symposium on Computer Architecture" (1988).

FRANK J. WEIL was born in Evanston, Illinois, on January 25, 1961. He received B.S.E.E. (with distinction), M.S.E.E., and Ph.D. degrees from Purdue University, West Lafayette, Indiana, in 1982, 1984, and 1988, respectively. Dr. Weil is currently working in the Software Engineering Research Lab, Communications Systems Research, Motorola Inc., Schaumburg, Illinois. His research interests include artificial intelligence, scheduling systems, parallel processing, and speech and image processing. Dr. Weil is a member of Tau Beta Pi, Eta Kappa Nu, the IEEE, and the IEEE Computer Society.

ANDREW WHINSTON holds the John P. Harbin Centennial Chair in Business at the University of Texas at Austin with appointments in the School of Business and the Computer Science Department. His primary teaching interest is management information systems; current research interests are in optimization approaches to algorithm construction, development tools for the construction of direct manipulation user interfaces, Petri net representation of logic and expert systems, decision support systems theory, distributed artificial intelligence, development of scheduling algorithms in flexible manufacturing systems, and organizational modeling.

He has authored or coauthored over 200 papers and 14 books in a wide range of areas. Two of the books he coauthored (with C. Holsapple and R. Ḅonczek) are *Foundations of Decision Support Systems* (Academic Press, 1981), and *Micro Database Management: Practical Techniques for Application Development* (Academic Press, 1985). He has been a consultant to various companies, governmental agencies, and international organizations on data processing questions.