MIMD IMAGE ANALYSIS WITH LOCAL AGENTS

James J. Clark and Robert P. Hewes

Division of Applied Sciences
Harvard University
Cambridge, MA

ABSTRACT

Complex systems that are typically thought of as exhibiting "life-like" qualities are observed to have many aspects in common with MIMD parallel computing systems. We present a computational model of a class of MIMD systems based on this observation. We describe the application of the model to the parallelization of image analysis tasks. We describe an example of our approach applied to the problem of extracting the circuit design from the layout image of a CMOS integrated circuit.

1. INTRODUCTION

In the quest for increased rates of computation computer scientists and engineers have turned toward parallel computing systems. Such systems speed computational tasks by dividing the task into parts, and executing each part in parallel with separate computational engines. Early parallel computers were of the Single-Instruction-Multiple-Data (SIMD) form [2], wherein each processing element executes the same computations on different data. The effectiveness of these types of parallel computers are limited by the extent to which an arbitrary task can be divided into a set of equivalent subtasks. The large majority of computational tasks, however, are very difficult to partition into sets of equivalent sub-tasks. Implementing these tasks in SIMD computers will result in a very inefficient use of computational resources. A more general, and powerful, class of parallel computers are the Multiple-Instruction-Multiple-Data computers [2]. These computers consist of a set of processing elements that can execute different computations on different data. MIMD computers, however, have the drawback that they can be very difficult to program efficiently, due to the need for the processors to act independently yet cooperate on the execution of a task.

This research was supported in part by the Brown-Harvard-MIT Center for Intelligent Control Systems, under A.R.O. grant number DAA103-86-K-0170.

One way to simplify the task of MIMD programming is to constrain the space of possible programs by providing a template which the programmer can use to specify a program. This template is equivalent to a model of a class of MIMD systems, and systems that are well described by this model will be straightforward to program using the template.

The question which is thus left is what should be the form of the model? We approach this question by observing that many natural systems have the characteristics of MIMD systems. Natural systems are "programmed" by evolution, but programmers of a computational system that is based on this natural model can adapt algorithms developed in nature for use in their programs.

Thus, our goal is to develop a computational model that will allow natural systems to be analyzed and simulated, and that will ease the task of programming a MIMD computer through the applications of constraints derived from the observation of natural MIMD systems.

Significant characteristics of such natural complex systems include:

- *Encapsulation*: Parts of the system can be combined, or encapsulated, into distinct units (which we will call agents), hiding the complexity of the unit's constituents.
- *Diversity*: The atomic units have differing characteristics.
- *Embeddedness*: The agents are embedded in an *environment* with which they can interact.
- *Locality*: Communication between agents and between agents and their environment are local. There is no global controller telling the agents what to do (apart from the data space).
- *Sparseness*: The agents, taken together, do not cover the environment with their loci of communication.

0-8186-6950-0/94 $4.00

- *Mobility*: The agents are able to move about the environment.
- *Mutability*: The behavioral description of the agents can change in response to changes in their internal state or to environmental stimuli.
- *Reproduction*: Agents can be created and destroyed.

Similar systems have been developed by biologists seeking to simulate and model biological systems [5, 6]. However, complex systems with the above characteristics are by no means restricted to biological systems. A collection of mobile robots interacting on a factory floor is a non-biological example of such a system. Thus, in addition to modeling biological systems, we want to use our model to help design and simulate artificial systems, such as groups of autonomous robots, nanotech assemblers, virtual reality systems, and computational applications such as VLSI design and image analysis.

Many image analysis tasks, such as image filtering and pointwise feature detection, are efficiently carried out by SIMD parallel computers. Other tasks, such as edge tracing and linking, are inefficient when mapped to SIMD machines. These tasks can be parallelized more effectively with MIMD computers, and with parallel machines that have more flexible interconnectivity than the standard SIMD machine. One of the most significant costs of moving from a SIMD parallel machine to a MIMD machine is the increase in the difficulty in specifying programs for the machine. In the MIMD setup, the processors must cooperate effectively in performing computations and communicating information. The architecture implied by the MASE model promise to ease the burden of programming parallel versions of such tasks.

2. THE MASE COMPUTATIONAL MODEL

We present in this section a computational model that specifies the form of the MIMD systems that we will allow. The constraints on the allowable MIMD systems are both explicit and implicit in our model. The explicit constraints on the MIMD systems manifest in the form of a program template. This template dictates, in a quite restrictive manner, the nature of the allowable programs, while retaining enough freedom to permit a wide range of programs to be implemented.

The two major elements of our computational model are the *agents* and the *environment*. A formal definition of these elements can be found in [3]. We will provide only an informal definition here.

The environment is a set and a mapping that associates elements of some object space to each element of the set. This mapping can be changed by the agents. An example of an environment is a finite two-dimensional lattice, where at each point of the lattice is associated an integer between 0 and 255 (i.e. a digitized image).

The environment acts as a shared memory for the agents, thereby allowing the agents to interact. The environment also serves as an input/output channel, allowing data (such as images) to be communicated to/from systems external to the MASE system.

An agent consists of a processor, a local memory unit, a collection of five routines, which are continually executed in sequence. Each agent has access to the environment, which acts as a shared memory for the system. The agents have localized access to the environment. This local area of the environment is called the *receptive field* of the agent. The five agent subprograms are:

- ϕ_r : READ. This function reads in data objects located in the receptive field into the agent's local memory.
- ϕ_u : UPDATE. This function maps the current agent state (local menory values) into a new state.
- ϕ_w : WRITE. This function writes out some of the agent's local variables to locations in the agent's receptive field.
- ϕ_a : ALTER. This function determines a new set of the five sub-programs for the agent.
- ϕ_m : MOVE. This function changes the receptive field of the agent.

Programming, or specification of, a MIMD system based on this model involves specifying, for each agent, the five functions as well as the initial state.

3. UTILITY BALANCING

The agent programs must be designed to exhibit the behaviour that will result in the desired task being efficiently executed. In any multi-processor system the most important measure of performance is the speedup in completing a given task obtained by using a number of processors over using a single processor. The two most important measures of a processor's utility are *redundancy* and *irrelevance*. A MIMD system, to be efficient, must apportion out tasks to its constituent processors in such a way as to minimize redundancy and irrelevance. This process is called *load balancing* and is one of the most important facets of MIMD system design [1]. Our MIMD model, based as it is on natural systems, can use load balancing strategies that have been developed by natural systems. One of the

most important of these strategies, aimed at reducing redundancy, is *dominance*, wherein the dominant agent gets to perform a certain activity while other agents are inhibited from doing so. The dominance can be based on fixed agent characteristics, such as an identification code, or can be based on variable agent attributes, such as age, time spent performing a task, or value of an internal state variable (e.g. health).

4. USAGE OF THE MASE MODEL

The MASE model is a model of a certain class of MIMD systems. It is not a good model for existing MIMD parallel computers. It does, however, imply a type of MIMD computer which, if constructed, would directly implement programs specified by the model. Until such a machine is built, we must content ourselves with the use of our model as a simulation and algorithm development tool. Note, however, that there are many existing, non-computer, systems where our model can be used directly, such as assemblies of mobile robots.

We have implemented in software a C++ based object oriented simulator that allows the user to execute a specification of a MIMD system adhering to a restricted version of our computational model. This simulator is an experimental testbed for demonstrating and analyzing the performance of solutions to computational problems that utilize our framework. In order to demonstrate the ease and efficiency with which a complex image analysis task can be implemented using our model, we applied the model to the task of extracting a circuit netlist from a fabrication mask level description of a VLSI layout. This requires a complex image analysis process. The VLSI layout extraction process produces a circuit netlist, which is a listing of circuit elements and their connectivity, from a description of the masks used in the fabrication of the integrated circuit graphical description format. As input to our extractor program we take a rasterized version of the CIF representation [4] of the layout. The output of the program is a series of statements indicating the parameters of each transistor and their connectivity.

The image shown in figure 1 depicts a layout of a simple four-transistor digital NAND circuit. The circuit contains eighteen contacts. Shown are the receptive fields of the agents of different types.

4.1. MASE VLSI Extraction Algorithm

There are many MASE algorithms that could conceivably be written to solve the VLSI layout extraction problem. We will present one such algorithm here, but make no claims as to its optimality. We will, however,

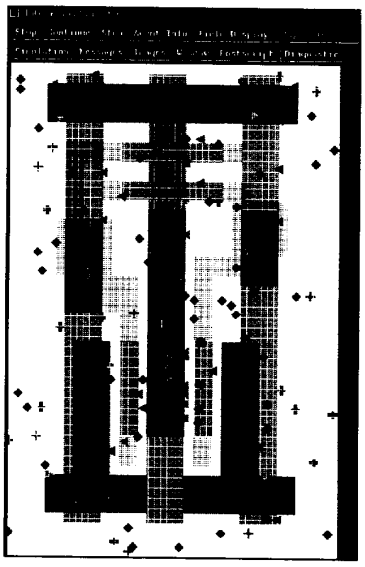

Figure 1: The Simulator view of the layout for a CMOS NAND circuit and the agent receptive fields.

apply whenever possible, utility balancing techniques based on the principles described earlier.

The layout extraction problem can be broken down into a number of subtasks, enumerated as follows, 1) Find wire boundaries. 2) Trace and label wire boundaries such that each wire has only one label and each label is attached to at most one wire. 3) Locate the transistors. 4) For each transistor, trace the transistor boundary to measure the gate length and width and to compile the labels for the gate, source, and drain terminals. Output a transistor description statement. 5) Find contact areas. For each contact, determine the labels of the wires that overlap the contact area, and output a node equivalence statement.

To carry out these tasks, our algorithm employs the following seven types of agents, layer_finder, node_labeller, fet_labeller, fet_output, contact_finder, node_director, and node_propagator.

layer_finder: The task of agents of the layer_finder type is to search the environment for the boundaries of unlabelled wires of metal, poly, or diff layers. The search pattern used is a raster scan. Our algorithm labels wires only on their boundaries. If no layer boundary is found at the current position of the agent, the agent continues to the next position in its scan. If an unlabelled boundary has been detected the agent alters it's type to that of a node_labeller agent. To reduce the number of layer_finder agents when most of the wires have been found a layer_finder agent is caused to turn into a node_propagator if a node_propagator agent is

in the agent's receptive field at that same time a label written by a node_director agent is also in the agent's receptive field.

node_labeller: The job of the node_labeller is to trace the boundary of the wire, while writing a unique label to the environment along the wire's boundary. The uniqueness of the label can be assured if we use the agent ID number as the label and if we ensure that a node_labeller that completes the labelling of a wire, as well as any of its descendents (i.e. agents with the same ID number), cannot change into a layer_finder. The tracing time can be cut by up to 50% if two agents cooperate in the labelling operation. As the agents have only local access to the environment, they will not know that there are other agents that are attempting to label the same wire. Eventually, however, an agent, in the course of it's tracing, will encounter a portion of the wire that has been labelled by another agent. At this point we invoke a *dominance* strategy. A seniority based dominance strategy is used. With this strategy, as an agent writes the node label it also writes the time at which it started labelling the wire. If the agent encountered a label written by another agent it would check the start time written by the other agent. If the other agent's start time was greater than the current agent's start time the other agent's label would be overwritten and the agent would continue tracing and labelling the wire boundary. If, however, the other agent's start time was less than the current agent's start time, the current agent would reverse direction and start labelling with the other agent's label and start time.

When an agent encounters a label that is equal to its current label it does one of two things, depending on whether or not the label is equal to its ID or is that of another agent. If the label is that of another agent, then the agent changes into a layer_finder. If the label is that of the current agent, then the entire perimeter of the wire has been labelled with that label. The agent then turns into a *node_director* agent, unless the wire being labelled is a DIFF wire, in which case the agent turns into a *fet_labeller* agent.

fet_labeller: The job of a fet_labeller agent is to find all occurences of the boundary of transistors along a DIFF wire, and to label these transistor boundaries as such. The label is used to indicate that the node label for the DIFF wire is stable and can be used by the fet_output agents in composing the transistor output statements.

fet_output: The role of the fet_output agents is to gain dominance of a transistor boundary, measure the length and width of the gate region of the transistor, and wait for the labels written to the DIFF and POLY wires on the boundary to become stable. Once all the labels are stable, the labels are written out to a file, along with the transistor gate length and width, and the agent changes to a node_propagator.

contact_finder: A contact_finder agent is created when a node_propagator detects the presence of a contact region which has not already been captured by another contact_finder. The contact_finder's role is to capture the contact region, thereby establishing dominance, and to wait until the wires overlapping the contact area become labelled (through propagation of the labels by the node_propagators). Once the wires have been labelled, the contact_finder agent outputs the labels of the two wires to a file in the form of an equivalence statement. If the contact_finder is in a contact region that has already been captured by another contact_finder, or when it has output a node equivalence statement, it turns into a node_propagator agent.

node_director: The purpose of the node_director agents is to retrace the boundary of a traced wire. This retracement is neccessary to signal the node_propagator agents to fill in the wire interior areas with the wire label. This filling in of the wire interior cannot begin until the wire boundary has been completely traced, as it is not until that time that a label on any point on the wire boundary is guaranteed to be the unique node label for that wire. There is only one node_director on every wire, as they are created from the dominant node_labeller agents on each wire. When the node_director agent completes it's retrace it turns into a node_propagator agent.

node_propagator: The purpose of the node_propagator agent is to propagate the label written on the boundary of a wire (by the node_director agents) into the interior of the wire so that the labels can be used by the contact_finders. The node_propagator agents also look for transistor regions that have been marked by fet_labeller agents but not yet been examined by fet_output agents. If such a transistor is found, the node_propagator turns into a fet_output agent and begins examining the transistor. Also, if an unlabelled contact area is found, the agent turns into a contact_finder agent.

4.2. Agent Interaction

Agents interact in two primary ways. The first way is the modification of an agent's processing via messages written to the environment by other agents. This could, for example, be an activity such as moving in a certain direction in response to a particular message. The second form of interaction occurs when an agent of one type turns into an agent of another type, based on internal state variable values or, again, on the values of external environmental messages. The agent interactions in the layout extractor algorithm are summarized

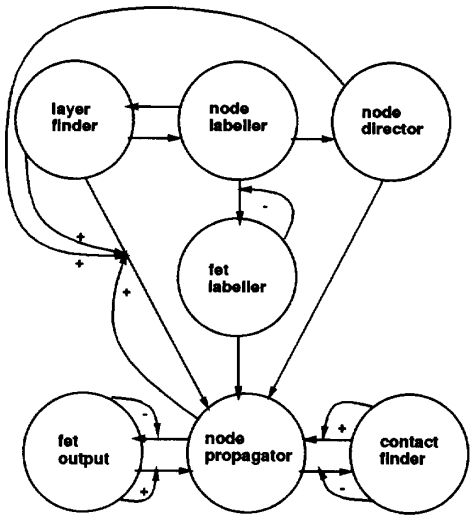

Figure 2: Interaction between agents of different types. Straight lines indicate transitions between agent types. Curved lines indicate the facilitatory (indicated by a '+') or inhibitory (indicated by a '-') effect of different agents on the agent transitions.

in figure 2. The straight lines indicate possible changes of one type into another. The curved line indicate the effects agents of various types have on the probability of a given transitions.

In figure 3 is shown the agent populations over time for the case of a D-flip-flop containing 32 transistors and 120 contacts. In this plot one can see the dynamics of the utility balancing aspect of the algorithm.

5. SUMMARY

We have presented a model for MIMD systems, based on observations of natural systems. This model allows the straightforward specification, via a template, of MIMD parallel programs. An example of the application of this model to an image analysis task was described, that of extracting a circuit from the bit-plane images of the fabrication mask layout of an integrated circuit. This example illustrated the use of load-balancing techniques to allow processors in a MIMD system to cooperate in the detection, labelling, and linking of features in an image.

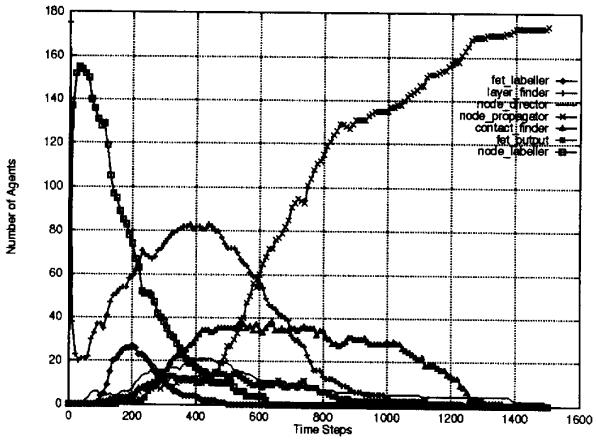

Figure 3: The number of each type of agent as a function of time for the D-flip-flop layout.

6. REFERENCES

[1] I. Ahmad and A. Ghafoor, "Semi-distributed load balancing for massively parallel multicomputer systems," *IEEE Transactions on Software Engineering*, Vol. 17, No. 10, October 1991, pp 987-1004

[2] M. J. Flynn, "Very high speed computing systems," *Proceedings of the IEEE*, Vol. 54, No. 12, 1966

[3] R. P. Hewes, *A Mobile-Agent and Shared-Environment Model of Parallel Computation*, Ph.D. thesis, Division of Applied Sciences, Harvard University, 1994

[4] C. Mead and L. Conway, *Introduction to VLSI Systems*, Addison-Wesley, 1980

[5] H. B. Sieburg, "The cellular device machine: Point of departure for large-scale simulations of complex biological systems," *Computers and Mathematical Applications*, Vol. 20, No. 4-6, 1990, pp 247-267

[6] C. E. Taylor, D. R. Jefferson, S. R. Turner, and S. R. Goldman, "RAM: Artificial life for the exploration of complex biological systems," in *Artificial Life*, C.E. Langton. ed., Volume VI, Santa Fe Institute Studies in the Sciences of Complexity, pp 275-295, Addison-Wesley, 1988