Proceedings of the 1986 Winter Simulation Conference
J. Wilson, J. Henriksen, S. Roberts (eds.)

"AXE": A SIMULATION ENVIRONMENT FOR ACTOR-LIKE COMPUTATIONS ON ENSEMBLE ARCHITECTURES

Jerry C. Yan
ERL 452, Computer System Laboratory,
Stanford University,
Stanford CA 94305.

Stephen F. Lundstrom
Microelectronics and Computer Technology Corporation,
3500 West Balcones Center Drive,
Austin, TX 78759.

ABSTRACT

This paper introduces a set of experimental tools which have been designed to study dynamic run-time distribution of work on mesh-connected concurrent processors. The computation is modeled (and simulated) at the *"operating system level"*. This environment is characterized by its fast turn-around time for model specification, simulation, as well as data collection.

1. INTRODUCTION

Research is being conducted to determine partition strategies that map distributed computations onto a specific class of multiprocessors such that execution time is minimized. One of the most difficult issues to be addressed is choosing an appropriate level of abstraction for studying the dynamic characteristics of the program and the resource management system - without resorting to lengthy instruction-set level simulations or general stochastic models. The "Axe"[1] simulation environment was designed to facilitate such investigations at the operating system level using discrete-time simulation.

The concurrent computations being investigated falls into a subset of the Actor programming paradigm [Agha 85] known as *serializers* [Atkinson 77]. In the Actor paradigm, computations are represented by collections of autonomous *actors* (or objects). Actors interact with one another only via message passing. When an actor receives a message, it may

- perform user-programmed computations;
- send messages to other actors; or
- create new actors of various kinds.

Parallel programs written as communicating sequential processes (CSP) [Brookes 83], remote procedure calls (RPC) [Nelson 81], and a number of other parallel programming paradigms can be easily ported to the actor programming paradigm. A number of artificial intelligence applications can also be expressed naturally in this manner.

The particular class of multiprocessors considered here is termed *ensemble architectures* [Lutz 84, Seitz 82] because it consists of a collection of homogeneous processing elements (or *sites*). Each site is connected to its nearest neighbors in a regular fashion (e.g. the Cosmic Cube [Su 85]). A site is *autonomous* - it contains its own storage, processor and a distributed operating system kernel governing local activities such as message forwarding, task scheduling, and memory management.

The next section explains how "Axe" is used in the on-going research program. Section 3 gives an overview of the system. Specification of software and hardware models are introduced in sections 4 and 5 respectively. The last section illustrates some of the initial results "Axe" produced.

2. RESEARCH CONTEXT

The problem being studied - known as the *partition problem* - has to deal with finding a mapping of actors to sites which results with the minimal execution time. It has been shown that even for programs with regular structures, the partition problem has been shown to remain NP-complete [Mayr 81]. The *partition strategy* - expressed as a set of heuristics - attempts to find an approximate solution to the partition problem for a given program on a given machine. A partition strategy consists from two parts - *placement* and *migration* heuristics. Placement heuristics suggest a site for

[1]"Axe" is not an acronym!

placing an actor when it is created whereas migration heuristics suggest *when* and *where* to move an actor to another location after it has been assigned a site for a certain time. A *dynamic* strategy, as opposed to a *static* one, incorporates run-time observations in its decision for placement or migration of actors, thus giving it the ability to adapt to changes in program behavior in response to changes in input data characteristics.

Three critical sub-problems need to be solved in order to develop partition heuristics.

1. *Program behavior abstraction and analysis* is necessary in order to
   - assign sites to actors declared at compile time,
   - plan where (and what) information is to be communicated between sites during run-time, and
   - aid making *"guesses"* using inherent program structure.
2. *Run-time information management* - which involves both gathering and communicating run-time observations - is necessary so that a strategy can be responsive to program behavior changes. Monitoring program execution introduces overhead in processing, storage as well as communicating. In order to minimize overhead, a minimal amount of information should be collected and transmitted only to sites which needs it.
3. *Fast turn-around facilitates for validation of partition heuristics* is essential both because of the size of the search space as well as the difficulty of the problem.

The "Axe" simulation environment was designed with these specific problems in mind:

- "Axe" *learns* about program behavior by simulation. Instead of relying solely on analysis of program text, "Axe" directly executes the program model, and collects run-time statistics which are used to characterize (i) individual actors and (and actor types) and (ii) the relationships between actors (and actor types).
- "Axe" *automatically* generates a plan for information gathering. The researcher may concentrate on building the software model.
- "Axe" is *fast* when compared with instruction-level simulation. At the same time, the model preserves the key characteristics of program behavior. These characteristics are important to evaluating dynamic partitioning strategies.

3. "AXE"

The "Axe" experimentation environment consists of a set of software tools that allows the research to perform all the following tasks in an integrated environment:

- *Computation model specification* - By analyzing program text, the researcher expresses the program using a *behavior description language*. This model - not the program text - is simulated.
- *Execution environment specification* - This includes various parameters that describes the multi-processing hardware, connection network and various operating system algorithms.
- *Simulation* - Data is collected automatically for performance evaluation.
- *Experimentation* - The researcher is able to study various issues in parallel processing: problem formulation, hardware architectural issues, matching machines and programs, and operating system level algorithms.

The current version of "Axe" includes:

1. a compiler/translator that converts application program models into forms understood by other modules of the simulation environment;
2. a simulator that projects the execution time of the program model on a multiprocessor with specified machine parameters and placement strategies; and
3. a monitor system that gathers run-time statistics to enable evaluation of partition strategies, as well as software and hardware architecture.
4. an experimentation executive that allows the user to inspect (and over-ride) decisions recommended by different heuristics sets.

The compiler is based on standard unix utilities *lex* and *yacc*. The syntax and semantics of the *behavioral description language* that drives the compiler is documented elsewhere [Yan 86]. The compiler automatically generates calls to statistical modules completely transparent to the user. The simulator is written on top of CSIM, a process-oriented simulation system developed by MCC [Schwetman 86]. CSIM's ability to emulate concurrent processes is used to implement multiple copies of operating system kernels as well as concurrent actor processes. In the current version, the experimenter is allowed to interact with the simulator in between simulations for various activities such as:

- invoke different migration/placement heuristic sets;
- manually make migration/placement decisions;
- change machine characteristics and operating system algorithms; and
- change input data to the program (model).

Four design disciplines were observed in the architecting the "Axe" system:

1. *simple/structured user-interface* - enabling the experimenter to model different parallel programs and define various machines precisely and efficiently;
2. *quick turn-around time for experiments* - keeping the researcher interested and productive;

Code	Model
(DefActor (File) (content owner))	(DefActor (File) (owner)) ;one fewer state variable!
. . .	. . .
(DefMethod (File :New_Content) (sender new_data) (if (≠ sender owner) (reply NIL) (progn (setq content new_data) (reply T))))	(DefMethod (File :New_Content) (sender) ;one fewer argument! (if (≠ sender owner) (reply NIL) (progn (use_cpu 4) (use_disk 5) (reply T))))

Figure 1. Code Segment vs. Program Model

3. *automatic gathering of simulation data* - allowing the researcher to concentrate on his/her research, leaving the task of instrumentation to "Axe"; and
4. *minimized need for re-compilation* - "Axe" is structured so that a maximum number of parameters may be modified without needing re-compilation. These include machine characteristics, operating system parameters, input data, simulation data collection methods as well as partition strategy selection.

4. APPLICATION SPECIFICATION

Realistic evaluation of new computer organizations and the accompanying resource management tools depends on the study of real applications. However, the development of complete language and compiler tools, together with run-time environments is currently prohibitive for short term use in research. Here, the computation is simulated based on a program model. This model preserves the message pattern between actors as well as the relative processing and storage requirements of each actor. The basic idea in the program model may be stated as follows: *replace (as many as possible) time-consuming statements in the real program by statements that simply advance simulation time or statements that describe resource utilization.* Figure 1 illustrates the difference between a piece of *real* code and its corresponding model using a "file" actor as an example.

A particular behavior of a ***file*** that has to do with new data is coded as the handler for the ***:New_Content*** message. In the original code, the sender of the message is verified before ***new_data*** replaces the ***content*** of the file. In the model, the verification process is preserved whereas the latter part is replaced with two statements describing the use of the CPU and storage device(s). This substitution process can be applied to very complex computations. A more detailed discussion of how to abstract program behavior, how to build program models and the language used for specifying these models is given elsewhere [Yan 86].

5. EXECUTION ENVIRONMENT SPECIFICATION

In order to study dynamic partition strategies in a concurrent systems environment, one must specify the execution environment to be studied in addition to modeling the application. The execution environment includes the underlying concurrent hardware, the topology of the connection network, the routing policies enforced, and the task scheduling algorithms being studied.

"Axe" models concurrent hardware as a collection of predefined abstract machines (known as *sites*) connected in some topology. A site represents various operating system functions available for management and execution of the parallel models of computation. This machine model may be tailored further parametrically by the user:

- values of hardware parameters which can be specified (or modified during run-time!) include:
  - *message sending/receiving overhead,*
  - *overhead involved in process creation/ blockage,*
  - *relative speeds of communication links to the processors,*
  - *total number of sites,*
  - *number of processors per site, and*
  - *memory size at each site*

- a number of *built-in* topologies and routing algorithms can be selected by modifying a single variable or by writing one simple function in C;
- a number of *built-in* partition and scheduling algorithms are also offered.

All the graphs shown below were obtained from "Axe" simulation results.

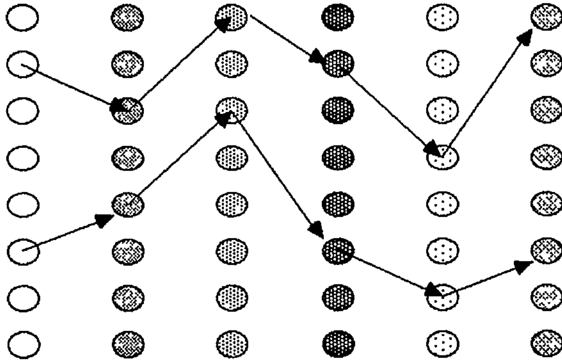


Figure 2. Pipeline-like Computation

6. EXPERIMENTAL USAGE

The experimentation process begins with the building of a program model and definition of the execution environment. If the researcher wishes, he/she may choose to define his own versions of routing algorithms and partition strategies. Then "Axe" takes over. These specifications are compiled. All the necessary tools for monitoring the simulation are automatically generated. The *execution model* is now generated. The user may run a number of separate experiments from one *execution model.* Simulations can be run with different input data, machine characteristics and predefined strategies/algorithms without having to recompile the system.

Three different studies are illustrated here, indicating the kind of research that "Axe" is currently supporting:

1. Variation of speed-up vs. communication costs
2. Variation of speed-up vs. number of sites
3. Comparison of partition heuristics

In all cases, the computation being modeled is *pipeline-like* (See Figure 2). There are 48 actors arranged in 6 stages. Messages are sent to actors in the first stage. Each actor computes for a certain time, then picked another one on the next stage as the target of a message containing the result of the computation. Target selection is dependent on the content of the message received. This process ripples down the pipeline. Each actor carries out different computation and receives different data than the others. Thus the actors, have different completion times. The machine being simulated is connected via a two dimensional, square, nearest neighbor grid. The routing strategy chooses the shortest route between two sites. When this route is blocked, the message is queued until a "hop" can be made to the next neighbor *en route.*

6.1 Variation of Speed-up vs. Communication Costs

The time to route a packet to a neighbor is increased gradually. As expected, Figure 3 shows that the *"maximum achievable speed up"* gradually decreases as the communication cost increases. These *"speed up"* values were obtained from the best mappings found by a set of partition heuristics when the same application model is mounted on 4 to 25 sites.

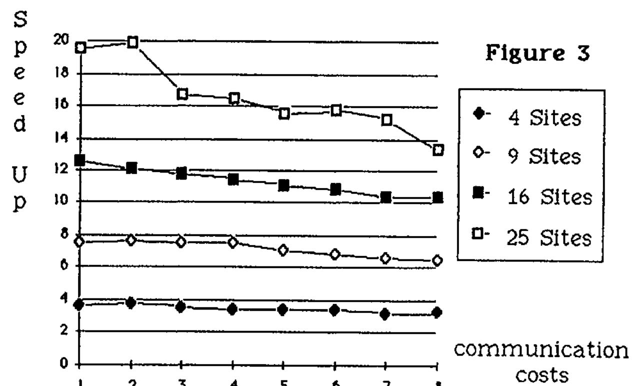


Figure 3

6.2 Comparison of Partition Heuristics

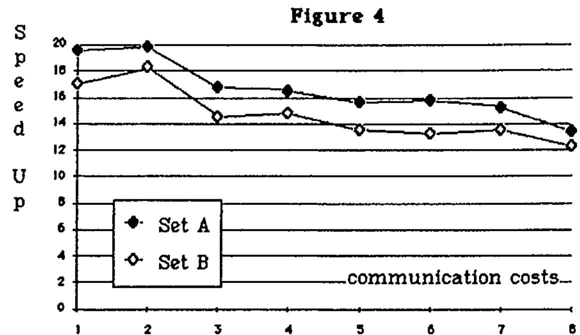
Figure 4 compares the *"maximum achievable speed up"* found by two different sets of partition heuristics. Set *A* shows an advantage over set *B* over a range of commùnication costs on 25 sites[2].

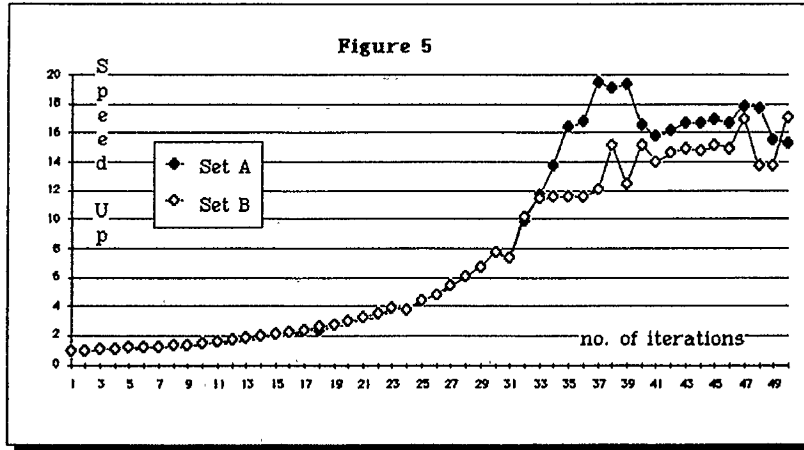
[2]Set *B* is similar to a simple greedy algorithm - a busy actor is moved from the busiest site to the free-est site. Set *A* consists of a more complicated heuristic set that estimates the relative merits of moving actors to various target sites taking communication costs into account. Since, it is not the purpose of this paper to discuss partition heuristics, the exact nature of these heuristics will not be detailed.

One way to develop partition heuristics is to use an *iterative "post-game"* approach:

1. begin with an initial mapping of actors to sites
2. simulate once and collects data
3. find a *"better"* mapping by applying migration heuristics to move one actor to another site
4. go to step 2

The above steps are repeated a finite number of times. Figure 5 illustrates the iteration process. All actors were placed in one sites initially. It can be seen that Set *A* attains the *"maximum achievable speed up"* in a smaller number of iterations[3].





7. SUMMARY

The study of dynamic resource management requires the use of significant test cases. The development of a detailed model of a complete application seems unreasonable. In the case reported here, an abstraction of the application program is created using actor-like models. This actor-based application abstraction is "executed" on an operating-system level model of the multiprocessor system. The "Axe" simulation environment allows specification of the applications, specification of the multiprocessor organization to be studied, and instrumentation of the computational experiment. This system is now operational and is being used in our research to develop run-time partition strategies for highly-concurrent systems.

[3]No attempts were made to stop the application of the heuristics if a better mapping is not found. This explains why the curves were not monitonically increasing.

ACKNOWLEDGMENTS

This research was supported by NASA under contracts NAG 2-248 and NCA2-109, using facilities provided under contract NAGW 419. We would also like to acknowledge MCC for permitting the use of CSIM, granting access to its computing facilities, and supporting S. Lundstrom's involvement with the program.

REFERENCES

[Agha 85] *Actors: A Model of Concurrent Computation in Distributed Systems*, Gul A. Agha, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Memo 844.

[Anderson 77] *Specification and Proof Techniques for Serializers*, R Atkinson and C Hewitt, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Memo 438.

[Brookes 83] *A Theory of Communicating Processes,* S D Brookes, C A R Hoare, A W Roscoe, Carnegie-Mellon University Department of Computer Science, CMU-CS-83-153.

[Lutz 84] *Design of the Mosaic Processor,* Christopher Lutz, California Institute of Technology, Technical Report 5203:TR:85.

[Mayr 81] *Well Structured Parallel Programs are not Easier to Schedule,* Ernst Mayr, Stanford University, Computer System Lab Technical Report STAN-CS-81-880.

[Nelson 81] *Remote Procedure Call,* B J Nelson, Xerox Corp. Palo Alto Research Center, CSL-81-153.

[Schwetman 86] *CSIM: A C-Based, Process-Oriented Simulation Language,* Herb Schwetman, Microelectronics and Computer Technology Corporation, Proceedings for Winter Simulation Conference '86, Washington DC.

[Seitz] *Ensemble Architectures for VLSI - A Survey and Taxonomy,* Charles L Seitz, 1982 Conference on Advanced Research in VLSI, Massachusetts Institute of Technology.

[Su 85] *C Programmers' Guide to the Cosmic Cube,* W-K Su *et al,* California Institute of Technology, M.S. Thesis (CITCS) 5129:TR:84.

[Yan 86] *Parallel Program Behavior Abstraction,* Jerry C Yan, Computer System Laboratory, Stanford University, CSL-TR-86-298

AUTHORS' BIOGRAPHIES

JERRY C. YAN is a Ph.D. student in the Electrical Engineering Department of Stanford University. He received an M.S.E.E. from Stanford University in 1984, and a B.S. (E.Eng.). from Imperial College, London University, England. His research interests include parallel processing, distributed resource management, simulations and object-oriented programming.

Jerry C Yan

ERL 452, Computer System Laboratory,

Stanford University, Stanford CA 94305.

(415)-723-1559

STEPHEN F. LUNDSTROM is Vice President and Program Director of the Parallel Processing research program at Microelectronics and Computer Technology Corporation (MCC). He is also Consulting Associate Professor of Electrical Engineering in the Computer Systems Laboratory, Electrical Engineering Department at Stanford University. His research interests include all aspects of highly concurrent systems and in understanding their application and performance.

Stephen F Lundstrom

Microelectronics and Computer Technology Corporation,

3500 West Balcones Center Drive,

Austin, TX 78759.