A HIERARCHICAL METHODOLOGY FOR MULTILEVEL AI COMPUTER ARCHITECTURES*

Chilgee Lee, Tag Gon Kim**, and Bernard P. Zeigler
AI and Simulation Group
Department of Electrical and Computer Engineering
University of Arizona, Tucson, Arizona 85721

**Department of Electrical and Computer Engineering
University of Kansas, Lawrence, Kansas 66045

ABSTRACT

Computational applications have advanced from numerical data processing to non-numerical data processing such as artificial intelligence (AI) symbolic processing. Advances in VLSI technologies also have led to the development of high performance microprocessors with significantly reduced cost. These advances have led to an increasing interest in computer systems compound of multiple processors to optimize performance versus costs.

The processors may be interconnected in a hierarchical manner to compose a more powerful computer system. Arranging the processing elements to match the natural problem structure is important for optimal performance. However hierarchical multicomputer architecture is too complex to be evaluated analytically or by existing modelling and simulation techniques.

This paper provides an introduction to an object-oriented, discrete event simulation environment for modelling and simulating hierarchical multilevel computer architectures. Starting from a structural knowledge representation scheme, System Entity Structure (SES), we describe how to synthesize a SES into a simulation model using modelbase concepts. Finally to show the power of our methodology and environment, modelling and simulation of a hierarchical multicomputer with multilevel clusters suited for solving the traveling salesman problem is presented.

* This research was partially supported by National Science Foundation (NSF) Grant CCR8714148, "Intelligent Simulation Environment for Advanced Computer Architectures"

INTRODUCTION

Computing is moving from sequential, centralized numerical data processing to parallel, distributed nonnumerical data processing such as symbolic processing and pattern recognition. Many scientists and engineers have addressed the advantages of the parallel, distributed multicomputer systems in terms of their higher performance, improved reliability and lower cost. Due to the advances in VLSI technology, a multicomputer system with even more than one million processors is feasible (Hwang and Degroot 1989).

The major approaches to high performance computer architecture design can be categorized based on their hardware granularity: coarse-grained, medium-grained and fine-grained (Sheng, Lee and Christensen 1990).

Coarse-grained architectures have the edge for well-structured problems such as numerical computation for aerospace applications, but suffer from the problems of the von Neuman architecture which separates the processor and memory. The data communication delay between processor and memory dictates the minimum required processing time. Medium-grained architectures are networks of minicomputers which have the inherent advantages of a network but lacks the power for memory intensive applications.

The fine-grained processors of the third approach have a higher degree of parallelism resulting from the ability to allocate the number of processors based upon load requirements. The demand for massively parallel processing power arises from the development of symbolic processing systems such as expert systems. Arranging the processing elements to match the natural irregular structure of data for symbolic processing has been shown to achieve high performance (Hillis 1985).

A CLUSTER CONCEPT IN MASSIVELY PARALLEL COMPUTERS

Processing elements in a massively parallel computer must be able to communicate with each other. However for a massively parallel computer a true complete connection (each processing element in an N-node network must have N-1 interfaces to other processing elements) is too expensive, and even impossible for large N.

Thus a multicomputer system that combines a number of separate individual groups of processors with point-to-point links is desirable (Uhr 1987). Uhr has emphasized the need to explore clusters, and compounds of clusters for massively parallel computers. These kinds of computer architectures can be synthesized in a rather simple and elegant manner by combining the clusters to form a multi-layer system.

As an example, Cm*, one of the earliest multilevel computers had five clusters of ten PDP-11/32s (Swan, Fuller and Siewiorek 1981). Each computer module (Cm) consists of a DEC LSI-11 processor, a local switch, bus memory and devices. Ten computer modules and one mapping processor, named Kmap, form a cluster. Finally the five clusters are interconnected through the intercluster bus (Figure 1).

A bus, ring, or crossbar are attractive candidates to link a small number of computers together into a local cluster. A large multilevel computer then may be built using any number of the local clusters. Several different types of topologies such as the hypercube connection of the Intel iPSC, have been suggested. Each topology may have several different message control protocols based on the designer's intentions.

OBJECT-ORIENTED MODELLING AND SIMULATION

The design of the multilevel computer architecture is too complex to be evaluated analytically, since task computation, communication requirements, and resource contention cannot be easily characterized. To maximize a system's performance, processor loads must be balanced dynamically during execution. Thus the problems in designing such a multilevel computer architecture must be investigated by means of simulation.

Simulation can offer an efficient means of investigating the enormous number of alternatives for existing or proposed multilevel computer architectures, thereby saving effort, time and cost. A multilevel computer architecture may be viewed as a discrete-event system. Existing discrete-event simulation languages focus on flat operations and collections of statistics based on event subroutine calls (Law 1990; Pegden 1986). These simulation languages are suitable for simulations of material handling systems such as those used in manufacturing. However a model developed using the conventional simulation languages is non-modular, not-reusable, inflexible and no support for hierarchical decomposition of the system. To enable the hierarchical decomposition of system and the development of modular, reusable models, object-oriented concepts are required.

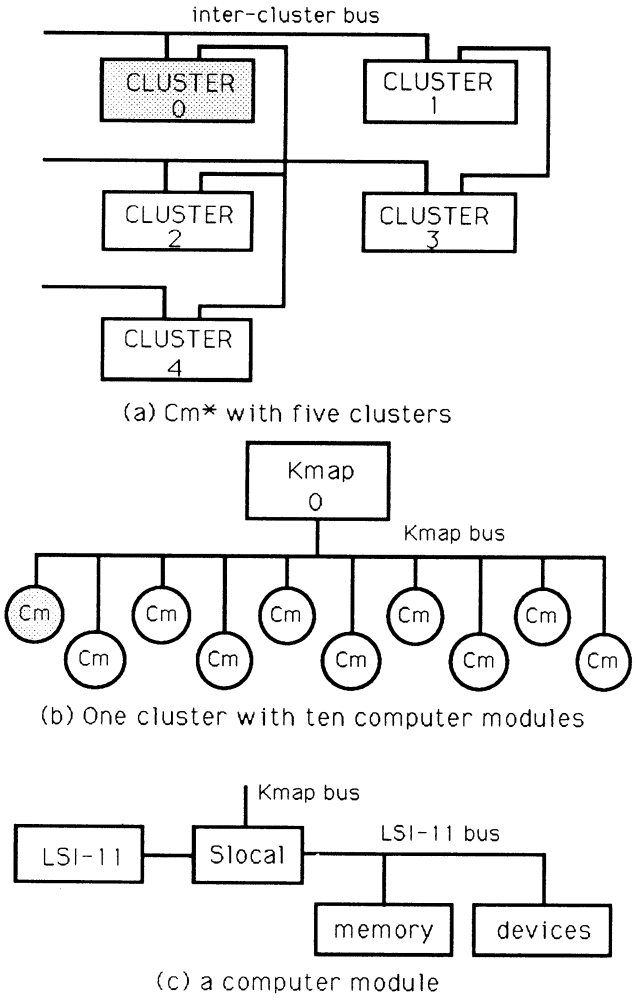

Figure 1. Construction of a Cm* - two level multicomputer

In traditional programming environments, data and procedures are separate entities; the programmer is responsible for applying active procedures to passive data structures. In object-oriented programming systems (OOPS), an object is an entity that combines its state variables and its associated methods (Thomas

1989). An object's method or state variable may only be accessed by sending a message to the object. Similar objects can be organized into a hierarchical family of classes which are definitions of types. Each instance of a class is distinct but shares certain fundamental properties with other instances. Objects in a subclass inherit all features of its parent class.

In a typical non-object-based program, a main routine calls on other routines to work at appropriate times and a large percentage of the machine's time is spent in the overhead of subroutine calls (Hillis 1985). The subroutine call in a conventional program is a command sent to by a master to a specific slave determined at compile time. In contrast, the message-passing paradigm allows a much higher degree of flexibility. In OOPS the actual receiver of a message is not determined until run time. Thus OOPS encourages a much more decentralized style of decision making and concurrent operation. The real system interaction in a multilevel computer architecture can be conveniently represented by message-passing.

The object-oriented programming technique enhances software maintainability, extensibility, and reusability. For the same reasons, in an object-oriented modelling and simulation environment we can reduce designing and testing time significantly. Major concepts in OOPS are as following (Thomas 1989):

Data encapsulation - information hiding
Message passing - not subroutine call
Dynamic binding - run time message interpretation
Inheritance - children inherit features of its parent

Incorporating object-oriented concepts into the modelling and simulation environment also reduces designing and testing time significantly. Similar processing elements and the clusters of those processing elements can be easily built using OOPS.

SYSTEM ENTITY STRUCTURE AND MODEL BASE

For hierarchical modelling and simulation, a framework for hierarchical knowledge representation for both the structure and behavior of a system is required. Zeigler (1984) proposed the Entity Structure Base (ENBASE) which contains a set of System Entity Structures (SESs) for the structural description and a Model Base (MBASE) which contains a set of models in a modular form. Each model has the behavioral description of component, such as how to react to the receiving of an external message (figure 2).

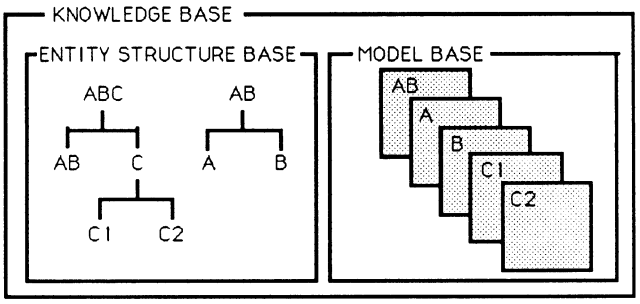

Figure 2. System Entity Structure/Model Base Framework

The SES represents knowledge of decomposition, component taxonomies, and coupling specifications and constraints. An entity, a distinguishable object, may have several aspects, each denoting a decomposition of subcomponents. An entity also may have several specializations, each representing a classification of the possible variants of the entity. A multiple entity represents the set of all members of an entity class. A SES for a Cm* is shown in figure 3. The subsystems of Cm* are identified as entities falling under an aspect called Cm*-dec, which represents the decomposition of the Cm* into a intercluster-bus and clusters. Device-spec represents that the device might be a terminal or a printer. There are three cases of multiple entities (clusters, Cms, and devices).

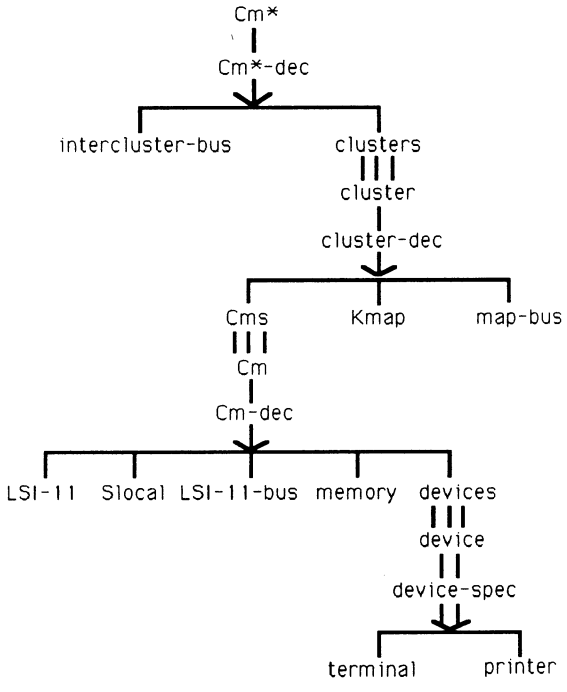

Figure 3. System Entity Structure for a Cm*

The user can reduce the initial SES to a structure called a composition tree using pruning operations based on his/her design goal. The composition tree contains all the information needed to synthesize a simulation model in hierarchical fashion from components in the MBASE. Modular models have a hierarchical structure in which subcomponents are coupled together to form larger ones. To facilitate synthesis, model components must be easily assembled into new, larger combinations for model reusability.

DEVS-SCHEME

DEVS-Scheme (Zeigler 1987; Kim 1988) is an environment for specification of hierarchical, modular discrete event models and simulations in a Lisp-based object-oriented framework (figure 4). In contrast to other knowledge based simulation systems, DEVS-Scheme is based on the DEVS formalism (Zeigler 1976), a theoretically well grounded means of expressing hierarchical, modular discrete event models. The implementation of DEVS-Scheme in PC-Scheme (TI 1986) is that of a "virtual hierarchical multicomputer architecture". DEVS-Scheme does not impose any limit on the depth of hierarchical structure, hence on the degree of compounding the clusters.

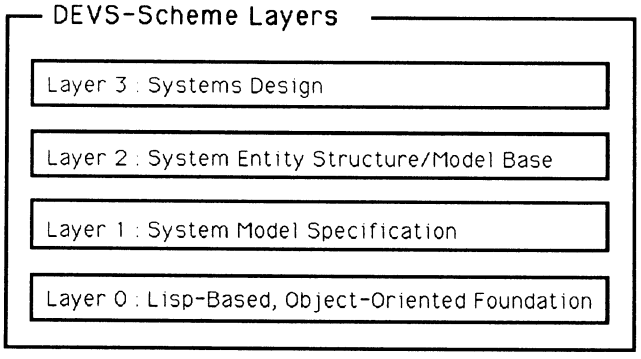

Figure 4. DEVS-Scheme Layers

The interconnection topologies of the processing elements and associated communication protocols determine the performance of hierarchical multicomputers relative to the problem domain. An integrated modelling and simulation environment should support topology selection as well as the number of clusters and their subcomponents.

To support the modelling and simulation of multilevel computer architecture, DEVS-Scheme provides a class, called kernel-models, whose components are of the same class and with uniform coupling. To facilitate class creation a method, called make-class, is provided. The subclasses so created have the same structure as their parent class in that their class variables, instance variables, and methods are the same as those of the parent class. Clusters are associated with different subclasses. Several topological modelling and simulation tools for tree-, ring-, hypercube-, cellular-, broadcast-, and controlled-models are provided in DEVS-Scheme.

A MULTILEVEL COMPUTER APPLICATION

The traveling salesman problem (TSP) is a well known problem of finding the smallest closed loop that connects all (N) cities only once in a given problem domain. The TSP is a NP-hard problem for which no tractable solution exists, and belongs to an important class of scheduling and routing problems. In approaching the TSP, researchers have taken two distinct tasks. One group tries to create an exact algorithm through branch and bound. The other group tries to solve the problem by creating an initial tour by some simple rules (random, nearest neighbor, etc.) and then improving this tour by various methods (Croes 1958).

Recently, due to improvements in the parallel computer technology, researchers have tried to solve the TSP by using parallel machines to achieve more speed up. Felten et al. applied the second approach to solve the TSP using a hypercube MIMD multiprocessor, and reported significant decrease in problem solving time (Felten, Karlin and Otto 1988). Because of the inherent message passing capability of hypercube machine, an initial tour can be easily improved by local path modification, as well as global cost reduction. The large number of processing elements in a hypercube machine may be used to solve the load balancing problem by using a one to one mapping of cities onto processing elements. Though this may solve the load balancing problem it also introduces a significant increase in the communication load.

The approach in this paper, which appears to be novel, is an application of the bounding heuristics technique. This combines the branch-and-bound algorithm with artificial intelligence techniques, to a multilevel cluster. Several agents are interconnected with a manager to build a cluster and then the cluster may be a subcomponent of a higher level cluster. Branch-and-bound constraints may be assigned to each agent to determine the shortest path based upon the given constraints (Aho 1983). Our simulation model for this approach is shown in figure 5.

A global manager determines an initial path length, named as so-far-best, by using the shortest-edge-first

algorithm. The global manager then provides each cluster with this path length and different constraints. The local manager of each cluster then provides more specific constraints as well as a so-far-best value to each agent in the cluster.

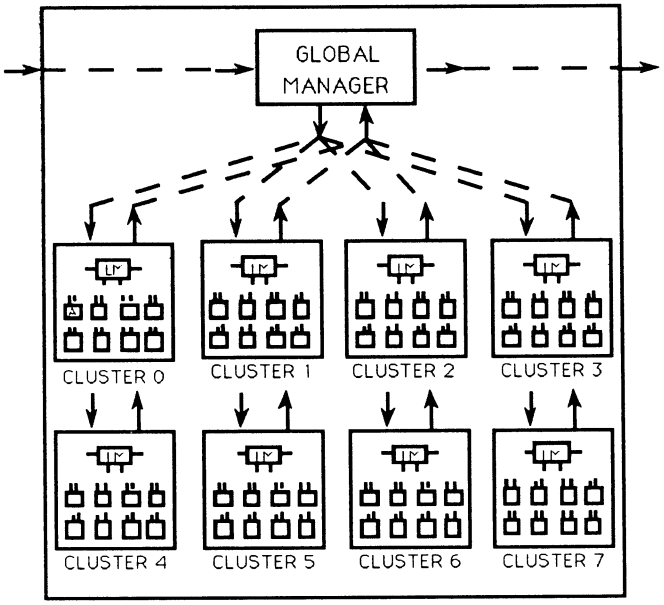

Figure 5. A multilevel computer architecture for TSP

The agents begin processing concurrently to build their own path based upon the given constraints. For each expansion of a path the agent finds some valid cities by removing the restricted cities, which are inferred from the given constraints and previously composed partial path, from (N-1) cities. The agent then calculates low-bounds for each valid city and selects good cities after dropping the cities whose low-bound is equal to or greater than the so-far-best. Cities with smaller low-bounds are considered to be more promising candidates. The partial path is expanded to the city whose low-bound is the smallest (best-first). The rest of good cities are saved into a stack for further investigation. In this way each agent attempts to complete a tour of all N cities.

If an agent finds a shorter path than the current so-far-best, the agent sends a message to its local manager to update the so-far-best. The local manager passes the updated so-far-best to the global manager. The global manager broadcasts the updated so-far-best to all local managers. The local managers then broadcast the so-far-best to all working agents so that the agents can reduce the number of good cities. Agents who fail to find a better path terminate their processing and report their idle status to the local manager so the local manager can assign another trial with different constraints. The final optimal path is determined after all agents complete their processing or have been terminated.

A manager with many agents in a single-level architecture has a high potential to be a bottleneck. The multi-level architecture overcomes the physical limits of a single-level architecture's fan-out and communication bandwidth. We can also take advantage of distributed processing by distributing the global manager load to several local managers.

CONCLUSIONS AND FUTURE RESEARCH

In the TSP application with DEVS-Scheme, the granularity of a multi-level computer, the communication bandwidth, and the control algorithm can be investigated efficiently. We implemented an asynchronous message passing scheme for communication between the manager and agents. The memory size for each agent was estimated based upon the application size, i.e., N cities.

A simulation model with upto sixty-four agents, decomposed into eight clusters has been tested on an Intel 80386-based PC and Explorer II. It is very efficient to use a Lisp machine to run these multilevel computer architecture models implemented with DEVS-Scheme because of its optimization for symbolic processing. The Texas Instruments made Explorer II is designed to support Lisp and through the use of a "Scheme to Common Lisp translator" supports TI-Scheme. Its 16 MB of main memory, 128 MB of virtual memory and several advanced system features can overcome the speed and model size limitations of PC-Scheme in the MS-DOS environment.

Currently there is not a single interconnection topology or communication protocol which adequately supports general applications. This hierarchical, modular modelling and simulation technique using SES and MBASE will allow the evaluation of various multilevel computer architectures, and the results will be a set of benchmarks which can be used to select the appropriate processing element type, interconnection topology and communication protocol for the particular applications such as real-time systems.

We are studying different topologies for multi-level computer architectures for various applications. Good topology candidates include broadcast, hypercube, cellular, tree and ring for modeling and simulating robots in a space station and optical computer architectures (Louri 1988, Chi 1990).

References

Aho, A.V., J.E. Hopcroft and J.D. Ullman. 1983. Data Structures and Algorithms, Addison-Wesley Publishing Co., Reading, MA

Bhuyan, L.N. 1987. "Interconnection Networks for Parallel and Distributed Processing," IEEE Computer, Vol. 20, No. 6, pp. 9-12, June

Chi, Sung-Do and B.P. Zeigler. 1990. "DEVS-based Intelligent Control of Space Adapted Fluid Mixing" To appear: "*Proceedings of the Fifth Conference on Artificial Intelligence for Space Applications* ," NASA/EB44, MSFC, Alabama, May

Concepcion, A.I. and B.P. Zeigler. 1988. "DEVS Formalism: A Framework for Hierarchical Model Development" IEEE Trans. on Software Eng., vol. SE-14, No. 2, Feb

Croes, G.A. 1958 "A Method for Solving Traveling Salesman Problems" Operations Research, Nov-Dec.

Felten, E., S. Karlin and S.W. Otto. 1988. "Applications of Monte Carlo Methods: Traveling Salesman and QCD" Solving problems on concurrent processors, Prentice-Hall, Englewood Cliff, NJ

Hillis, W.D. 1986. The Connection Machine: MIT Press, Cambridge, Mass.

Hwang, K.,and D. Degroot. 1989. Parallel Processing for Supercomputers and Artificial Intelligence, McGraw Hill, NY, NY

Kim, T.G. 1988. "A Knowledge-Based Environment for Hierarchical Modelling and Simulation" Ph.D. Dissertation, Univ. of Arizona, Tucson, AZ

Kim, T.G., C. Lee, E.R. Christensen and B.P. Zeigler. 1989. "System Entity Structuring and Model Base Management" submitted to IEEE Trans. on Systems, Man, and Cybernetics

Law, A.M. and W.D. Kelton. 1990. Simulation Modeling and Analysis, 2nd Ed, To be published

Louri, A. and K. Hwang. 1988. "A Bit-Plane Architecture for Optical Computing with Two-Dimensional Symbolic Substitution," *Proceedings of the 15th Annual International Symposium on Computer Architecture* ," Honolulu, HI, pp.18-27, May

Pegden, C.D. 1986. Introduction to SIMAN: Systems Modeling Corp, State College, Pennsylvania

Reed, D.A. and D.C. Grunwald. 1987. "The Performance of Multicomputer Interconnection Networks," IEEE Computer, Vol. 20, No. 6, pp. 63-73, June

Sheng, O.R., C. Lee and E.R. Christensen. 1990. "Distributed Operating Systems: Taxonomy, Issues and Applications" submitted to IEEE Computer special issue on Operating Systems, May

Swan, R.J., S.H. Fuller and D.P. Siewiorek, 1981. "Cm* - A Modular, Multi-microprocessor" Tutorial on Parallel Processing, pp. 146-153, IEEE Computer Society

Texas Instruments. 1987. PC-Scheme Manual, TI Inc., Austin, Texas

Thomas, D., 1989. "What's in an Object?" Byte, pp. 231-240, March

Uhr, L. 1987. Multi-Computer Architectures for Artificial Intelligence: Toward Fast, Robust, Parallel Systems. New York, NY: John Wiley & Sons

Zeigler, B.P. 1976. Theory of Modelling and Simulation. New York, NY: Wiley

Zeigler, B.P. 1984. Multifacetted Modelling and Simulation. London, UK and Orlando, FL: Academic Press

Zeigler, B.P. 1987. "Hierarchical Modular Discrete-Event Modelling in an Object-Oriented Environment," Simulation, Vol. 49, No. 5, pp.219-230, Nov.

Chilgee Lee

Ph.D. candidate in the Department of Electrical and Computer Engineering, University of Arizona. He received his MSE in Electrical and Computer Engineering from Arizona State University in 1985 and BSEE from Sungkyunkwan University, Korea, in 1980. He worked as a broadcast engineer with Korean Broadcasting Systems 1979 - 1983. His major research interests are discrete event simulation, object-oriented software engineering and performance modelling of distributed computer systems.