MISSE: A MULTI-LEVEL INTELLIGENT SYNTHESIS AND SIMULATION ENVIRONMENT

Pao-Ann Hsiung

Institute of Information Science
Academia Sinica, Taipei, TAIWAN, R.O.C.
Email: eric@iis.sinica.edu.tw

ABSTRACT

Multi-level Intelligent Synthesis and Simulation Environment (MISSE) is an object-oriented, top-down, high-level design environment for multiprocessor systems. Three important aspects of multiprocessor system design: modeling, synthesis, and simulation are supported in MISSE. First, multiprocessor systems are hierarchically classified and system parts modeled as objects with inter-relationships. Second, two system level design methodologies: *Performance Synthesis Methodology* and *Intelligent Concurrent Object-Oriented Synthesis Methodology* are proposed and implemented in MISSE. Third, an object-oriented flexible simulation tool, *Modularized Reconfigurable Simulator*, is proposed to simulate the various synthesized design alternatives. In a typical design process using MISSE, a multiprocessor system is iteratively modeled, classified, synthesized, simulated, and its performance evaluated at each of the four MISSE-levels: *system*, *cluster*, *node*, and *instruction*. The environment is theoretically consolidated by an analytical model called *Multi-token Object-oriented Bi-directional net*, which is a high-level Petri net similar to the popularly used Colored Petri net. Besides the systematic design of a complete multiprocessor system, MISSE can also be used to explore the design of new interconnection architectures, to verify fault-tolerant architecture schemes, to validate performance bounds of multiprocessor task scheduling algorithms, and to test parallel application software. Thus, MISSE is suitable for the rapid prototyping of completely new systems as well as for the evaluation and improvement of existing ones.

1. INTRODUCTION

The design automation of multiprocessor (MP) systems calls for an integrated environment in which the whole design process, starting from system modeling to design simulation and performance evaluation, can be completed within a single common locale such that the unnecessary construction of interfaces and exchange of data between different design phases can be avoided. *Multi-level Intelligent Simulation and Synthesis Environment* (MISSE) is proposed for this purpose. Such an environment will not only speed up the whole design process, but also eliminate human or machine errors during data exchange between any two different design phases, for instance, misrepresentation of a modeled system when it is input to a synthesizer.

When a new MP architecture or system interconnection is proposed, one has to verify its various characteristics, for example, reliability, scalability, and fault-tolerance. Application algorithms or programs have to be rewritten and tested on the newly proposed architecture. MISSE provides such a design and test environment. New MP task scheduling methodologies also need such an environment to validate its correctness and performance bounds. Hence, both newly proposed hardware as well as software can be designed and tested within MISSE.

2. RESEARCH RESULTS

Though MISSE was designed for the system-level synthesis of multiprocessor systems, yet both new hardware architecture and system software applications have been tested and verified using MISSE. The following three subsections give a description on the various research results achieved under this project: the design of two novel fault-tolerant *Extra-Stage Cube* Multistage Interconnection Networks (MIN), the validation of performance bounds of some recently introduced multiprocessor task scheduling algorithms, and the testing of some hypercube programs.

2.1 Fault-Tolerant Alternative Extra-Stage Multistage Interconnection Network

Chen *et al.* proposed two fault-tolerant MINs: *Alternative Extra-Stage Cube* (AESC) and *Combined Extra-Stage Cube* (CESC) [1]. An example of AESC MIN is shown in Fig. 1. The fault-tolerant capabilities of AESC and CESC were theoretically analyzed by Chen. In this article, we model these MINs as objects and synthesize them into real parallel architectures. These architectures are then simulated using the *Modularized Reconfigurable Simulator*, which will be described in Section III.C. On simulation, the single and double fault-tolerance capabilities of AESC and CESC, respectively, are experimentally verified using MISSE.

2.2 Multiprocessor Task Scheduling

Recently, Lin *et al.* proposed several new heuristic algorithms for scheduling multiprocessor tasks with tight

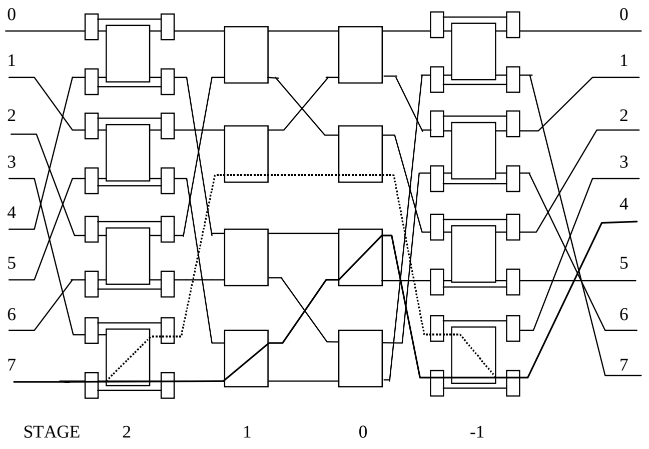


Fig. 1 Alternative Extra Stage Cube MIN

performance bounds [2]-[6]. The performance bounds of these algorithms were analyzed theoretically by them.

The scheduling algorithms modeled and implemented in MISSE are *Modified Largest Dimension First* (MLDF) [3], *Largest Scheduled Parallelism First* (LSPF) [4], *Largest Scheduled Dimension First* (LSDF) [5], and *Largest Width with Largest Processing Time first* (LWLPT) [6]. Different hypercube-based parallel systems have been specified, modeled, and synthesized using MISSE, and the MLDF, LSPF, and LSDF algorithms have been used during the simulation process to schedule MP tasks on these systems. The performance bounds experimentally obtained reflect the correctness of the theoretical results. Since the LWLPT algorithm schedules tasks on an abstract MP system without considering any interconnection networks, the implementation of LWLPT in MISSE is used to show how different interconnection networks, for instance Shared Bus, MIN, and Hypercube, affect the performance bounds of the scheduling algorithm.

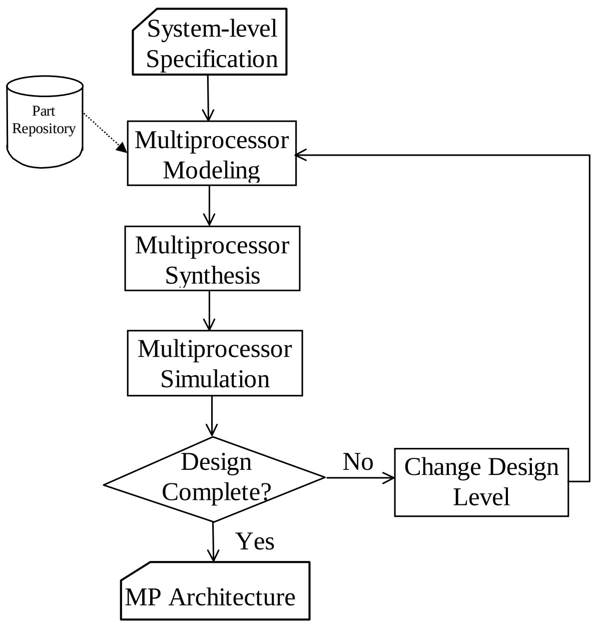


Fig. 2 MISSE Design Flow

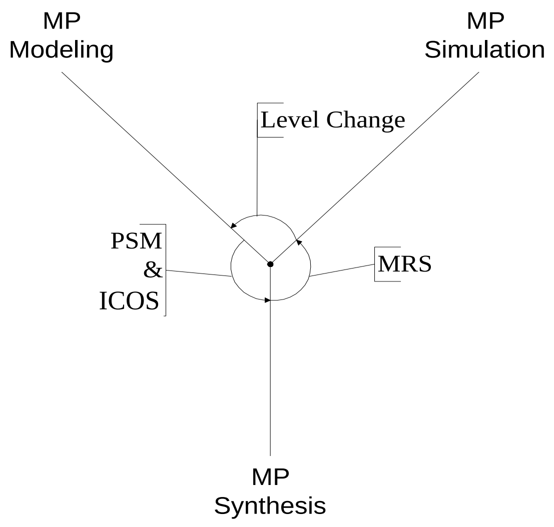


Fig. 3 MISSE Y-Chart

2.3 Hypercube Programs

Based on the *Modularized Reconfigurable Simulator*, a hypercube-based system simulator [7] was developed. Since this simulator could execute hypercube programs written in C using pre-defined libraries, all existing hypercube algorithms and newly invented algorithms could be executed and tested in MISSE.

3. MULTI-LEVEL INTELLIGENT SYNTHESIS AND SIMULATION ENVIRONMENT

Multi-level Intelligent Synthesis and Simulation Environment (MISSE) provides the designer with a complete environment for the design of multiprocessor systems. MISSE adopts a top-down design methodology such that a global view of the system can be maintained, the system under design can be simulated, and its performance evaluated at each design level. As shown in Fig. 2, MISSE begins with the designer's system-level specification of a multiprocessor architecture, then it models, synthesizes, and simulates the system, and finally outputs the architecture-level configuration of the desired system.

As shown in Fig. 3, MISSE includes three main aspects of multiprocessor design: MP modeling, MP synthesis, and MP simulation. These three design aspects form the three branches of the MISSE Y-Chart, which is used to model the three phases of the design transformation process of an MP system as detailed in the following three subsections. MP modeling, as described in Section 3.1, is based on Rumbaugh's *Object Modeling Technique* [8]. For MP synthesis, as described in Section 3.2, two methodologies are proposed: *Performance Synthesis Methodology* (PSM) [9], [10] and *Intelligent Concurrent Object-Oriented Synthesis Methodology* (ICOS) [11]. PSM was proposed first and then ICOS, an extended work based on PSM, improved its synthesis efficiency by incorporating

Machine Learning and *Fuzzy Logic* in the design process and by adopting a distributed approach. Lastly, for MP simulation, a *Modularized Reconfigurable Simulator* is proposed in Section 3.3.

3.1 Multiprocessor Modeling

MISSE uses object-oriented techniques to model multiprocessor systems. Each system part is modeled as an individual component class which is stored in an object-oriented hierarchical part repository for future retrieval and use in the Multiprocessor Synthesis phase.

PSM uses an *Object Base* and a *Model Base*. The *Object Base* consists of component parts modeled as objects with two types of relationships among them, namely, aggregation and generalization [8]. The *Model Base* consists of memory organization based architecture models: UMA, NUMA, COMA, and NORMA [13] and model-related information. PSM uses the *Model Base* to map user specifications into architecture models which are further refined and implemented using object parts from the *Object Base*. On the other hand, ICOS relies on a *Class Hierarchy* which is basically similar to the *Object Base* used in PSM, except that a dynamic relationship among dependent component classes, known as "dependence," is also modeled. This dependence relationship helps to maintain the precedence among the objects to be synthesized in the distributed environment.

MISSE is a system-level design environment, hence the user input is basically system-level design specifications. Two different forms of input are used by PSM and ICOS. PSM allows the designer to input system-level specifications through Functional Models which is a combination of the Dynamic Model and Functional Model used in Rumbaugh's *Object Modeling Technique* (OMT) [8]. An example of the Functional Model is illustrated in Fig. 4. ICOS works at a more abstract level by providing the designer with a specification language. An example input specifications is given in Fig. 5, which corresponds to the Functional Model example in Fig. 4. Using the specification language, the designer can thus specify architecture, performance, and synthesis related requirements.

3.2 Multiprocessor Synthesis

Two synthesis methodologies: *Performance Synthesis Methodology* (PSM) and *Intelligent Concurrent Object-Oriented Synthesis Methodology* (ICOS) have been implemented in MISSE. PSM is an iterative, heuristic synthesis methodology, whereas ICOS is a distributed, fuzzy, intelligent synthesis methodology.

3.2.1 Performance Synthesis Methodology

Performance Synthesis Methodology (PSM) [9], [10] is a simple heuristic-based synthesis methodology for the

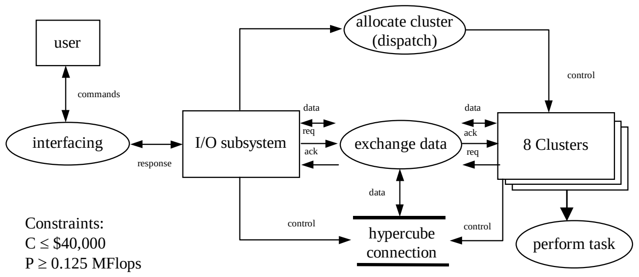


Fig. 4 PSM Input: Functional Model Example

system-level design of multiprocessor systems. PSM uses an *Object Base* and a *Model Base* for modeling and synthesizing system parts. *Colored Petri Nets* used in modeling system components and *Object Modeling Technique* used in the design process both contributed to the shortening of system development time and to the reduction of design cost. First, user specification consisting of functional models and performance constraints is translated into architecture models. Then, the system is configured by selecting the method of control, the memory organization, the type of processor, and the type of system interconnection. Finally, a heuristic design-space exploration algorithm is used to generate several near-optimal design alternatives. The best architecture is chosen by evaluating the design alternatives using a flexible *Performance Estimation Formula* (PEF) that mainly considers the system level design features, such as system throughput (*T*), utilization (*U*), reliability (*R*), scalability (*S*), fault-tolerance (*F*), and cost (*C*).

$$\mathrm{PEF} = \frac{T \times R \times S \times F}{U \times C} \quad (1)$$

Several systems were successfully synthesized using this top-down object-oriented methodology, thus showing its feasibility as a design automation tool for parallel systems.

3.2.2 Intelligent Concurrent Object-Oriented Synthesis Methodology

After the successful implementation of PSM in the system

architecture:
  system:
    Architecture Type = Message-Passing,
    Control Type = SIMD,
    Memory Type = Distributed-Unshared,
    System Interconnect = Hypercube,
    System Processors = 32, *No. of Clusters* = 8
  cluster:
    Processor Unit = RISC,
    Cluster Interconnect = Bus,
    Cluster Processors = 4
performance:
    Cost ≤ $40,000, *Power* ≥ 0.125 Mflops
synthesis:
    Machine Learning = Yes

Fig. 5 ICOS Input: Specification Example

level design of MP systems, we have further extended the work to improve its synthesis efficiency and to produce more balanced designs. Besides enhancing the use of more object-oriented techniques, various other techniques, such as distributed synthesis, fuzzy design-space exploration, fuzzy specification-guided learning, and example-guided learning have all been incorporated into the design scheme. The resulting work was *Intelligent Concurrent Object-Oriented Synthesis Methodology* (ICOS) [11].

First, object-oriented relationships such as *aggregation*, *generalization*, and *dependence*; and object-oriented operators such as *iterator*, *generator*, and *updator*; both are used in the methodology to guide the process of component synthesis. Second, instead of following the commonly used centralized control of synthesis tasks, a distributed control approach is adopted to manipulate the exponential exploration of design space. Using distributed control in ICOS necessitates checking for design completion and synthesis rollback, for which a *Multi-token Object-oriented Bi-directional net* (MOBnet) model [12] was proposed. Third, since the comparison between two or more designs is not a crisp decision, both the design-space exploration and the identification of similar design configurations during learning should be fuzzified for ease of comparison. Finally, the application of machine learning makes the methodology intelligent enough to reuse the experiences of previous synthesis configurations. Experiments show all these applied techniques contribute to the synthesis efficiency and the degree of automation.

3.3 Multiprocessor Simulation

Multiprocessor simulation is much more complicated than uniprocessor simulation which itself is already quite a complex task. Different memory organization models, system interconnections, and the interaction between processors are some features exclusive to multiprocessors, which must be correctly and efficiently simulated such that the design performance can be more accurately estimated at the end of each design level in MISSE.

There are two purposes of simulation in MISSE: (1) the performance evaluation of design alternatives during the design-space exploration, (2) the final validation of estimated performance at the end of each design level.

3.3.1 Modularized Reconfigurable Simulator

Modularized Reconfigurable Simulator (MRS) is an object-oriented flexible simulation tool used in MISSE. Both hardware and software multiprocessor parts are modeled as subsimulators. Each subsimulator has

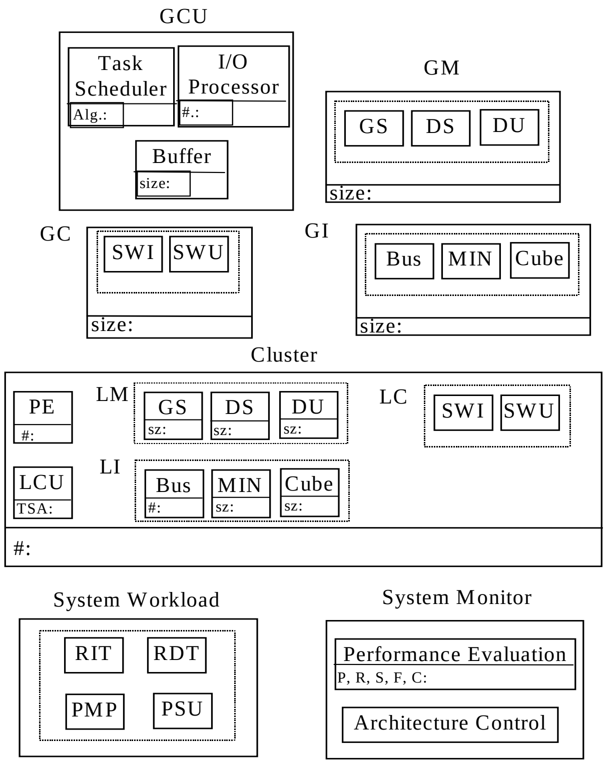


Fig. 6 Unconfigured Modularized Reconfigurable Simulator Architecture

simulation attributes which determine the characteristics and the number of a component to be simulated. The unconfigured MRS architecture, as shown in Fig. 6, consists of groups of subsimulators. A group, as enclosed within a dotted-line box, consists of subsimulators representing the specializations or instances of a multiprocessor part. Besides the actual hardware and software parts of a multiprocessor, there are two more components: the system workloads and the performance monitor. A combination of the system workloads is used to generate the simulation workload, and the function of the system monitor is to collect dynamic performance information of the system under simulation.

3.3.2 MRS Simulation Procedure

As shown in Fig. 7, the input to MRS is the detailed configuration of the system as synthesized in the MP Synthesis phase of MISSE. This information is used to configure the MRS architecture. Each simulation attribute is configured according to the input information which may consist of the type of memory organization (shared or unshared, global or distributed), the kind of global interconnection, the total number of processors, etc. One subsimulator is selected from each group of subsimulators. An example of a configured MRS architecture is shown in Fig. 8.

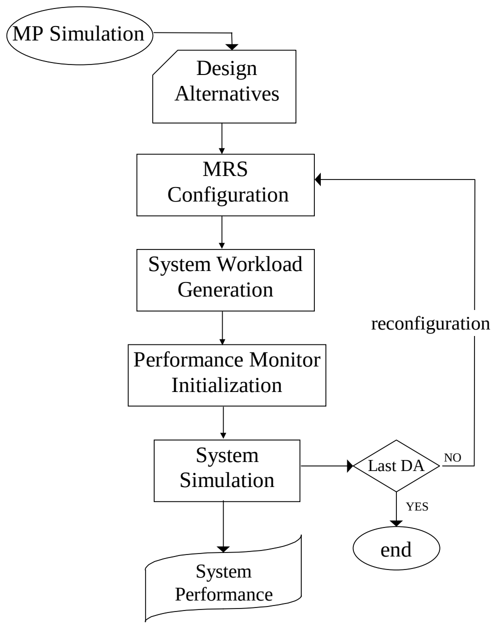


Fig. 7 MRS Simulation Procedure

A simulation workload is also generated using the system workloads component of MRS, which contains several predefined workloads, such as random independent tasks, random dependent tasks, pre-determined memory access proportions, and pre-determined system interconnection utilization. One or more of these workloads are used to create a simulation workload which when used in the simulation process gives the desired performance figures. Fig. 9 shows the various workloads, the different performance characteristics, and the relationships among them. By executing MRS with one or more workloads, one can evaluate the corresponding performance characteristics of a design alternative. Different types of workloads are used for the evaluation of different performance characteristics such that the results can be more accurate. Once the MRS is configured and the simulation workload generated, the system monitor is initialized so that it can begin collecting the desired performance results of the system under simulation. Finally, the configured *Modularized Reconfigurable Simulator* is executed with the generated simulation workload and the performance readings collected by the initialized performance monitor.

3.3.3 Advantages of MRS

MRS was developed using the basic fact that any two design alternatives of the same system usually differ in only one or two design characteristics. This small difference between design alternatives allows the easy transition from the simulation of one design alternative to that of another through a simple reconfiguration of MRS. The reconfiguration of MRS involves direct replacements of subsimulator modules and/or reconfiguration of simulation attributes of currently used subsimulator modules. For example, if one design alternative was a 1024-processor hypercube and another was a 1536-processor system connected by a generalized cube with no

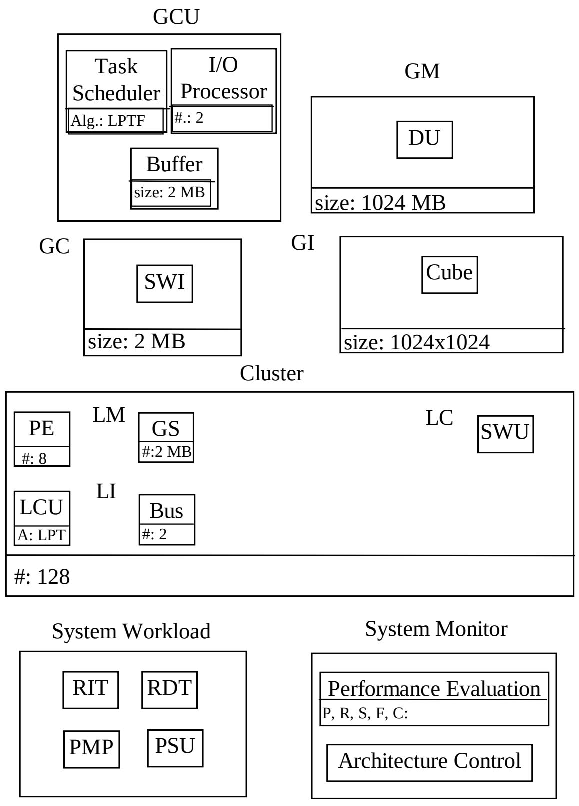


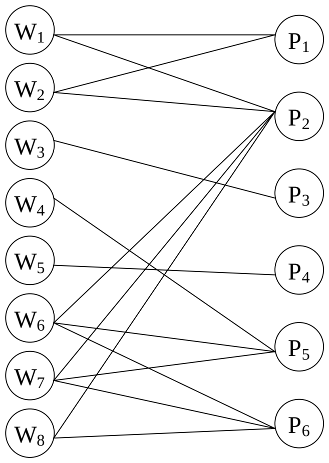
Fig. 8 Configured Modularized Reconfigurable Simulator Architecture

difference in any other design characteristics, then the former can be reconfigured into the latter by replacing the hypercube system interconnection subsimulator module with a generalized cube MIN subsimulator module and the number of PE in the Cluster subsimulator reconfigured from 1024 to 1536. On reconfiguration of MRS, the simulation workloads have to be regenerated and the performance monitor re-initialized so that the system can be re-simulated.

Besides the ease of design transformation, MRS is also flexible and extendible. New architecture, protocols, and components can be easily included in MRS by creating the corresponding subsimulator modules. MRS is both technology and implementation independent since no technology specific information is used and the implementation of the subsimulators into hardware or software components are not implied. The architecture of an example hypercube simulator is shown in Fig. 10.

3.4 MISSE Design Levels

MISSE follows a top-down design methodology with each design cycle consisting of modeling, synthesis, and simulation. This cycle occurs at each design level. Between two design cycles, levels change from a higher design level into a more detailed design level. As shown in Fig. 11, MISSE considers four design levels: system level, cluster level, node level, and instruction level. First, at the system level, the user-given system level



W_1: Random Independent Parallel Tasks
W_2: Random Dependent Parallel Tasks
W_3: Predetermined Memory access Proportions
W_4: Predetermined System Interconnection Utilization
W_5: Fault Testing
W_6: High Comp/Comm Ratio
W_7: High Comm/Comp Ratio
W_8: Comp/Comm $\cong$ 1

P_1: System Throughput
P_2: System Utilization
P_3: Average Memory Access Latency
P_4: Fault-tolerance
P_5: Communication Overhead
P_6: System Efficiency

Fig. 9 Workloads and Performance Factors used in MRS

specifications are analyzed and an object-oriented MP model created, then either PSM or ICOS is applied to synthesize this model into a system-level description of the MP architecture which may include the total number of processors, the total amount of RAM, the global interconnection (shared bus, multistage interconnection network, or hypercube), and the control scheme (SIMD or MIMD, synchronous or asynchronous). This system level MP architecture is simulated by configuring MRS based on the architecture description.

Second, once the system level performance characteristics are verified through simulation, the MP model is then perceived at the cluster level, where the MP model is synthesized into a cluster level architecture description which may consist of the number of processors per cluster, the amount of cluster memory, the type of cluster interconnection, and the cluster control scheme. MRS is now configured to simulate the various design alternatives at the cluster level. One or more of the alternatives are chosen for further synthesis. Then, the MP model is viewed at the node level. A *node* is the basic computation

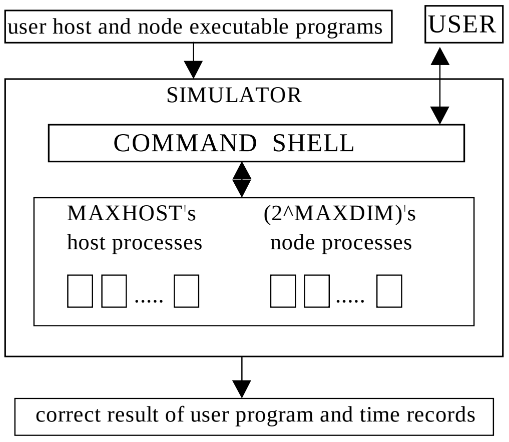
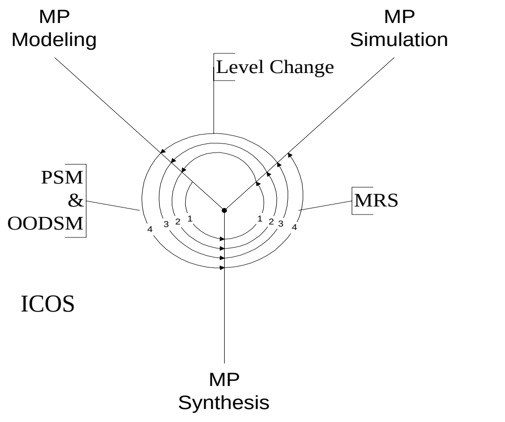

user host and node executable programs
USER
SIMULATOR
COMMAND SHELL
MAXHOST's host processes
(2^MAXDIM)'s node processes
.....
.....
correct result of user program and time records

Fig. 10 Hypercube Simulator Architecture

unit in MISSE and includes a CPU, local memory, and local cache. Synthesis at this level results in a node level description of the system, consisting of the type of CPU and the amount of local memory and local cache. Simulation at this level provides a more accurate estimation of the system performance since the type of CPU and the other node level characteristics are already determined. Finally, at the instruction level, the MP model is viewed as a collection of interacting instruction set processors. Since MISSE is basically a system level design environment, the CPU is chosen from existing ones, rather than resynthesized, hence there is no actual synthesis at this level, but this level is still considered such that actual parallel programs can be simulated and a more realistic performance estimation of the MP system obtained. This simulation may be trace-driven using program traces.



1: System Level, 2: Cluster Level, 3: Node Level, 4: Instruction Level

Fig. 11 MISSE Design Levels

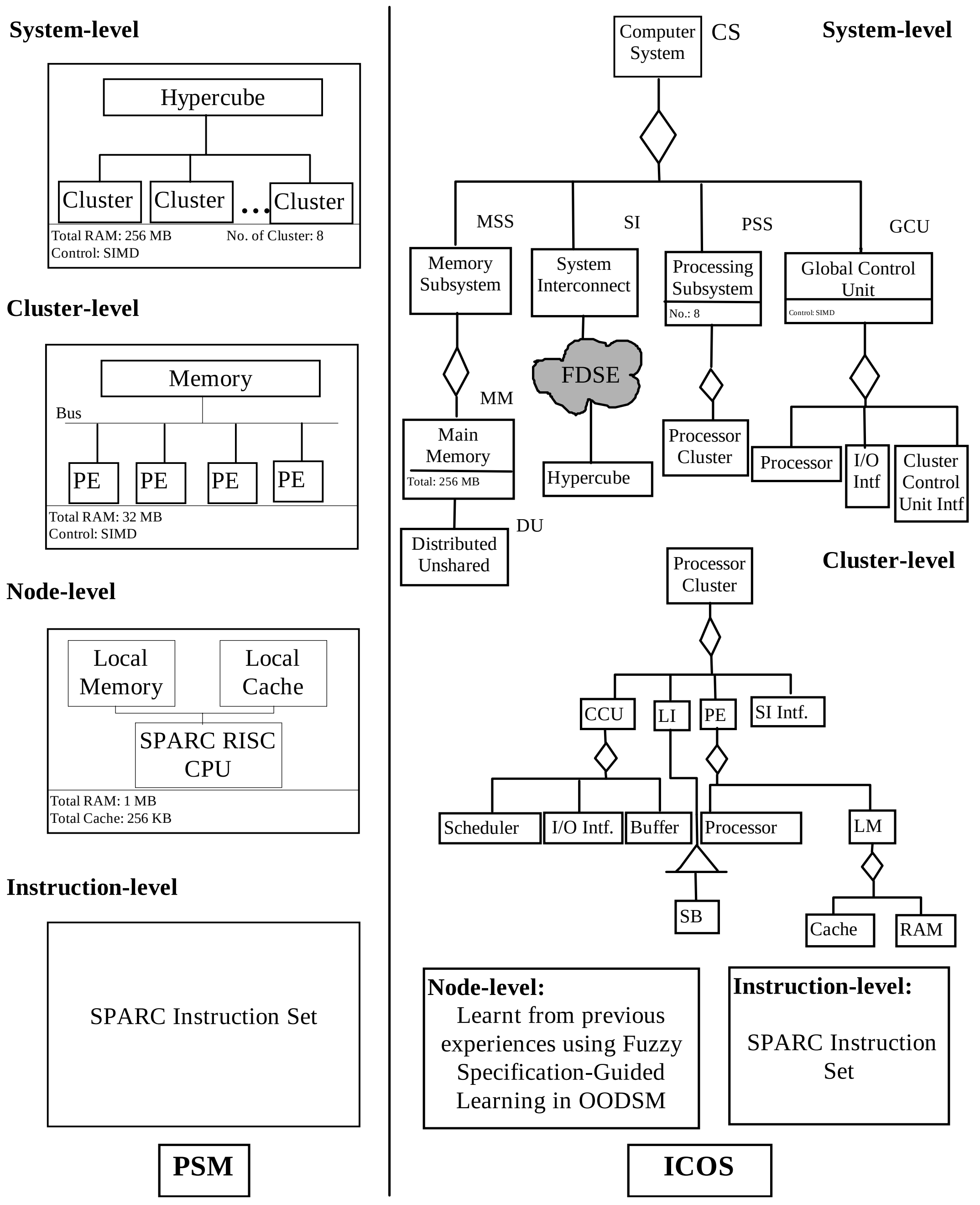


Fig. 12 Multi-level Design Example

The input Functional Model and the specification input illustrated in Fig. 4 and Fig. 5, respectively, were synthesized using MISSE. The results of synthesis at each design level are shown in Fig. 12 for both PSM and ICOS. The system level design results in an 8-cluster system interconnected using the hypercube connection, the method is SIMD, and the total system memory is 256 MB. The cluster level design results in each cluster containing 4 processing elements (PEs) interconnected by a shared bus with 32 MB of cluster memory. The node level design results in each node having a SPARC RISC CPU, 1 MB local memory, and 256 KB local cache.

4. CONCLUSION AND FUTURE WORK

MISSE, a *Multi-level Intelligent Synthesis and Simulation Environment*, provides the designer with a complete environment for the system level design of multiprocessor systems. Using such an environment increases the productivity of a designer as it reduces the design time and eliminates the interchange of design formats between two successive design phases. The successful application of object-oriented techniques in all the three phases of MP design has not only simplified the manipulation of system components, but also increased synthesis efficiency [11].

Besides the design of complete MP systems, MISSE can also be used for verifying design characteristics of partial systems or system parts such as the introduction of a new interconnection architecture and the verification of its capability of fault-tolerance. In addition to hardware design, software can also be tested within the MISSE environment. Example such as multiprocessor task scheduling algorithms and parallel programs were given in this article. One of the most important future work is the integration of software development and testing techniques into MISSE such that the hardware-software codesign [17] of multiprocessor systems can be made feasible.

REFERENCES

[1] C. H. Chen and S. J. Chen, "An Alternative Extra Stage Structure to Increase the Reliability of MIN," *Proc. of International Computer Symposium,* December 1991.

[2] J. F. Lin and S. J. Chen, "Scheduling Algorithm for Non-preemptive Multiprocessor Tasks," *Computers and Mathematics with Applications,* Vol. 28, No. 4, pp. 85-92, 1994.

[3] J. F. Lin and S. J. Chen, "Scheduling Parallel Tasks on Hypercubes," *Electronic Letters,* Vol. 30, No. 11, pp. 841-842, 1994.

[4] J. F. Lin, W. B. See, and S. J. Chen, "Performance Bounds on Scheduling Parallel Tasks with Communication Cost," *IEICE Trans. on Info. & Sys.*, Vol. E78-D, No. 3, pp. 263-268, March 1995.

[5] J. F. Lin, W. B. See, S. J. Chen, "Scheduling Parallel Tasks with Setup Time on Hypercube Systems," *Proc. of International Computer Symposium*, December 1994, pp. 689-693.

[6] J. F. Lin and S. J. Chen, "An Analysis of Multiprocessor Tasks Scheduling," to appear in *Computer Systems Science and Engineering*,

[7] T. Y. Lee, M. J. Tsai, and S. J. Chen, "The Designing of a Simulator for Hypercube Supercomputer," *Proc. of the 1995 Workshop on High Performance Multiprocessor Systems*, HsinChu, Taiwan, ROC, July, 1995, pp. 248-254.

[8] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, *Object-Oriented Modeling and Design*, Prentice-Hall, Englewood Cliffs, 1991.

[9] P. A. Hsiung, S. J. Chen, T. C. Hu, and S. C. Wang, "PSM: An Object-Oriented Synthesis Approach to Multiprocessor System Design," *IEEE Trans. on VLSI Systems*, Vol. 4, No. 1, pp. 83-97, March 1996.

[10] P. A. Hsiung and S. J. Chen, "Object-Oriented Synthesis Application Tool," *Fourth Workshop on Object-Oriented Technology,* September 1994.

[11] P. A. Hsiung, C. H. Chen, T. Y. Lee, and S. J. Chen, "ICOS: An Intelligent Concurrent Object-Oriented Synthesis Methodology for Multiprocessor Systems," *ACM Trans. On Design Automation of Electronic Systems*, Vol. 3, No. 2, to appear in April 1998.

[12] P. A. Hsiung, T. Y. Lee, and S. J. Chen, "MOBnet: An Extended Petri Net Model for the Concurrent Object-Oriented System-Level Synthesis of Multiprocessor Systems," *IEICE Trans. On Information and Systems*, Vol. E80-D, No. 2, pp. 232-242, February 1997.

[13] K. Hwang, *Advanced Computer Architecture*, McGraw-Hill Inc., Singapore, 1993.

[14] C. A. Petri, "Kommunikation Mit Automaten," Schriften des IIM Nr. 2, *Institut fur Instrumentelle Mathematik*, Bonn, 1962. English Translation: Technical Report RADC-TR-65-377, Griffiss Air Forse Bas, New York, Vol. 1, Suppl. 1, 1966.

[15] K. Jensen, "Colored Petri Nets and The Invariant Method," *Theoretical Computer Science*, Vol. 14, pp. 317-336, 1981.

[16] D. R. Jefferson, "Virtual Time," *ACM Trans. on Programming Languages and Systems*, Vol. 7, No. 3, pp. 404-425, July 1985.

[17] P. A. Hsiung, "CMAPS: A Cosynthesis Methodology for Application-Oriented Parallel Systems," accepted to appear in *ACM Trans. on Design Automation of Electronic Systems.*