Parallel Object-Oriented Synthesis Environment Based On Message-Passing

Pao-Ann Hsiung
Institute of Information Science
Academia Sinica, Taipei, Taiwan.

Abstract

A system-level computer design environment is proposed by integrating parallel design techniques and object-oriented technology. The system parts are modeled using object-oriented technology, such that not only are the static features of the components encapsulated, but also are the dynamic design states. A system is designed by empowering each object class with design autonomy, thus leading to a distributedly-controlled environment where objects actively seek to synthesize themselves in parallel using messages. System synthesis is accomplished and related problems are solved by eight different kinds of messages passed among the objects. Problems inherent to parallel design, such as emptiness and deadlock are also solved.

1: Introduction

Traditionally, systems were synthesized in a sequential manner, that is, only one design was synthesized at a time. This often resulted in a large overall design time and delayed detection of infeasible specifications. To remedy such a situation, system parts must be synthesized simultaneously and design alternatives produced in parallel. The application of parallel techniques often increases the efficiency of an executing process. But, at the same time some overhead of parallel data manipulations are incurred. Furthermore, parallel techniques also introduce inconsistencies among participating actors. These two issues are inherent to parallelism. Both of these issues can be easily handled when we use object-oriented techniques for static modeling as well as dynamic manipulation. In this paper, a novel way of increasing the efficiency of computer system design automation using parallel techniques, with a negligible overhead and a consistent design result, is proposed. It basically is an integration of object-oriented and parallel design techniques.

Parallel design technique is mainly the concurrent design of more than one system parts. An *active* approach is proposed. Objects *actively* seek to design themselves once they have all the required specifications. Architectural dependencies among parts, such as the data transfer speed, the channel bandwidth, etc are all modeled as relationships in a Class Hierarchy [5]. Consistency among the parts is maintained by transmitting the specifications using messages.

There have been several methodologies and tools developed to either fully or partially automate the design process at various levels of hardware system design such as the MICON system [4], the *Megallan* system [3] including the *System Architect's Workbench* [13], WOLFIE, and LASSIE [14], *Performance Synthesis Methodology* (PSM) [6] and *Intelligent Concurrent Object-oriented Synthesis* (ICOS) methodology [5], to mention a few. The problem of synthesis efficiency had been tackled by introducing heuristics into the design methodology. Rather than searching the design-space exhaustively, several techniques have been proposed in the past to partially explore the region that most likely contains the optimal design solution. The techniques include fuzzy logic [9], learning [12], object-oriented design [10], object-oriented language [2], specification reuse [1], and formal approaches [7].

With the increasing wide-spread use of object-oriented technology in software modeling and development, there has been a technology transfer from software to hardware [8]. Application of OO to hardware synthesis has matured from a simple modeling as in PSM [6] to a complex design methodology such as ICOS [5]. OO has also been applied in the formal analysis of systems [11] and synthesis [8]. Though OO has been applied, but the actual method of application is not very clear from the previous work. With the increased use of parallel computers, it is naturally desirable to parallelize the synthesis process in order to design more complex and larger systems. In contrast to the previous work, the current work does not lose optimality in the solutions for efficiency in the design process. Object-oriented modeling and design techniques coupled with the parallel design process illustrate an efficient design environment as evidenced by practical implementation [5] and formal analysis [7]. Solutions to the emptiness and deadlock problems are also proposed.

Due to page-limit, we mainly delve on the solutions to the two problems inherent to parallelism. The paper is organized as follows. Section 2 shows how some parallel design related problems such as *emptiness* and *deadlock* are solved. Section 3 gives an application example. Section 4 concludes the paper with some future work.

2: Object-oriented parallel design related problems

The objects in the Class Hierarchy are classified into three types of nodes: *Aggregate* node (A-node), *Generalized* node (G-node), and *Physical node* (P-node) [6]. Totally, eight types of messages are used for different types of communication among the objects. Classified into three groups, there are *synthesis-related*, *update-related*, and *rollback-related* messages. There are three synthesis-related messages: `synthesize`, `synthesis-complete`, and `synthesis-incomplete`; two update-related messages: `update` and `update-complete`; and three rollback-related messages: `rollback`, `rollback-complete`, and `rollback-incomplete`. All messages are implemented as method invocations.

2.1: Design completion check

A system designer stipulates his or her requirements by giving system-level specifications including performance-related constraints such as the minimum throughput, the maximum cost, the utilization factor, and architecture-related constraints such as the system interconnection, the amount of main memory, the memory organization, et al. In terms of the current technology, the given specifications may either by feasible or infeasible. Sometimes, infeasibility due to obvious contradictions and errors is detected easily by a pre-design specification analysis. But at times, infeasibility may go undetected until the design process has well advanced into some intermediate stage. Hence, it is desirable to detect infeasible specifications at the earliest-possible stage of the design process. We call this the *emptiness* problem. A practical solution called the *design-completion check* for the emptiness problem is presented.

Since the synthesis process is a *distributed-control parallel-design* process, a mechanism is needed to ensure that a particular design is either feasible and complete or infeasible and incomplete. This design completion check process is accomplished using two types of messages or method invocation calls, namely, `synthesis-complete` and `synthesis-incomplete`.

As the synthesize message is gradually propagated in a top-down direction and broadcast in a breadth-first-search hierarchy traversal, a synthesize message eventually reaches a P-node at the hierarchy leaf. The three types of nodes behave in the following ways.
(a) Whenever a P-node receives a synthesize message, it starts to instantiate itself. If this instantiation is feasible and complete, the P-node sends a synthesis-complete message upwards to its parent node in the Class Hierarchy; otherwise, the P-node performs a *synthesis rollback* action as described in the next subsection.
(b) Whenever a G-node receives at least one synthesis-complete message from one of its child objects and synthesis-incomplete messages from the other child objects, the G-node sends a synthesis-complete message to its parent object. If no synthesis-complete message is received from its child nodes, then the G-node performs a *synthesis rollback* action.
(c) Whenever an A-node receives a synthesis-complete message from each of its child objects that has been sent a synthesize message, the A-node sends a synthesis-complete message to its parent object. If the A-node receives a synthesis-incomplete message from any one or more of its child objects, it first performs a *synthesis rollback* action.

In all the above cases, if the rollback fails, a synthesis-incomplete message is sent upwards to its parent class by the node object. The above design completion check process will finally result in either a synthesis-complete or a synthesis-incomplete message being received at the root node that represents the whole computer system. In the former case, the design is feasible under the current constraints and specifications, while in the latter case, it is infeasible.

2.2: Synthesis rollback

Since it is a top-down parallel design environment, specifications are propagated from the top of the class hierarchy towards the leaf physical classes. In the course of this propagation, it may happen that some component cannot be synthesized under the derived specifications as propagated by its parent class. This is called the *deadlock* problem since the parent requires its child to satisfy certain specifications while the child cannot do so. A *rollback mechanism* is proposed to solve this problem. Two types of messages: rollback and rollback-complete, are used.

Whenever an object, either an A-node or a G-node, receives a rollback message along with the object characteristics that have triggered the rollback and the range of values acceptable for each of the object characteristics, the object then behaves as follows.
Case a) If by resynthesizing itself, the object can relax the concerned specifications and characteristics to satisfy all the constraints of the parent object, the associated object(s), and the child object(s), then it will resynthesize itself and relax the specifications and characteristics. After synthesis, the object will send a confirmation in the form of a rollback-complete message to the sender of the rollback message.
Case b) If the object cannot relax the specifications and characteristics, then it will propagate the rollback message along with the related information to its parent object and/or the associated objects. The rollback process could be quite expensive as it might propagate throughout the whole design and result in resynthesizing the whole system from the start. In order to avoid this, the values of the object characteristics are specified as ranges or enumerations whenever possible.

On receiving a rollback-complete message, if the object itself had been the receiver of a rollback message, it will pass on the rollback-complete message to the rollback message sender. In this way, all rollbacks are confirmed and finally the original message sender will receive a confirmation. If a rollback fails, a synthesis-incomplete message is sent.

3: Application

The proposed design technique has been implemented as a working design framework in which the recently proposed *Intelligent Concurrent Object-oriented Synthesis* (ICOS) methodology [5] for system-level synthesis of parallel systems was developed. Besides practical implementation, it has also been theoretically modeled, validated, and analyzed by the authors using high-level Petri nets called *Multi-token Object-oriented Bi-directional net* (MOBnet) [7]. ICOS and MOBnet were respectively concerned with the synthesis methodology and the model analysis. The general design environment incorporating parallel and OO techniques was not presented before. An example is given here to understand the processes of the design completion check and synthesis rollback.

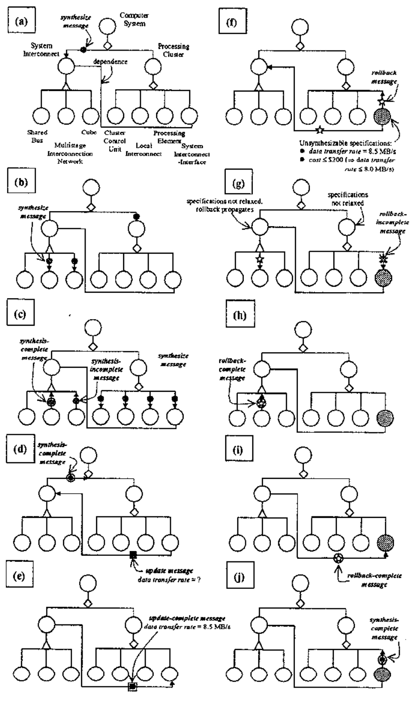
(a)
synthesize message
Computer System
System Interconnect
Processing Cluster
dependence
Shared Bus
Multistage Interconnection Network
Cube
Cluster Control Unit
Local Interconnect
Processing Element
System Interconnect -Interface
(b)
synthesize message
(c)
synthesis-complete message
synthesis-incomplete message
synthesize message
(d)
synthesis-complete message
update message
data transfer rate = ?
(e)
update-complete message
data transfer rate = 8.5 MB/s
(f)
rollback message
Unsynthesizable specifications:
data transfer rate = 8.5 MB/s
cost ≤ $200 (⇒ data transfer rate ≤ 8.0 MB/s)
(g)
specifications not relaxed, rollback propagates
specifications not relaxed
rollback-incomplete message
(h)
rollback-complete message
(i)
rollback-complete message
(j)
synthesis-complete message

Figure 1. Application Example

The target multiprocessor system is an SIMD machine with a `Multistage Interconnection Network` (`MIN`) as the system interconnection. The system specifications include a *data transfer rate* of at least 8 MB/s for each block of `MIN`, a total throughput of at least 64 MB/s, and a maximum total cost of $7,000. These specifications are input to ICOS. We will mainly concentrate on how situations with unsynthesizable specifications are handled. As shown in Fig. 1, (a) and (b) illustrate the synthesis process, (c) and (d) illustrate design completion check, (d) and (e) illustrate the specification update process, (f)–(j) illustrate synthesis rollback. Since parts (a)–(e) of Fig. 1

are straight-forward and self-explanatory, we only explain in detail the parts (f)–(j).

The specifications derived from `Processing Cluster` (`PC`) for `System Interconnect Interface` (`SI-Intf`) include a maximum *cost* of $200, which could only result in an interface that has a maximum *data transfer rate* (*dtr*) of only 8.0 MB/s. This is in contradiction to the *dtr* specification *updated* from `System Interconnect` (`SI`) in Fig. 1 (e) (which is 8.5 MB/s). Thus, `SI-Intf` is unsynthesizable in such a situation. It makes requests for synthesis rollback to both `PC` and `SI` (Fig. 1 (f)). `PC` cannot increase the cost of `SI-Intf` due to other cost constraints (Fig. 1 (g)). Meanwhile, `SI` propagates the rollback message to `MIN` (Fig. 1 (g)). `MIN` decreases the *dtr* specification to 8 MB/s as required (Fig. 1 (h)). Finally, rollback completes informing `SI-Intf` of the change in the *dtr* specification value (Fig. 1 (i)). Thus, `SI-Intf` can now synthesize itself under the derived *cost* of $200 and updated *dtr* of 8 MB/s (Fig. 1 (j)).

4: Conclusion and future work

An elegant integration of object-oriented and parallel design techniques has resulted in an efficient design environment and at the same time has solved the emptiness and deadlock problems of parallelism. Being a general design environment, one of its advantages is that it can be easily incorporated into any system design automation methodology such as PSM [6] and ICOS [5]. Experimental results of using the proposed design technique in ICOS has shown how the overall design time can be sped up by a factor of two to three as compared to PSM. Besides the practical efficiency, it was also shown how two parallel design related problems, namely, emptiness and deadlock are solved. It was also formally validated and analyzed in a related work [7].

References

[1] V. De Antonellis and B. Pernice. Reusing specifications through refinement levels. *Data and Knowledge Engineering*, 15(2):109–133, April 1995.

[2] M. J. Chung and S. Kim. An object-oriented VHDL design environment. In *Proc. 27th ACM/IEEE Design Automation Conference*, pages 431–436, 1990.

[3] A. J. Gadient and D. E. Thomas. A dynamic approach to controlling high-level synthesis CAD tools. *IEEE Trans. on VLSI Systems*, 1(3):328–341, September 1993.

[4] A. P. Gupta, W. P. Birmingham, and D. P. Siewiorek. Automating the design of computer systems. *IEEE Trans. on CAD*, 12(4):473–487, April 1993.

[5] P.-A. Hsiung, C.-H. Chen, T.-Y. Lee, and S.-J. Chen. ICOS: An intelligent concurrent object-oriented synthesis methodology for multiprocessor systems. *To appear in ACM Trans. on Design Automation of Electronic Systems*, 3(2), April 1998.

[6] 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*, 4(1):83–97, Mar. 1996.

[7] 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 Info and Syst*, E80-D(2):232–242, Feb. 1997.

[8] P.-A. Hsiung, T.-Y. Lee, and S.-J. Chen. Object-oriented technology transfer to multiprocessor system-level synthesis. In *Proc. 24th Int'l Conference on Technology of Object-Oriented Languages and Systems*, Sept. 1997.

[9] E. Q. Kang, R.-B. Lin, and E. Shragowitz. Fuzzy logic approach to VLSI placement. *IEEE Trans. on VLSI Systems*, 2(4):489–501, December 1994.

[10] S. Kumar, J. H. Aylor, B. W. Johnson, and Wm. A. Wulf. Object-oriented techniques in hardware design. *IEEE Computer*, 27(6):64–70, June 1994.

[11] Y. K. Lee and S. J. Park. OPNets: An object-oriented high-level Petri net model for real-time system modeling. *Journal Systems Software*, 20:69–86, 1993.

[12] T. M. Mitchell, S. Mahadevan, and L. I. Steinberg. LEAP: A learning apprentice for VLSI design. In *Proc. 9th IJCAI*, pages 573–580, 1985.

[13] D.E. Thomas, E.M. Dirkes, R.A. Walker, J.V. Rajan, J.A. Nestor, and R.L. Blackburn. The system architect's workbench. In *Proc. 25th ACM/IEEE Design Automation Conference*, pages 337–343, June 1988.

[14] M.T. Trick and S.W. Director. Lassie: Structure to layout for behavioral synthesis tools. In *Proc. 26th ACM/IEEE Design Automation Conference*, pages 104–109, June 1989.