International Conference on Computer Design, October 1992

ProperCAD: A Portable Object-oriented Parallel Environment for VLSI CAD*

Balkrishna Ramkumar and Prithviraj Banerjee
Center for Reliable and High-Performance Computing,
Coordinated Science Laboratory, University of Illinois,
Urbana, IL 61801

Abstract

Most parallel algorithms for VLSI CAD proposed to date have one important drawback: they work efficiently only on machines that they were designed for. As a result, algorithms designed to date are dependent on the architecture for which they are developed and do not port easily to other parallel architectures.

This paper describes a **P**o**r**table **o**bject-oriented **p**arallel **e**nvi**r**onment *for* **CAD** *algorithms (***ProperCAD***) under development at University of Illinois. The objectives of this research are two-fold. (1) To develop new parallel algorithms that are portable. We accomplish this by writing the algorithms using the ProperCAD environment: a library of functions that permits portability of parallel CAD algorithms across MIMD machines. Programs written using this environment run* unchanged *on all parallel machines for which this environment is available. (2) To design the parallel algorithms around a good sequential algorithm with a well-defined parallel-sequential interface. This will permit the parallel algorithm to benefit from future developments in sequential algorithms.*

1 Introduction

In view of the increasing complexity of VLSI circuits of the future, the requirements on VLSI CAD tools will continuously increase. Parallel processing for CAD applications is becoming gradually recognized as a popular vehicle to support the increasing computing requirements of future CAD tools. Recent research on parallel CAD applications have been reported for a wide variety of applications such as placement [2, 8, 14, 15], floor planning [6], circuit extraction [3, 9, 18], test generation and fault simulation [11], etc. Parallel processing for VLSI CAD has become a reality in industry as well. Hardware vendors such as Solbourne have already announced products with multiple CPUs in a single workstation. Software CAD vendors such as Mentor have announced products such as CHECKMATE, a parallel design rule checker using multiprocessing, to accelerate a single job. A major limitation with almost all such previous work is that the parallel algorithms have been targeted to run on specific machines like an Intel iPSC/2 hypercube or an Encore shared memory multiprocessor. Such work, although interesting, is not usable by the rest of the VLSI CAD community since the algorithms are not portable to other machines.

A second serious problem also presents itself in the design of parallel algorithms. The software development cycle for parallel algorithms is considerably longer than for sequential algorithms. This has two important implications. The first is that they are considerably more costly to develop than sequential algorithms. This is only exacerbated by the lack of portability across parallel machines. The second implication is a more pragmatic one. Given the fast pace of progress in the development and improvement of sequential algorithms for CAD applications, for a given application, sequential algorithms frequently outperform parallel algorithms due to the longer development time of the latter. For example, this is evident in parallel test pattern generation. The latest version of HITEC [10], a uniprocessor test pattern generation program for sequential circuits is already comparable in performance and is slightly better in quality of results than a recent parallel algorithm for test pattern generation [11].

A related issue in the development of parallel algorithms is that certain approaches are inherently parallelizable and others are extremely hard to parallelize. More often than not, the tradeoff between these two approaches is in the quality of results. Cell placement is a good example. The quadrisection al-

*This research was supported in part by the National Aeronautics and Space Administration under grant NAG 1-613, and in part by the Semiconductor Research Corporation under grant SRC 91-DP-109.

gorithm [17] is easily parallelizable and is significantly faster than algorithms based on simulated annealing. Unfortunately, it cannot produce results comparable to TimberWolf [16], a sequential technique that uses simulated annealing (and a host of related tricks). An interesting possibility would be to use a hybrid of these two (and possibly other) techniques, where, for example, quadrisection could be used for decomposition of the layout area into regions, and TimberWolf would be used for placement in a given region[1]. However, to experiment with such techniques, it should not be necessary to rewrite the software entirely. Any attempt to rewrite TimberWolf [16] will not only be extremely time consuming, it is also unlikely to be comparable in performance. However, if it is possible to decouple the parallel and sequential algorithms and provide a well-defined interface between the two, the TimberWolf software can be substantially reused, making it practical to experiment with hybrid schemes such as these.

It would be presumptuous to assume that it will be trivial to interface the parallel algorithm with the sequential algorithm as described above. For this to be practical, it is imperative that sequential algorithms be written in a modular fashion. Fortunately, object-oriented programming in C++ (or even disciplined C programming) goes a long way in realizing this requirement. Many CAD vendors are already rewriting many of their well-established CAD applications using such disciplined, modular programming methods due to the benefits offered in program design and maintenance.

The most important questions that need to be addressed in the development of parallel algorithms are therefore: "How can we design parallel algorithms that are truly portable across parallel machines?", "How can we exploit good sequential algorithms in the design of parallel algorithms", and "How can parallel algorithms keep pace with future developments in sequential algorithms?" These are the main objectives of a new project to be discussed in this paper. We are developing a **Por**table **o**bject-oriented **p**arallel **e**nvi**r**onment for **CAD** applications (**ProperCAD**). We describe how it is possible to design parallel algorithms keeping these objectives in mind. We demonstrate this briefly with three parallel CAD applications: flat circuit extraction, test pattern generation for sequential circuits, and combinational logic synthesis..

[1]We do not claim that such an approach is original.

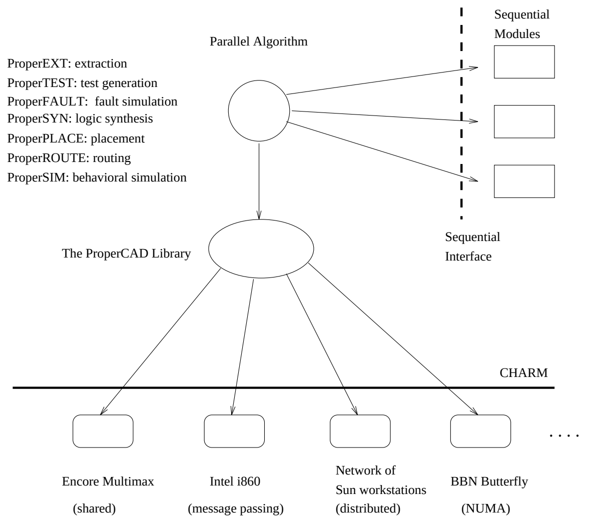

Figure 1: A high-level view of the ProperCAD project. We list the CAD applications under development above.

2 The ProperCAD Environment

ProperCAD is essentially a set of library routines that forces a programmer to write parallel programs in an object-oriented style using a coarse-grained data-flow style of execution[2]. It abstracts away all machine dependent features from an application program and provides a uniform set of routines that can be used by the programmer to render the application portable across parallel MIMD machines. Features like dynamic object creation, inter-object communication using messages, mapping of objects to processors and dynamic load balancing are all available and can be used at the discretion of the programmer. The environment has been implemented in the most efficient manner possible on different MIMD machines. Thus, CAD applications written using these library routines will port *without change* on all the parallel machines for which the ProperCAD environment is available (see Figure 1).

ProperCAD supports a message-driven style of execution. Conceptually, a pool of messages is maintained, where each message represents work. A message may request the creation of a new instance of

[2]The ProperCAD environment builds upon a system for portable parallel programming called *CHARM* [5, 7]. Earlier versions of CHARM went by the name of the *Chare kernel*. Our research aims to specialize this system for parallel CAD applications. A C++ version of the ProperCAD environment is under implementation.

an object, thereby dynamically creating object instances. Messages may also be exchanged between existing objects where an operation *Op* in an object is invoked by sending a message to the object destined for *Op*, containing the parameters expected to *Op*. When a message is created in this manner, it is put in the work pool. Messages requesting the creation of new objects are distributed (and periodically balanced) across the available processors. The messages in the pool are serviced until no more are available. At this point, the program terminates.

A program comprises a set of object definitions[3]. Each object has a set of entry points to operations in the object. Each operation is defined by a sequential block of C-code. Every message has an entry point associated with it. Instances of object definitions may be created dynamically at run time by creating a message using the *CreateObject* library call. Upon creation, these messages will invoke the operation corresponding to the entry point specified in the message. Messages may also be sent to existing objects at one of the entry points using the *SendMsg* library call. No interrupts or blocking (e.g. for synchronous receives) are possible within an operation.

One of the user-defined objects must be the *main* object which contains some special entry points. One is an initialization entry point: execution always begins by executing the operation associated with this entry point. A second (optional) entry point is *quiescence*. When no messages remain to be serviced, quiescence is detected and the *quiescence* entry point in the *main* object is invoked. If this entry point is not defined in the program, the program terminates. Otherwise, the user may take the necessary action at this point by defining this entry point in the *main* object (for example, report results).

Objects permit the declaration of persistent data that is visible only when executing any code within the object. By definition entry points correspond to operation that are visible outside the object. Objects also permit the declaration of functions private to the object. However, data manager functions are visible to other objects and are typically used as access procedures to data managed by the data manager object.

For reasons of portability, the programmer is expected to provide routines to *pack* and *unpack* messages exchanged between objects. A pair of such routines must be provided for each message type in the program for which packing/unpacking is necessary. These routines will be called from the ProperCAD environment on nonshared memory machines when it is necessary for a message to cross processor boundaries. They are never called on shared memory machines. The algorithms therefore run efficiently on shared and distributed memory machines as well.

The application is expected to create a large number of messages representing parallel work. Typically, this is done by decomposing the problem hierarchically into smaller and smaller subproblems which can be evaluated in parallel, until a threshold is reached. This threshold is user-defined, and is used to indicate that subproblems smaller than the threshold are to be evaluated sequentially. The threshold also determines the parallel-sequential interface. Ideally, the threshold determines the point at which it is cheaper to solve a subproblem sequentially in preference to decomposing it further into parallel subcomponents. Determining this threshold accurately is not necessary as long as sufficiently large number of messages are created each of which represents a reasonable amount of work (e.g. > 50 ms). The more the messages created, the better the load balancing. The problem decomposition is thus independent of the number of processors available.

The ProperCAD environment is currently available on variety of shared memory and nonshared memory machines including the Encore Multimax, the Sequent Symmetry, the Alliant FX/8, the Intel iPSC/2 and i860 hypercubes, the NCUBE 2 hypercube, and a network of Sun workstations. It is currently being written for the BBN TC2000 Butterfly multiprocessor. Four portable implementations of the environment have been developed so far, one for shared memory machines, one for nonshared memory machines, one for NUMA type machines, and one for a network of workstations. Every time a new parallel machine is available, the environment can be written for the new machine with relatively little effort[4].

3 CAD Applications

We have successfully implemented three CAD applications on the ProperCAD environment: *ProperEXT*: VLSI circuit extraction [12], *ProperTEST*: test pattern generation for sequential circuits [13] and *ProperSYN*: combinational logic synthesis [4]. For reasons of space, we do not discuss the implementation here.

[3] These objects are similar to *actors* [1], a type of concurrent object.

[4] This is true unless the architecture of the new machine is radically different to existing architectures. In this case, a new implementation of the environment best suited to the architecture will be developed.

Table I: Performance of the ProperEXT circuit extractor on benchmark circuits on several parallel machines. Note that the extractor has been executed unchanged on all these machines.

Circuit	**Sun** 1PE	**Sun Network** 4 PEs	**Multimax** 8 PEs	**Symmetry** 8 PEs	**NCUBE 2** 64 PEs
Prog. Logic Array	22.6 s	6.8 s	10.4 s	18.2 s	4.4 s
Hypercube Router	73.3 s	34.5 s	29.4 s	51.7 s	34.7 s
Multiplier Array	82.2 s	41.2 s	64.2 s	96.6 s	—
Static Ram	122.8 s	66.2 s	55.0 s	96.2 s	28.6 s
Placement Coprocessor	—	314.8 s	124.6 s	233.2 s	48.9 s
Associative Cache	—	—	127.6 s	191.4 s	56.4 s

Table II: Performance of the ProperTEST test generator on ISCAS 89 benchmark circuits on several parallel machines. The data in parenthesis indicates the speedups obtained with respect to 1-processor performance on the respective machines. No degradation in quality of fault coverage was observed (see [13]).

Circuit	**Sun** 1 PE	**Sun Network** 8 PEs	**Multimax** 8 PEs	**Symmetry** 16 PEs	**Intel i860** 8 PEs
s386	316.2 s	40.4 s	145.3 s	149.0 s	31.9 s
s713	48.0 s	8.3 s	18.8 s	10.4 s	9.6 s
s1196	22.8 s	5.8 s	20.8 s	15.6 s	4.5 s
s1238	36.8 s	8.9 s	25.4 s	22.5 s	6.5 s
s5378	1653.8 s	348.3 s	840.5 s	379.9* s	1024.4 s

* - data for 12 PEs on the Sequent Symmetry

The parameters used to run s5378 were different on different machines (depending on the available memory). The data for s5378 cannot be used to compare the performance of the different machines.

Table III: Preliminary performance of ProperSYN on MCNC benchmark circuits on several parallel machines.

Circuit	**Sun** 1 PE	**Sun Network** 4 PEs	**Multimax** 8 PEs	**Symmetry** 8 PEs	**Intel i860** 8 PEs
bw	352.5 s	80.4 s	67.8 s	189.3 s	32.2 s
f51m	137.9 s	30.8 s	48.2 s	112.6 s	13.2 s
misex1	30.7 s	12.9 s	7.7 s	18.1 s	5.3 s
misex2	96.5 s	27.9 s	29.4 s	65.7 s	10.0 s
rd73	547.2 s	172.4 s	83.4 s	117.8 s	21.6 s
sao2	497.2 s	107.7 s	107.6 s	225.0 s	30.0 s

In Tables I-III we provide sample performance of the three application programs on the Sequent Symmetry, the Encore Multimax, the Intel i860 hypercube, the NCUBE 2 and a network of Sun workstations. This data is provided primarily as proof of concept. ProperEXT was run on real circuits developed as part of graduate level course projects at the University of Illinois. In Table II, we provide the performance of ProperTEST on sequential ISCAS 89 benchmark circuits. In Table III, we provide preliminary performance of ProperSYN on the MCNC benchmark circuits. (ProperSYN is based on transduction; all optimizations have not yet been implemented.)

4 Summary and Future Work

We have described a new environment for the portable object-oriented parallel execution of CAD algorithms. The main objectives of this research are to make porting of parallel software feasible and practical, and exploit the current and future advances in sequential CAD algorithms. We have briefly demonstrated the feasibility of this approach on three examples: VLSI circuit extraction, test generation and logic synthesis.

We believe that this multilevel separation of a machine specific library, a parallel algorithm and a sequential algorithm with well-defined interfaces between them is the most efficient way to develop parallel CAD algorithms. This permits the experts in each of these different areas to concentrate on their fortes. An environment such a ProperCAD is best written by an expert in parallel programming with intimate knowledge about the target machines. The parallel algorithms can then be developed with the constraint that the algorithms are expressed using the ProperCAD environment. Finally, experts in the area of circuit extraction, test generation, logic synthesis etc. should be designated the responsibility of developing efficient sequential algorithms for their respective problems. We constrain them to express their algorithms in a modular fashion: a desirable requirement for program design and maintenance in any case. The ProperCAD environment serves to bridge the effort in these various different areas of specialization.

We are testing this approach further with several other applications in the CAD domain. We are currently developing algorithms for parallel cell placement, parallel global routing and parallel behavioral simulation. Work is also under way to improve the ProperCAD environment. Currently, the programmer must provide routines for packing and unpacking messages exchanged between objects. We are exploring techniques for automating this process. Other optimizations to the environment are also being implemented. We also expect to port the environment to other parallel machines like the Intel Paragon and the Thinking Machines CM-5 as and when they become available to us.

References

[1] Agha, G.A. *Actors: A Model of Concurrent Computation in Distributed Systems.* MIT press, 1986.

[2] Banerjee P., Jones M.H., Sargent J. Parallel Simulated Annealing Algorithms for Standard Cell Placement on Hypercube Multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 1:91–106, January 1990.

[3] Belkhale, K.P, Banerjee, P. PACE2: An Improved Parallel VLSI Extractor with Parameter Extraction. In *Proceedings of the International Conference on Computer Aided Design*, pages 526–530, November 1989.

[4] De, K., Ramkumar, B., Banerjee P. ProperSYN: A Portable Parallel Algorithm for Logic Synthesis. Technical report, Center for Reliable and High-Performance Computing, University of Illinois at Urbana-Champaign, 1992.

[5] Fenton, W., Ramkumar, B., Saletore, V.A., Sinha A.B., Kalé, L.V. Supporting Machine Independent Programming on Diverse Parallel Architectures. In *International Conference on Parallel Processing*, August 1991.

[6] Jayaraman, R., Rutenbar, R.A. Floorplanning by Annealing on a Hypercube Multiprocessor. In *Proceedings of the International Conference on Computer Aided Design*, pages 346–349, November 1987.

[7] Kalé, L.V. The Chare Kernel Parallel Programming System. In *International Conference on Parallel Processing*, August 1990.

[8] Kravitz, S.A., Rutenbar, R.A. Multiprocessor-Based Placement by Simulated Annealing. In *Proceedings of the 23rd Design Automation Conference*, pages 567–573, June 1986.

[9] Levitin S. MACE: A Multiprocessor Approach to Circuit Extraction. Master's thesis, MIT, Cambridge, Mass., June 1986.

[10] Niermann, T.M., Patel, J.H. HITEC: A Test Generation Package for Sequential Circuits. In *Proceedings of the European Conference on Design Automation*, pages 214–218, February 1991.

[11] Patil, S., Banerjee P., Patel, J.H. Parallel Test Generation for Sequential Circuits on General Purpose Multiprocessors. In *Proceedings of the 28th Design Automation Conference*, June 1991.

[12] Ramkumar, B., Banerjee, P. A Portable Parallel Algorithm for VLSI Circuit Extraction. Technical report, Center for Reliable and High-Performance Computing, University of Illinois at Urbana-Champaign, 1992.

[13] Ramkumar, B., Banerjee P. Portable Parallel Test Generation for Sequential Circuits. Technical report, Center for Reliable and High-Performance Computing, University of Illinois at Urbana-Champaign, 1992.

[14] Ravikumar, C.P., Sastry S. Parallel Placement on Hypercube Architectures. In *International Conference on Parallel Processing*, pages III: 97–100, August 1989.

[15] Rose, J.S., Snelgrove W.M., Vranesic, Z.G. Parallel Cell Placement Algorithms with Quality Equivalent to Simulated Annealing. *IEEE Transactions on Computer-Aided Design*, 7, no. 3:387–396, March 1988.

[16] Sechen, C., Sangiovanni-Vincentelli A.L. The TimberWolf Placement and Routing Package. *IEEE Journal of Solid-State Circuits*, 23/2:410, 1988.

[17] Suaris P., Kedem, G. A Quadrisection-Based Combined Place and Route Scheme for Standard Cells. *IEEE Transactions on Circuits and Systems*, 8:234–244, March 1989.

[18] Tonkin, B.A. Circuit Extraction on a Message Passing Multiprocessor. In *Design Automation Conference*, pages 260–265, June 1990.