Bipartitioning for Hybrid FPGA-Software Simulation

Ashutosh Singla Thomas M. Conte
North Carolina State University
Raleigh, North Carolina 27695-7911
P.O.Box 7911
conte@eos.ncsu.edu

Abstract

Simulation is an important step in the design cycle of VLSI systems. The increasing size and complexity of modern systems require simulation techniques optimized for time. Researchers are resorting to parallel simulation to reduce simulation time. Logic partitioning plays an important role in parallel simulation. Two factors, concurrency amongst the partitions and communication between them, determine the effectiveness of partitioning. The concurrency achieved and the communication overhead resulting from the intersecting signals can directly affect the speed-up achieved in the simulation. Hybrid FPGA-software simulation offers an alternative for increasing the speed of simulation. In addition to above factors, size and cost of FPGA also determine the partitioning technique for FPGA based emulation. This paper addresses the issues involved in hybrid FPGA-software simulation and presents a new partitioning scheme. With our approach, communication between partitions reduces to at least 50% of that observed in the best of the other algorithms. Also for most of the benchmarks, only 25% of the circuit elements are in the FPGA partition. Presimulation is employed as an effective tool to achieve this aim.

1 Introduction

Simulation-based design verification of VLSI systems significantly reduces design cost. Unfortunately, simulation is computationally expensive. The increasing size and complexity of modern systems require fast and efficient simulation techniques.

Parallel and distributed simulation can achieve a speed-up of two to three orders of magnitude. Several software techniques have been proposed for parallel and distributed simulation [1], [2]. Also FPGAs are becoming popular in the emulation of large VLSI systems to reduce simulation time. But their size and cost present a major limitation.

Hybrid FPGA-software simulation offers an alternative for reducing simulation time which overcomes the limitations posed by FPGA based emulation. In VLSI systems, only a fraction of the circuit is alive most of the time. If these highly active elements can be detected, then FPGAs can be used to implement them. This overcomes the limitation posed by the size of FPGA, as only a small part of circuit is highly active. The remaining larger part can be placed in software. The problem can be restated as minimization of communication between these two partitions. This paper presents a new partitioning scheme useful for hybrid FPGA-software simulation. The following section gives an overview of related work. The partitioning scheme is discussed in Section 3. Section 4 presents the experiments and the results. The last section concludes the paper with a discussion of future research directions.

2 Previous Work

Levendel, et al. [3] presented a partitioning algorithm based on strings. Strings are defined as a set of connected gates with at most one fan-out and one fan-in. This algorithm ensures that there is at least one fan-out of a gate in the same partition. The scheme aims at maximizing the concurrency but without any consideration of the communication between partitions.

Fan-in cone and Fan-out cone partitioning by Smith, et al. [4] tends towards reducing communication. The Fan-in cone of a gate A is the set of all gates which affect the output of gate A. The Fan-out cone of A is set of all gates affected by the output of A. In fan-in cone partitioning, first the fan-in cones of each gate are found. Then, the gates driven by primary inputs are assigned evenly to the processors.

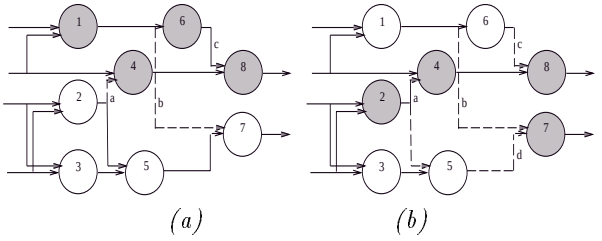

Figure 1: Fan-in Cone Partitioning

A gate is randomly selected for assignment to some partition. The fan-in cone of this randomly selected gate is compared with the union of the fan-in cone of gates already placed on each partition. The partition which has the largest set in common is selected for the gate. This process is repeated until there are no gates remaining. The impact of the scheme is limited when load balancing is considered. As a result of load balancing, after a processor is full, it is not considered for further partitioning. This increases the communication as shown later on. But load balancing is essential as the algorithm tends to put most of the gates in one partition. Also, it is possible to find partitions that have less communication as compared to that of cone partitioning. The reason is that there can be more than one set of partitions because of the randomness of the algorithm. Figure 1(a) illustrates fan-in cone partitioning resulting in *a, b* as communicating signals. Figure 1(b) illustrates another possible fan-in cone partitioning resulting in *a, b, c, d* as communicating signals. Clearly the partitioning in Figure 1(a) shows less communication as compared to that in Figure 1(b).

Fiduccia and Mattheyses's [5] partitioning scheme reduces the number of arcs(cutsize) that cross the partitions. An initial partition is selected and is iteratively modified so as to reduce the cutsize. Cutsize is the number of arcs(signals) that cross the partitions. The algorithm moves a node(gate) at a time. The node which causes maximum reduction in the cutsize is moved. The problem with the algorithm is that minimizing the cutsize does not ensure minimization of activity between the partitions. The reason is that actual communication between two partitions is the sum of activities on all the signals in the cutsize. There can be a cutsize which has more signals but with a total activity less than that found using the above algorithm. The partitioning shown in Figure 2(a) can be a result of the above algorithm. The numbers along the circles or arrows represent weights of corresponding nodes and arcs. The communicating signal is *b* with an activity of 15. For the same circuit another partitioning is shown in Figure 2(b), resulting in *d and e* as communicating signals. The total activity on these signals is 10, which is less than that on *b*.

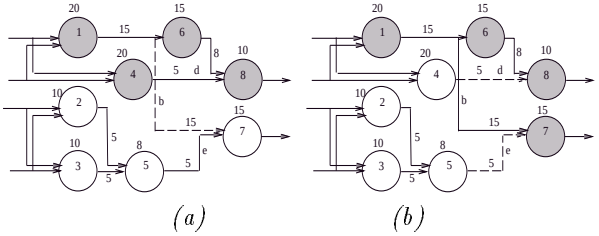

Figure 2: Fiduccia & Mattheyses Partitioning

3 A Hybrid FPGA-Software Partitioning Scheme

The previous schemes suffer from the fact that the actual communication differs from the number of signals crossing a partition, and the actual circuit activity differs from the number of nodes in a partition. Presimulation can be an effective tool to determine actual circuit behavior. Chamberlain and Henderson [6] show that, for random input vectors, the first 10% of a simulation is an excellent predictor of the evaluation frequency for the rest of the run.

Our scheme uses presimulation in bipartitioning a circuit for hybrid FPGA-software simulation. A VHDL model of the circuit is used. The FPGA partition is constructed from the circuit elements which are executed the most frequently. The remainder of the circuit is placed in a partition that is simulated by software. Circuit elements in the FPGA partition wait for input signals coming from the software partition. This communication needs to be reduced, otherwise synchronization overhead can impact performance. The aim of our approach is to bipartition the circuit so that the communication between the two is minimized and one partition consists of highly active elements.

3.1 Algorithm

The bipartitioning algorithm is based on a weighted graph. Each gate is represented as a node in the graph, and the fan-out or fan-in of each gate are shown as arcs connecting the nodes. Weight of each node is obtained by presimulation of the circuit and represents the number of events for that node. Two thresholds, hard-threshold and soft-threshold, are set in accordance with the maximum weight of a node of the circuit. *Hard-threshold* is the minimum weight of a node

below which the node is not considered for the FPGA partition. *Soft-threshold* is the minimum weight of a directly connected node (via a particular node's fan-in or fan-out) below which the node in question is placed in the software partition (see example below). Following steps constitute the algorithm after the graph has been constructed:

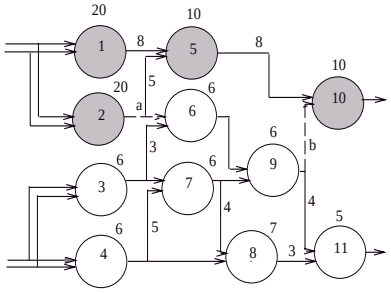

Figure 3: BIFSS Partitioning

(1) Start from any node of the graph (we shall call it the *head* node). (2) Go to step 9 if the weight of the *head* is below *hard-threshold*. (3) Place *head* in the FPGA partition. Find fan-out nodes of the *head*. Select any fan-out node (we shall call this the *second-head*). (4) If the weight of *second-head* is below *soft-threshold*, then place it in the software partition and add the interconnecting signal to the cutsize. Assign *second-head* to an adjacent node. If there is no adjacent node, then move back to the last node which has an adjacent node and assign that to *second-head* and repeat step 4. (5) If the weight of *second-head* is above *soft-threshold*, then place it in the FPGA partition. Find the fan-out nodes of *second-head* and assign one of these to *second-head*. If there are no fan-out nodes, then move back to the last node which has an adjacent node and assign that to second-head. Go to step 4. Continue until there are no adjacent nodes remaining. (6) Find the fan-in nodes of the *head*. Select any fan-in node. (We shall call it *second-head*.) (7) Perform step 4. (8) If the weight of second-head is above *soft-threshold*, then place it in the FPGA partition. Find the fan-in nodes of *second-head* and assign one of these to *second-head*. If there are no fan-in nodes, then move back to the last node which has an adjacent node and assign that to *second-head*. Go to step 7. Continue until there are no adjacent nodes remaining. (9) If any node remains unassigned, assign that to *head* and go to step 2.

The algorithm is explained using the example in Figure 3. *Hard-threshold* is 10 and *soft-threshold* is 6. The numbers along the circles or the arcs represent weights of the corresponding nodes or arcs. Following the above algorithm, nodes 1, 2, 5, 10 belong to FPGA partition and the remaining belong to software partition. Cutsize consists of arcs *a* and *b*. Note that node 8 is in software partition, although it is above *soft-threshold* . It is because this node is not the fan-out or the fan-in of any node which is above *hard-threshold*.

Table 1: Benchmarks characteristics

	Benchmarks				
	s9234	s13207	s15850	s35932	s38584
Gates	5597	7951	9772	16065	19253
Flip-flops	211	638	534	1728	1426
Inputs	36	62	77	35	38
Outputs	39	152	150	320	304

Table 2: Number of gates in FPGA partition

Partitioning algorithm	Benchmarks				
	s9234	s13207	s15850	s35932	s38584
Fan-in	2905	4295	5154	8897	10340
Fan-out	2905	4295	5153	8897	10340
Random	2904	4294	5153	8896	10339
Bifss	1630	254	1769	11190	5104

4 Results

Analysis was performed on five ISCAS89 test circuits. A summary of the test circuits is given in Table 1. All experiments were done on HP-9000/735 workstations. Circuits were analyzed and simulated using the *Vanilla CAD*^{TM} VHDL analyzer and simulator. The partitioning scheme was applied on presimulation results from 2000ns of simulation time. Random input vectors using a uniform distribution were generated for presimulation. *Hard-threshold* and *soft-threshold* were set to 50% and 30% of maximum weight of a node. Table 2 and Table 3 show the number of gates in the FPGA partition and the cutsize, respectively. Results are shown for the fan-in cone partitioning (**fan-in**), fan-out cone partitioning (**fan-out**), random partitioning (**random**) and the bipartitioning (**bifss**) schemes. Two sets of results were obtained each from a run of 10,000ns of simulation time. For each set, a Poisson distributed random number generator with various seeds was used to generate input vectors. Tables 4–5 show activity in FPGA partition for 10,000 ns simulation time. Tables 6–7 provide inter-partition communication.

The results confirm the assumptions described earlier. Table 2 shows that highly active elements placed in the FPGA partition represent a small percentage of the total number of circuit elements. Furthermore,

Table 3: Cutsize

Partitioning algorithm	Benchmarks				
	s9234	s13207	s15850	s35932	s38584
Fan-in	2732	4375	2648	10642	11408
Fan-out	3189	3216	5038	10740	11119
Random	3240	4622	5567	10704	11720
Bifss	885	80	803	7280	5147

Table 4: Activity in FPGA partition: Run 1

Partitioning algorithm	Benchmarks				
	s9234	s13207	s15850	s35932	s38584
Fan-in	114370	172333	232913	844030	629577
Fan-out	117296	161597	227752	875423	619183
Random	68125	171218	233528	861957	616551
Bifss	98601	17137	155788	928275	477104

Tables 4-5 show that this small percentage of elements has high activity. For the s15850 benchmark, 17% of circuit elements are in FPGA partition, which shows 61% of the overall activity. Only benchmark s35932 is an exception with 62.88% of the total number of circuit elements placed in the FPGA partition. The reason being that the components are closely bound in this benchmark. In the worst case, the cutsize of the bipartitioning algorithm is 31.59% less than that from other algorithms. Tables 6–7 further confirm that the connectivity resulting from the bipartitioning scheme is at least 48% less than that from the best of the other algorithms. The reason for this, as described earlier, is that the cutsize or number of gates in a partition is not proportional to the connectivity or activity in the partition.

5 Conclusions

The hybrid FPGA-software partitioning technique minimizes the communication between partitions while simultaneously placing highly active elements in one partition. Empirical results disprove the assumptions used by earlier partitioning schemes and demonstrate the value of presimulation data for partitioning.

The results are based on random input vectors. It is unknown how the results will differ when input vectors, more typical of VLSI systems, are used. Also, the technique has yet to be applied to larger circuits.

Future work involves the study of synchronization overhead for the hybrid FPGA-software scheme using an Altera RIPP-10 board and FLEXlogic FPGAs.

Table 5: Activity in FPGA partition: Run 2

Partitioning algorithm	Benchmarks				
	s9234	s13207	s15850	s35932	s38584
Fan-in	110251	172266	221481	862231	572324
Fan-out	113263	161475	216995	894720	561679
Random	65801	171175	222327	881163	558961
Bifss	95424	17400	149154	950261	440545

Table 6: Connectivity: Run 1

Partitioning algorithm	Benchmarks				
	s9234	s13207	s15850	s35932	s38584
Fan-in	35775	49925	40339	301125	179022
Fan-out	44150	35934	67031	303232	176216
Random	45151	55784	81491	302063	186559
Bifss	10086	1437	9147	98293	89459

Acknowledgements

We want to thank Dr. Gary Beihl of AT&T GIS for providing the *Vanilla CAD*^{TM} VHDL simulator. Thanks also to Nick Tredennick of Altera/Tredennick, Inc. for support of our reconfigurable logic research projects.

This research is supported by the National Science Foundation under grant MIP-9410377 and by an equipment donation from Altera.

Table 7: Connectivity: Run 2

Partitioning algorithm	Benchmarks				
	s9234	s13207	s15850	s35932	s38584
Fan-in	34197	50286	37627	307609	157547
Fan-out	42245	35910	62911	309980	155128
Random	43164	55916	77406	308852	164423
Bifss	9490	1496	8664	92301	77636

References

[1] Fujimoto R. M., "Parallel discrete event simulation", in C. ACM, 33(10):30-53, 1990.

[2] Misra J., "Distributed discrete event simulation", in ACM Computing Surveys, 18(1):39-65, 1986.

[3] Levendel Y. H., Menon P. R., and Patel S. H., "Special-purpose computer for logic simulation using distributed processing", in Bell Syst. Tech. J., 61, 10, 2873-2909, 1982.

[4] Smith S. P., Underwood B., and Mercer M. R., "An analysis of several approaches to circuit partitioning for parallel logic simulation", in Proceedings of the 1987 International Conference on Computer Design, IEEE, New York, 664-667, 1987.

[5] Fiduccia C. M., and Mattheyses R. M., "A linear time heuristic for improving network partitions", in Proceedings of the 19th ACM/IEEE DAC, ACM, New York, 175-181, 1982.

[6] Chamberlain R. D., and Henderson C., "Evaluating the use of presimulation in VLSI circuit partitioning", in Proceedings of the 8th Workshop on Parallel and Distributed Simulation, SCS, 139-146, 1994.