

Zamlog: A Parallel Algorithm for Fault Simulation Based on *Zambezi**

Minesh B. Amin[†], Bapiraju Vinnakota

[†] Guru Technologies, Inc., 3065 Democracy Way, Santa Clara, CA
Dept. of Elec. Engg., University of Minnesota, Minneapolis, MN 55455

Abstract

We present a new multiprocessor sequential circuit fault simulator, Zamlog, based on a novel uniprocessor simulator, Zambezi. Both the fault and test sets are partitioned for multiprocessor simulation. The parallelization technique, designed to preserve the efficiency of Zambezi, is simple to implement and has low communication requirements. Experimental results indicate that Zamlog can obtain speedups of up to 95. The speedups obtained and the scalability are between 3 and 10 times better than any reported in the literature. Furthermore, the speed-ups obtained are with respect to a uniprocessor algorithm which is superior, by an average of 40%, to those used to gauge the speed-ups of previous parallel systems.

1 Introduction

Even with efficient fault simulators [1]-[3], simulating large circuits is expensive. Hence, there has been an interest in parallelizing simulation [4]-[12]. We present a new data parallel fault simulator for sequential circuits, *Zamlog*, based on a high performance uniprocessor algorithm. The uniprocessor algorithm, *Zambezi*, combines parallel fault [1] and parallel pattern [2] algorithms. Within one W -bit computer word, *Zambezi* simulates both W patterns and multiple faults simultaneously. Previous algorithms simulated either multiple patterns or multiple faults simultaneously, not both. *Zambezi* was developed by enhancing the control flow of parallel pattern algorithms by using fault propagation techniques, multiple insertion propagation, and multiple fault insertion. *Zambezi* has two main components, logic simulation and fault simulation. In general, fault simulation occupies a much large fraction of uniprocessor execution time than does logic simulation. Hence, the data partitioning in *Zamlog* parallelizes fault simulation to a greater degree than logic simulation.

2 Multiprocessor Simulation

In data parallel simulation, each processor simulates a subset of the simulation data with the entire circuit and algorithm. In fault simulation, the vectors, the faults, or both may be partitioned. The efficiency of the uniprocessor algorithm may deteriorate when the data set is partitioned. For example, a good fault ordering, such as a depth-first ordering, may be destroyed when the fault set is partitioned. Our data partitioning techniques are designed to minimize their impact on the efficiency of *Zambezi*.

2.1 Vector Set Partitioning

We assume that the vectors in the input set can be decomposed into individual sequences. One may partition the input set into groups of sequences, *and* also partition individual sequences. If the latter is done, state information for both good and faulty circuits, will have to be passed between processors during simulation. This increases communication requirements. Hence, sequences are preserved by our partitioning process. The set of input vectors is partitioned into disjoint subsets, of approximately equal size. The degree of partitioning is limited by the number of sequences. As the degree of partitioning increases, maintaining equality in subset sizes will be more difficult, reducing parallelization efficiency.

2.2 Fault Set Partitioning

A fault set partitioning technique has to divide the original fault set into K disjoint subsets such that approximately the same effort is required to simulate each of them. The partitioning technique will depend on the uniprocessor simulation algorithm. *Zamlog* uses an enhanced version of the *LOG* partitioning technique [12]. *LOG* partitioning is based on an analysis of the distribution of workload in fault simulation, and was originally developed for parallel pattern single fault simulation. *Zambezi* simulates both multiple faults and patterns simultaneously. Usually, the faults simulated simultaneously belong to the same gate. *LOG* partitioning distributes the faults on each gate to multiple processors. If applied directly, *LOG*

*This research was supported by the National Science Foundation through grant MIP-9502240

partitioning would adversely impact the efficiency of *Zamlog*. Let the term *faulty gate* refer to a gate with at least one faulty incoming edge, or faulty output. The *enhanced LOG partitioning* scheme is defined as follows: (1) The faulty gates at each level are distributed uniformly; (2) Within each level, all faulty gates within each output cone are distributed uniformly. Therefore, enhanced LOG partitioning distributes faulty gates uniformly.

2.3 Parallel Simulation

Zamlog uses a set of linear arrays, which can be mapped on to many topologies, for parallel simulation.

One Dimensional Parallelism: When the number of processors available is small, a single linear array is formed. Let the number of processors be $ZWay$. The fault set F is partitioned into $ZWay$ subsets. Initially, set F_j , $0 \leq j < ZWay$ is mapped to processor P_j . The test set T is also partitioned into $ZWay$ subsets, and subset T_j , $0 \leq j < ZWay$, is mapped to processor P_j , and resides there for the entire simulation. Note that a processor can identify the data sets assigned to it without any communication. Once partitioning is complete, each processor logic simulates its vector set. The logic values computed are used for the entire simulation. Fault simulation begins when logic simulation is complete. First, at each processor, the resident set of vectors is simulated with the subset of faults initially assigned to it. All processors work in parallel. After the fault subset is simulated, detected faults are deleted from it, and the remaining faults are sent to its neighbor, $P_{j+1 \% ZWay}$. At each processor, the above procedure is repeated with the new fault set. The maximum number of such iterations is $ZWay$.

Two Dimensional Parallelism: This is used when the number of available processors is greater than the number of test sequences. The P processors are configured as a set of $KWay$ linear arrays, each with $ZWay$ processors. $ZWay$ is the largest integer for which $KWay = \frac{P}{ZWay}$ is an integer. Simulation is parallelized as follows: (1) Each linear array LA_i is assigned a set of faults F_i , $0 \leq i \leq KWay - 1$ and the entire set of test vectors T . The set of faults is partitioned with enhanced *LOG* partitioning. (2) Within each array, the local fault set and the test set are each partitioned into $ZWay$ subsets. The partitioning and assignment of the test sets in all arrays is identical. Again, a processor can independently identify both the vectors and faults assigned to it. (3) Each linear array independently and in parallel simulates the test set with its fault set in $ZWay$ iterations (as described above). All communication during algorithm execution is within arrays. No global synchronization

is required. Note that the fault set is partitioned to a greater degree than the vector set. Such a solution is natural since on a single processor the actual fault simulation occupies a much larger fraction of execution time than does logic simulation.

2.4 Analysis of Zamlog

Figure 1 summarizes the steps in data partitioning. The total number of processors is $P = KWay \times ZWay$. The j^{th} processor in linear array $i(LA_i)$, $P_{i,j}$ communicates only with processors $P_{i,(j-1) \% ZWay}$ and $P_{i,(j+1) \% ZWay}$. That is, only with its neighbors within LA_i . The parallel run time is determined by the slowest linear array. For a linear array, the run time is determined by the slowest processor within it. When $ZWay = 1$, there is no communication. As $ZWay$ increases, the communication increases proportionally. The only information passed between processors are lists of undetected faults. In our implementation, each fault requires 4 bytes of storage. Thus, the communication costs are linear in the number of faults, and drop as execution progresses. Communication requirements can be changed according to the available bandwidth, by altering $ZWay$.

3 Results

The performance of *Zamlog* was evaluated as follows: (1) Random vectors were used to simulate each of the six largest benchmark circuits. This approach is similar to the one taken by [11]. Run times with deterministically generated vectors are too low for parallelization. (2) The test set consisted of 8 sequences, each of varying length, with a total of 960 vectors. The test set size is comparable to that used in [11]. (3) All the experiments were run on a SGI Challenger. The system consists of 16 150MHz R4400 CPUs with 0.5G-bytes of total memory. The system bus has a bandwidth of 1.2G-bytes/sec. (4) Let the run time with P processors be T_p . The *speedup* is the ratio $\frac{T_1}{T_p}$. The efficiency is the ratio $\frac{T_1}{PT_p}$.

For many configurations, the number of processors is greater than the number available to us physically in the SGI challenger. To demonstrate the higher degree of scalability available with *Zamlog*, we emulate such configurations as follows. Given a $KWay$ by $ZWay$ configuration, the simulation is executed in $KWay$ iterations. A linear array with $ZWay$ processors is used to simulate a single data cluster in each iteration. The slowest iteration determines the execution time for the larger configuration. Thus, communication costs are included in the measured parallel execution times for all configurations with $ZWay > 1$.

For the six largest ISCAS89 benchmark circuits,

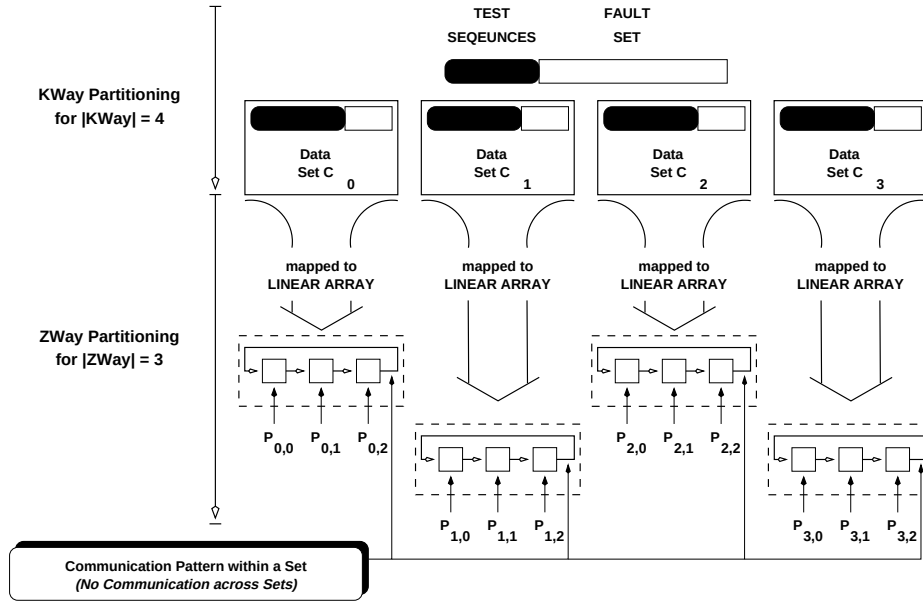


Figure 1: Overview of *Zamlog* partitioning scheme

Circuits	$Zway$	$Kway$				
		1	4	8	16	32
s9234	1	98.00	30.36	18.48	12.10	8.42
	2	37.60	12.43	7.91	5.42	4.06
	4	17.04	5.46	3.73	2.84	2.04
	8	9.75	3.35	3.39	1.83	1.52
s13207	1	175.01	45.78	32.24	22.58	17.39
	2	58.00	20.31	13.66	10.17	8.28
	4	27.64	10.04	6.71	4.93	5.79
	8	15.41	5.77	4.01	3.02	2.59
s15850	1	222.97	76.69	54.89	28.36	20.10
	2	84.18	27.17	17.10	11.75	8.92
	4	43.97	15.53	9.27	6.23	5.22
	8	21.62	7.32	4.36	3.25	2.59
s35932	1	86.01	50.32	46.27	41.25	60.43
	2	46.84	26.40	22.93	21.19	29.88
	4	23.38	13.46	11.52	15.42	10.27
	8	24.50	10.99	8.35	6.95	6.35
s38417	1	1304.06	373.93	210.45	132.82	102.17
	2	560.05	164.12	95.69	62.13	48.74
	4	256.80	78.52	48.76	31.63	25.43
	8	131.51	40.06	28.60	16.99	13.86
s38584	1	847.29	273.10	173.70	127.49	121.71
	2	291.80	108.68	76.28	58.39	66.52
	4	173.09	61.62	54.52	30.66	33.19
	8	96.61	34.46	22.82	23.56	15.75

Table 1: Parallel execution times

Authors	Max. Speedups	Max. Processors	Architecture
This paper	96	256	SGI Challenger
Parkes et al. [11]	27.7	64	Intel Paragon
R. B. Mueller-Thuns et al. [9]	7.4	8	iPSC/Sequent/Encore
S. Tai et al. [8]	8.5	16	iPSC/2
V. Narayanan et al. [6]	42.27	15,000	CM2
A. Bataineh et al. [7]	69.0	8	CRAY-YMP using
N. Ishiura et al. [4]	10.00	1	Fujitsu VP-200

Table 2: Performance comparison

Table 1 shows the execution times for *Zamlog* for $1 \leq Kway \leq 32$ and $1 \leq ZWay \leq 8$. Table 2 compares our results with those reported in the literature. The size of *ZWay* indicates the length of the linear array. As *ZWay* increases, the corresponding performance for a fixed *KWay* increases. Using up to 256 processors, speedups range between 3-13 for s35932 and between 3-96 for s38417. Efficiency numbers can be directly computed from the data presented. For the smaller configurations, efficiency is quite high. Recall that our partitioning scheme preserves test sequences. Since the sequences are of varying lengths, the number of vectors within each subset may be unequal. Hence, as *ZWay* increases the imbalance in subset sizes reduces efficiency.

4 Conclusion

We have presented a new parallel simulator, *Zamlog*, which uses a collection of linear arrays to simulate large circuits. *Zamlog* partitions the set of faults to be simulated both across and within arrays. The set of test vectors is only partitioned within the component arrays. The parallel simulation can be set up easily since each processor can independently identify the simulation data to be used by it. During algorithm execution each processor only communicates with its neighbors in its linear array. No communication between arrays is required. Given the simplicity of the configuration, and the limited communication, it can be realized on a variety of common parallel architectures. An implementation on an SGI system achieved speed-ups in the range of 13-95. Further, *Zamlog* uses a fault simulator *Zambezi* which has been shown to be superior to uniprocessor algorithms used by previous parallel simulators. To the best of our knowledge, these results represent an order of magnitude improvement compared to results reported in literature.

References

- [1] T. M. Nierman, W. T. Cheng, and J. H. Patel, "Proofs: A fast memory efficient sequential circuit fault simulator," *IEEE T-CAD*, 1992.
- [2] N. Gouders and R. Kaibel, "Paris: A parallel pattern fault simulator for synchronous sequential circuits," in *Proc. ICCAD*, 1991.
- [3] M. B. Amin and B. Vinnakota, "Zambezi: A parallel pattern parallel fault sequential circuit fault simulator," in *Proc. VTS*, 1996.
- [4] N. Ishiura, M. Ito, and S. Yajima, "Dynamic two-dimensional parallel simulation technique for high-speed fault simulation on a vector processor," *IEEE T-CAD*, pp. 868–875, August 1990.
- [5] S. Patil and P. Banerjee, "Performance trade-offs in a parallel test generation/fault simulation environment," *IEEE T-CAD*, pp. 1542–1558, December 1991.
- [6] V. Narayanan and V. Pitchumani, "Fault simulation on massively parallel simd machines algorithms, implementations and results," *JETTA*, no. 3, pp. 79–92, 1992.
- [7] A. Bataineh, F. Ozguner, and I. Szauter, "Parallel logic and fault simulation algorithms for shared memory multiprocessors," in *Proc. IC-CAD*, pp. 369–372, 1992.
- [8] S. E. Tai and D. Bhattacharya, "Pipelined fault simulation on parallel machines using the circuit flow graph," in *Proc. ICCD*, 1993.
- [9] R. B. Mueller-Thuns, D. G. Saab, R. F. Damiano, and J. A. Abraham, "VLSI logic and fault simulation on general-purpose parallel computers," *IEEE T-CAD*, pp. 446–460, March, 1993.
- [10] M. B. Amin and B. Vinnakota, "Data parallel fault simulation," in *Proc. ICCD*, 1995.
- [11] S. Parkes, P. Banerjee, and J. Patel, "A parallel algorithm for fault simulation based on Proofs," in *Proc. ICCD*, 1995.
- [12] M. B. Amin and B. Vinnakota, "Data parallel sequential circuit fault simulation," in *Proc. IS-CAS'96*, 1996.