Modeling Large Scale Circuits Using Massively Parallel Discrete-Event Simulation

Elsa Gonsiorowski
Department of Computer Science
Rensselaer Polytechnic Institute
110 8th Street, Troy, NY
gonsie@rpi.edu

Christopher Carothers
Department of Computer Science
Rensselaer Polytechnic Institute
110 8th Street, Troy, NY
chrisc@cs.rpi.edu

Carl Tropper
School of Computer Science
McGill University
Montreal, Canada
carl@cs.mcgill.ca

***Abstract*—**
As computing systems grow to exascale levels of performance, the smallest elements of a single processor can greatly affect the entire computer system (e.g. its power consumption). As future generations of processors are developed, simulation at the gate level is necessary to ensure that the necessary target performance benchmarks are met prior to fabrication. The most common simulation tools available today utilize either a single node or small clusters and as such create a bottleneck in the development process. This paper focuses on the massively parallel simulation of logic gate circuit models using supercomputer systems. The focus of this performance study leverages the OpenSPARC T2 processor design using Rensselaer's Optimistic Simulation System (ROSS). We conduct simulations of the crossbar component on both a 24-core SMP machine and an IBM Blue Gene/L. Using a single SMP core as the baseline, our performance experiments on 1024 cores of the Blue Gene/L demonstrate more than 131-times faster execution. Our results capitalize on the balanced compute and network power of the Blue Gene/L system.

I. Introduction

As supercomputer systems approach exascale, the core count will exceed 1024 and number of transistors used in these designs will number in the 10's of billions. However, at the same time there is an increased need to understand the performance dynamics of these systems at the lowest levels in order to provide highly accurate performance and power utilization measures. For example, if an 25 MWatt supercomputer's power estimation is off by 5% that equates to 1.25 MWatts or $1.25 million per year of operation. Because of the size and operating costs of these systems, accurate estimates are needed.

Digital logic simulation is a well known tool which can be used to address this problem. However, to date, gate-level simulations has been too computationally intensive to be an effective solution. As a first step, this paper focuses on the parallel simulation of the OpenSPARC T2 crossbar (CCX) component making use of current supercomputer systems. Using ROSS, we are able to investigate both conservative [1] and optimistic [2] simulation. We include both strong scaling and weak scaling experiments, along with attempts to increase the efficiency of optimistic simulation. For the weak scaling experiments, a single crossbar is replicated many times and nominally connected across a few data lines. These experiments are conducted on two machines:

- A 24-core symmetric multiprocessing machine with 2.667 GHz cores (baseline).
- An IBM Blue Gene/L with 700 MHz cores interconnected with fast communication networks [3].

We demonstrate the efficacy of using massively parallel systems to improve the performance from almost 900,000 events per second on a single SMP machine core to over 116,000,000 state-transition events per second on 1024 Blue Gene/L cores. This represents an improvement in performance of nearly 131 times. Additionally, we find that because of the low variance in timestamp increments within the gate model, conservative synchronization outperforms optimistic processing with reverse computation.

II. Related Work

Much work has been done in digital logic simulation, employing many different simulation techniques. Hardware based simulation [4] can be efficient, but is not as flexible as software based implementations. Additionally, computing power is ever increasing and the best software simulators will be able to take advantage of future progress.

With the rise of GPU programming, [5] and [6] have investigated possible simulation speedups in this area. While their results have been significant, GPUs lack the memory capacity for extremely large circuit models.

The most comparable digital logic simulations research has been preformed by Tropper et al [7]–[10] with the latest results in [11]. Both [7] and [8] focus on Clustered Time Warp simulations. The optimistic XTW simulator [10] was an outgrowth of this work. In this paper we utilize the Verilog parser [9] built in conjunction with XTW. The continuation of this work focuses on speedup gains obtained through dynamic load balancing [11]–[13].

III. The ROSS Framework

ROSS is a framework designed for parallel discrete-event simulation and is built upon Jefferson's Time Warp [14]. Each discrete object in the simulation is known as a *Logical Process* (LP). LPs communicate with each other through messages (also known as events). In this specific simulation, the LPs model gate objects. The messages sent between the LPs represent electrical signals between gates.

Unlike many other simulation systems, ROSS has no form of state-saving. While event reconstruction has been used with circuit simulation before [15], reverse computation [16] is most effective for this model due to small message size. To revert the state of a gate, a reverse computation is used. Through reverse messages, the gate object must be able to undo any state changes which happened during forward event processing. Thus, for each forward event course of action, a corresponding reverse computation must be implemented.

The underpinnings of ROSS [16]–[19] are written in ASNI C and designed for efficiency. This includes the reversible random number generator library based upon a Combined linear Congruential Generator. All memory is handled within ROSS itself, and is centered around managing the event objects. The overarching event list (for each processor) is a priority queue, implemented as either a Calendar Queue [20] or a Splay-Tree [21]. This research utilizes only the Splay-Tree implementation.

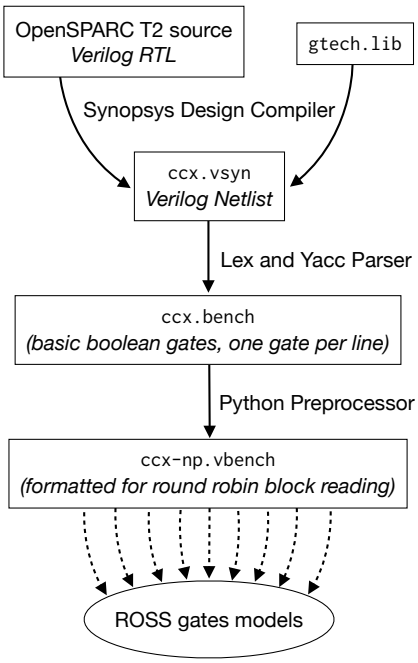

Fig. 1. Process for transforming RTL code to machine readable gate descriptions.

IV. The Data

The OpenSPARC T2 processor design is an excellent example of open-source hardware [22]. It is a 64-bit, eight-core, microprocessor description in VHDL and Verilog. Using the Synopsys Design Compiler and scripts provided by the OpenSPARC code base, we were able to generate gate level descriptions for discrete sections of the processor. In this paper we focus solely on the CPU-cache crossbar, the processor component which links the eight processing units. This module contains over 200,000 gates.

The gate description is transformed and processed in many ways before it can be efficiently used in a simulation model. This process is described in Figure 1.

A. Source

The OpenSPARC T2 design is provided in Verilog Register Transfer Language (RTL). This flexible description is fairly high level, especially from a hardware perspective. It encodes modules at an abstract, logical level. (Data Format 1).

Data Format 1 Verilog RTL

```
module ccx (scan_in, scan_out, ...);
wire [1:0] scan_in_buf;
input [1:0] scan_in;
...
clkgen_ccx_cmp clk_ccx (...);
endmodule
```

B. Gate Level

Using the Synopsys Design Complier, the RTL source can be synthesized into a gate level description. At this level several high level modules can be synthesized into one flat netlist. This file format is still completely valid Verilog code. The module is defined with connection arguments and the netlist of its gates. For this research, we used the generic technology library (GTECH) [23]. (Data Format 2).

Data Format 2 Verilog Netlist

```
module ccx (scan_in, scan_out, ...);
input [1:0] scan_in;
...
GTECH_NOT U61 (.A(n1319), .Z(n94428));
GETCH_BUF U78 (.A(scan_out[1]), .Z(ccx_out[15]));
endmodule
```

C. Basic Boolean Gates

The GTECH standard cell library is provided by Synopsys. This gate library contains over 100 specialized gates, many of which operate at a higher level than the traditional boolean logic level. To facilitate development, each GTECH module is further broken down into its basic gates. Of these basic gates, there are only 8 logical types: repeater (DFF), NOT, AND, NAND, OR, NOR, XOR, and XNOR. These gates are flexible and allow for a maximum of four inputs. Three additional types are available: input, output, and clock, and are kept as distinct types.

The transformation is accomplished by a Verilog parser provided by Li et al [8]. This parser utilizes Lex and Yacc [24].

The resulting file format has one gate description per line. Input and output gates specify the purpose of a named connection. Each gate object is written as an assignment. The

named output wire is the result of a gate function which takes named input wire arguments. At this level most of the original wire names are maintained, however, intermediate wires have been inserted by the parser. (Data Format 3).

Data Format 3 Basic Gates

```
INPUT(scan_in[0])
INPUT(scan_in[1])
OUPUT(scan_out[0])
OUTPUT(scan_oun[1])
...
n1319 = NOT(n94428)
ccx_rstg_out[15] = DFF(scan_out[1])
```

D. Machine Format

The final conversion step is simple processing to allow for efficient machine reading. At this point, almost all of the details of each gate are captured in a single line. However, gate names are still expressed as strings and must be converted to global identification numbers. For maximum compactness, line numbers correspond to the global id and the implicit order must be understood by the ROSS model.

The meaning of each number on a line in Data Format 4 is as follows:

- 0^{th} number: Line number implies global ID
- 1^{st} number: Output count
- 2^{nd} number: Gate type
- $3^{rd} - 6^{th}$ number: Global IDs of any input connections

Data Format 4 Machine Gates

```
0 2
0 3 3
2 5 46762
1 7 126287 126288 126289 126290
1 6 126192 126274
12 4 29309
```

E. Mapping and Parallel I/O

When formatting data files for reading in parallel, one must ensure that individual blocks can be read efficiently by any single processor. Block reading means that each line must contain the same number of bytes, allowing for block size calculation at runtime. Thus, each line is filled with whitespace characters to ensure a consistent line length.

The final step for the data file is to ensure that the order matches the LP mapping at runtime. Due to the fact that the original crossbar component (in Verilog RTL form) had a high level module ordering, the final machine data file may have some overarching, implicit ordering. For small simulations on many processors (as in the strong scaling experiments) we attempt to balance the load across all processors which contain a part of a distinct crossbar. To create this balance, the gate objects are distributed, round robin style, among processors. Again, block reading by an individual processors means that the lines of the data file must also be rearranged in a round robin fashion.

Depending on the simulation, several processors may need to read a single data file. This easily accomplished, in parallel, through MPI (Message Passing Interface) [25]. Within the `MPI_File_open` function, one can specify both the file mode and processor group. The MPI constants `MPI_MODE_RDONLY` and `MPI_COMM_SELF` indicate read-only mode on a single processor. This means, even when many processors are accessing the same file, each processor is given its own file handler. The final step of file reading involves the `MPI_File_read_at` function, with one function call for each line/gate to read. By using parallel reads, this method is both fast and efficient [26].

V. The Model

A. Gate Objects and Messages

The gate level simulation is executed at the Logical Process (LP) level. Each LP models a single gate, sending messages to other gates (LPs). While some data is stored in the ROSS LP object (such as the global identification number of the LP), most information is stored in a state object. This `struct` defines each gate's type, its input and output connections, as well as the statistics for the individual gate. At a global level, gate functions are defined and are indexed by each unique gate type. These functions transform the input and output connection arrays and can be easily redefined for a different EDA gate library.

Within the ROSS framework the state object is transformed based upon received messages. Due to the use of reverse computation, there is only a current copy of each LP's state. The small messages represent (in this model) the electronic high/low or $1/0$ signals between gates. Within ROSS, a message is processed by the forward event handler; if a rollback occurs, the same message is un-processed with the reverse event handler. This allows, by updating the message itself, for single values to be saved between forward and reverse computations.

The lifetime of a gate object within the ROSS framework is as follows:

1) Global Virtual Time < 0: Initialization via file input.
2) GVT = $0..10$: Setup messages to link gates across the network.
3) GVT = $10..end$: Simulation message processing with possibilities of rollbacks.
4) GVT $> end$: Wrap up and final statistic reporting.

B. Simulation Time

In the gate simulation, one unit of global simulation time represents one clock cycle. A basic assumption in this model is that each gate has a delay of one clock cycle. Within the clock cycle a single LP follows the basic timeline in Figure 2. All messages sent from one gate object to another are sent at time $x.5$. These messages have a staggered arrival time, and arrive within a 0.2 clock cycle window. The first message to

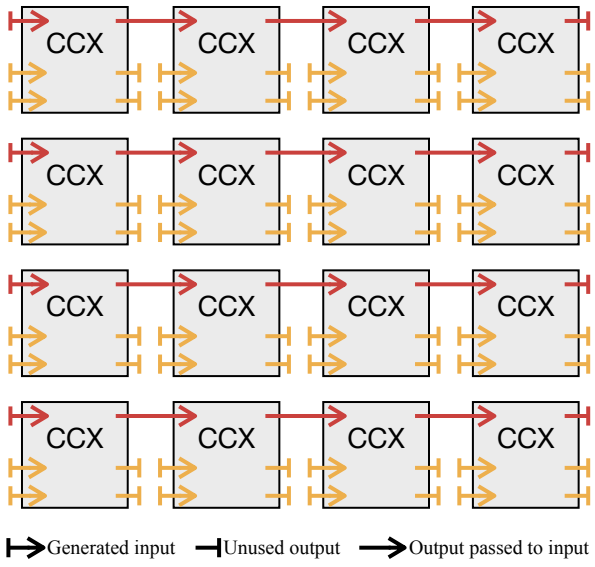

Fig. 3. A 4x4 synthetic circuit. Crossbar instances within the same row are connected along certain data lines. Most input signals are randomly generated for each crossbar instance. This sample circuit contains almost 3.4 million gates.

arrive at a gate object triggers a self-update event, scheduled at the next half clock cycle. This self-scheduled event triggers the next round of inter-gate messages.

The lookahead value is the minimum value timestamp that an LP can use to schedule a future event. Within this model a lookahead value of 0.4 is used. Overall, the average timestamp increment is 0.5 clock cycles.

C. Large Circuits

The future of computing would appear to lie in parallel systems; however, parallelization is happening at the chip level. Single chips contain many computing cores. As the size and complexity of these chips grow, the size of the simulation models grow accordingly. Large scale simulations, on the order of one billion gates will be necessary to execute. The designs for these future technologies are not available to the community. Hence the only option for demonstrating our simulation on such a large scale circuit is to devise our own model for experimentation.

While advanced approaches to synthesis exist [27], the most basic and obvious method is duplication of an existing circuit. In order to efficiently create a large scale circuit for simulation, we duplicated the crossbar. In addition, some data lines (such as the CPU repeaters) are connected across crossbar instances. The overall duplication is grid based, with data line connections occurring along horizontal rows of crossbar instances, see Figure 3.

VI. Results

The circuit simulations were performed on two machines. The first is a 24-core symmetric multiprocessing (SMP) machine, operating at 2.667 GHz. The second is an IBM Blue

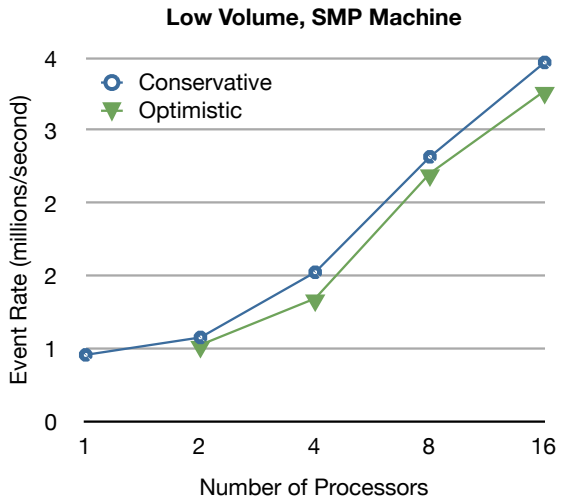

Fig. 4. Single crossbar experiments 2.667 GHz SMP cores. Inputs are generated randomly every 30 simulation clock cycles.

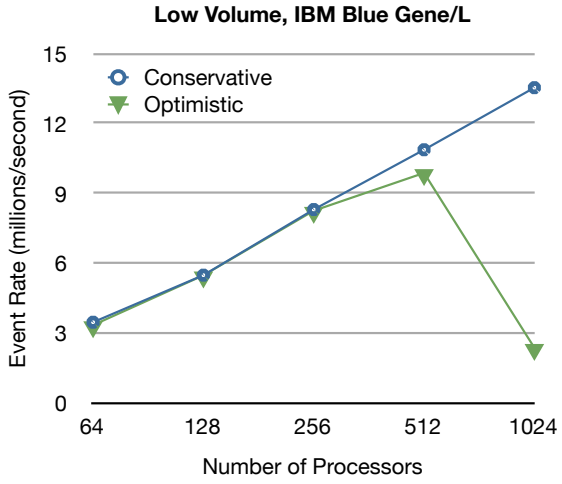

Fig. 5. Single crossbar experiments 700 MHz Blue Gene/L cores. Inputs are generated randomly every 30 simulation clock cycles.

Gene/L, with up to 1024 cores, each operating at 700 MHz. With the Blue Gene, ROSS is able to take advantage of the communication networks [18], [19]. In terms of computing power, 16 SMP cores is equivalent to 64 Blue Gene/L cores. This is due to the faster clock rates, a deep pipeline, and more functional units in the SMP's Intel X5650 processors. Throughout our testing, we focus on the event rate statistic of the simulation. This is principally due to the fact that the simulation is deterministic, resulting in consistent event counts for any sized simulation, irrespective of how many processors are used to execute it. Since each event represents a gate transition, the event rate metric reflects the speed of the overall simulation as well as the (high) volume of events we are able to compute.

For all experiments, the base crossbar component consists

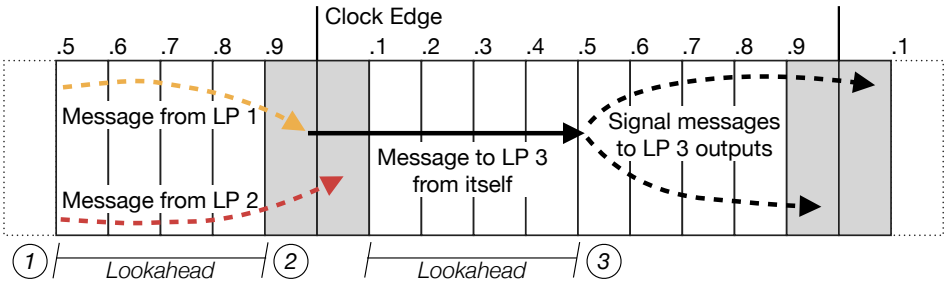

Fig. 2. Simulation clock cycle time slice. All messages are scheduled for at least 0.4 "seconds" in the future, allowing us to use this as the lookahead. ① Messages across LPs are sent at time = x.5. ② Messages randomly arrive within a window around the clock edge, time = x.0. The first message to arrive at an LP triggers a "calculate" message sent to itself. ③ Each updated LP receives a "calculate" message at time = x.5. Here the LP internally calculates its new output values. This values sent to its outputs (step ①).

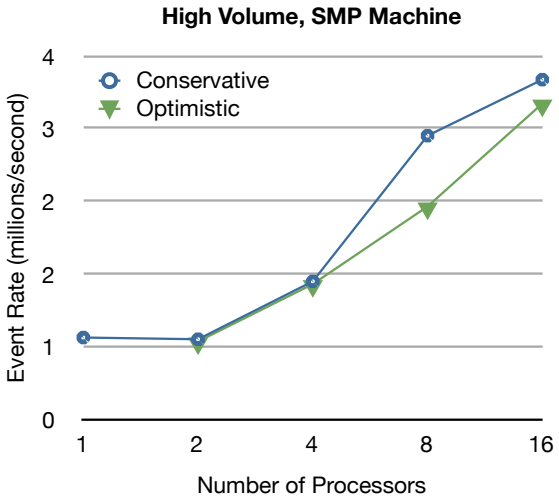

Fig. 6. Single crossbar experiments 2.667 GHz SMP cores. Inputs are generated randomly every 2 simulation clock cycles.

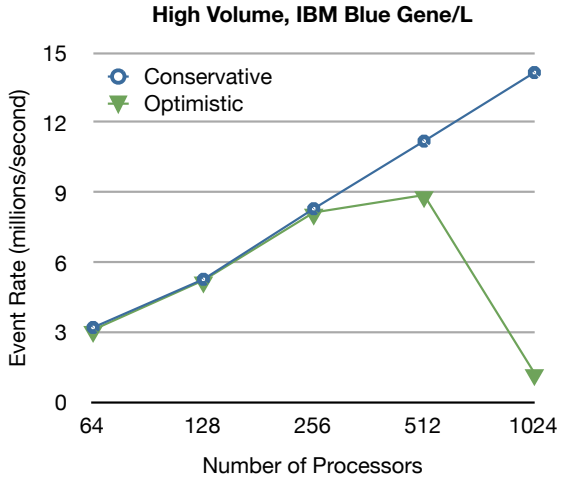

Fig. 7. Single crossbar experiments 700 MHz Blue Gene/L cores. Inputs are generated randomly every 2 simulation clock cycles.

of 211,001 individual gates.

A. Single Crossbar

The initial phase of testing consisted of a strong-scaling scenario: a single crossbar. Obviously, scale up occurred as the number of processors used increased. In Figures 4 and 5 a low volume of inputs was used, one wave of randomly generated inputs every 30 "clock cycles". We also tested a larger overall workload among all processors. Figures 6 and 7 shows a high volume of randomly generated input signals, one every two "clock cycles". These relatively small scale (211,001 gates) experiments generate approximately 1.4 billion events over 3000 simulated clock cycles.

In both of these experiments, it is surprising that the optimistic simulation did not result in better performance then the conservative simulation [28]. This drop off in performance on the part of the optimistic simulation is due to the scant amount of work on each of the 1024 processors. Coupled with a high percentage of remote events, 62.56% in the high volume

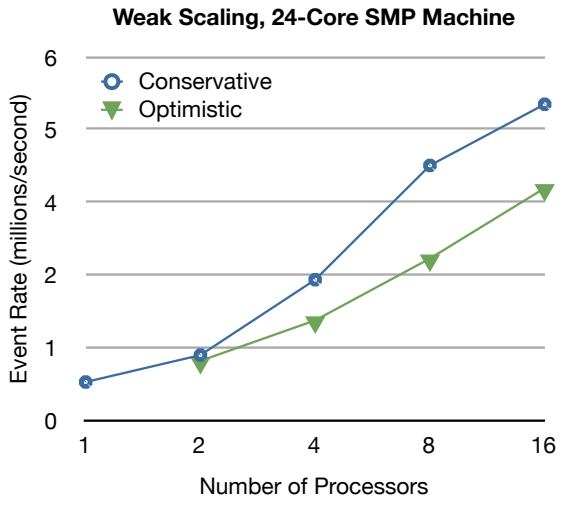

Fig. 8. Large circuit study, with one crossbar instance per processor.

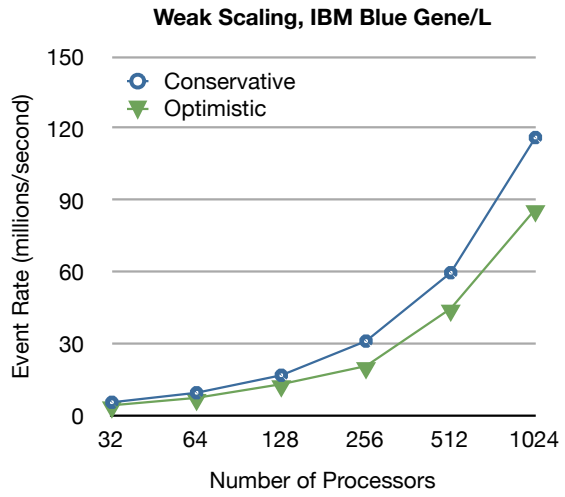

Fig. 9. Large circuit study, with one crossbar instance per processor. Each synthetic circuit was 32 crossbar instances wide.

scenario, we have an overly optimistic execution [2].

B. Large Synthetic Circuit

The next experiment was a weak scaling study, as seen in Figures 8 and 9. This experiment clearly shows the power of the conservative simulation. The largest synthetic circuit, a 32x32 crossbar circuit (a single crossbar on each of the 1024 processors) consists of over 216 million gates. This conservative simulation contained 1.5 trillion events with an event rate of 116 million events per second. Gate-level models at this scale have, to the best of our knowledge, never been done before.

The result is an improvement in execution of at least 131-times. It should be noted that this is only a "conservative" estimate based upon the weak scaling experiments. Even if such a large simulation could be executed on a single core, we would expect to see super linear speedup when compared with a parallel system simulation because of cache memory effects.

C. Optimistic Tuning

In an effort fine-tune the simulation to improve the optimistic performance, we focused on reducing the forced GVT count. A forced GVT is usually an attempt by ROSS to reclaim some memory. Optimistic simulations usually take up more memory than their conservative counterparts due to the need to keep messages in case of rollbacks (and thus reverse computation). The following experiments were conducted on the 24-core SMP machine, using a 4x4 synthetic circuit on 16 cores (Figure 3).

The first experiment explores the effect of the size of the memory allocation (in terms of number of allocated events per processor), see Table I.

As is clear from this table, more allocated memory reduces the forced GVT count. However, memory is not a "free" resource, especially within supercomputing systems. Other variables must also be adjusted to achieve optimistic performance.

Memory Allocated (Millions of Events)	Forced GVTs
0.6	1004
0.8	478
1.0	827
1.2	415

TABLE I
MEMORY ALLOCATION EFFECTS ON FORCED GVT COUNT. EXPERIMENTS WERE RUN WITH CONSTANT PARAMETERS BATCH = 8 AND GVT-INTERVAL = 2048.

Within ROSS, the *batch* and *GVT-interval* parameters determine how many events are processed between successive global GVT computations. Batch specifies how many local events are processed before checking for network events. The GVT-interval defines how many event processing loops occur between GVT computations. The effect of these experiments on number of forced GVTs can be seen in Table II. The high batch and GVT-interval product can be tolerated in this simulation due to the very small (less than 1% remote event count).

Batch × GVT-Interval	Forced GVTs
2 × 1024	1312
4 × 1024	750
8 × 1024	577
16 × 1024	415
32 × 1024	393
32 × 2048	236
32 × 4096	195

TABLE II
EFFECT OF BATCH AND GVT-INTERVAL PARAMETERS ON FORCED GVT COUNT. EXPERIMENTS WERE RUN WITH A CONSTANT MEMORY ALLOCATION OF 0.8 MILLION EVENTS.

We see here that forced GVTs are persistent and difficult to get rid of. Further attempts to improve optimistic performance will be part of our future research.

VII. CONCLUSIONS AND DISCUSSION

In this paper we have shown that gate-level simulation is possible for large circuits. A modestly sized (1024 node) supercomputer was used to demonstrate a 131-times performance increase when compared to a single core sequential simulation. In our most successful simulation, we were able to achieve 116 million gate-transaction events per second for a circuit of 216 million gates. We shall continue our research to improve our simulation speeds and to model more realistic circuits.

The lookahead of 0.4 time units (in relation to the timestamp increment of 0.5 time units) is in the medium to large range, based on the performance study by Carothers and

Perumalla [28]. From this study, we note that conservative synchronization outperforms optimistic synchronization for medium and large lookahead up to 8K cores. We observe the same phenomena here. Should the lookahead become smaller due to changes in the model or core count increase, we expect optimistic performance to improve and potentially overtake conservative performance. Future experimentation is required to confirm this hypothesis. A potential caveat for optimistic performance are high fan-out gate transitions that result in the scheduling of large volumes of events into the future. This places considerable pressure on memory and results in time consuming "forced" GVT computations. More experimentation is needed to determine the performance impact of these high fan-out event computations.

Future research should also examine the impact of circuit properties, such an high fanout, on the efficiency of conservative and optimistic simulations.

REFERENCES

[1] Y.-B. Lin and P. A. Fishwick, "Asynchronous parallel discrete event simulation," *IEEE TRANSACTIONS ON SYSTEMS, MAN AND CYBERNETICS*, vol. 26, 1996.

[2] R. Fujimoto, *Parallel and distributed simulation systems*, ser. Wiley series on parallel and distributed computing. Wiley, 2000. [Online]. Available: http://books.google.com/books?id=b_dQAAAAMAAJ

[3] N. Adiga *et al.*, "An overview of the bluegene/l supercomputer," in *Supercomputing, ACM/IEEE 2002 Conference*, nov. 2002, p. 60.

[4] M. Riepe, J. Silva, K. Sakallah, and R. Brown, "Ravel-xl: a hardware accelerator for assigned-delay compiled-code logic gate simulation," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 4, no. 1, pp. 113 –129, march 1996.

[5] D. Chatterjee, A. DeOrio, and V. Bertacco, "Gcs: High-performance gate-level simulation with gpgpus," in *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, april 2009, pp. 1332 –1337.

[6] V. Bertacco and D. Chatterjee, "High performance gate-level simulation with gp-gpu computing," in *VLSI Design, Automation and Test (VLSI-DAT), 2011 International Symposium on*, april 2011, pp. 1 –3.

[7] H. Avril and C. Tropper, "Clustered time warp and logic simulation," in *Parallel and Distributed Simulation, 1995. (PADS'95), Proceedings., Ninth Workshop on (Cat. No.95TB8096)*, Jun. 1995, pp. 112–119.

[8] L. Li, H. Huang, and C. Tropper, "DVS: an object-oriented framework for distributed Verilog simulation," in *Parallel and Distributed Simulation, 2004. PADS 2004. 18th Workshop on*, Jun. 2003, pp. 173–180.

[9] S. Meraji, W. Zhang, and C. Tropper, "On the Scalability of Parallel Verilog Simulation," in *Parallel Processing, 2009. ICPP '09. International Conference on*, Sep. 2009, pp. 365–370.

[10] Q. Xu and C. Tropper, "XTW, a parallel and distributed logic simulator," in *Parallel and Distributed Simulation, 2004. PADS 2004. 18th Workshop on*, Jun. 2005, pp. 181–188.

[11] S. Meraji and C. Tropper, "Optimization Techniques for Parallel Digital Logic Simulation," *Parallel and Distributed Systems, IEEE Transactions on*, vol. PP, no. 99, p. 1, 2012.

[12] H. Avril and C. Tropper, "The Dynamic Load Balancing of Clustered Time Warp for Logic Simulation," in *Parallel and Distributed Simulation, 1996. Pads 96. Proceedings. Tenth Workshop on*, 1996, pp. 20–27.

[13] S. Meraji, W. Zhang, and C. Tropper, "A multi-state q-learning approach for the dynamic load balancing of time warp," in *Principles of Advanced and Distributed Simulation (PADS), 2010 IEEE Workshop on*, may 2010, pp. 1 –8.

[14] D. R. Jefferson, "Virtual time," *ACM Transactions on Programming Languages and Systems*, vol. 7, no. 3, pp. 404–425, 1985.

[15] L. Li and C. Tropper, "Event reconstruction in time warp," in *Parallel and Distributed Simulation, 2004. PADS 2004. 18th Workshop on*, May 2004, pp. 37–44.

[16] C. Carothers, K. Perumalla, and R. Fujimoto, "Efficient optimistic parallel simulations using reverse computation," in *Parallel and Distributed Simulation, 1999. Proceedings. Thirteenth Workshop on*, 1999, pp. 126–135.

[17] C. Carothers, D. Bauer, and S. Pearce, "ROSS: a high-performance, low memory, modular time warp system," in *Parallel and Distributed Simulation, 2000. PADS 2000. Proceedings. Fourteenth Workshop on*, 2000, pp. 53–60.

[18] D. Bauer, C. Carothers, and A. Holder, "Scalable Time Warp on Blue Gene Supercomputers," in *Principles of Advanced and Distributed Simulation, 2009. PADS '09. ACM/IEEE/SCS 23rd Workshop on*, Jun. 2009, pp. 35–44.

[19] A. O. Holder and C. D. Carothers, "Analysis of Time Warp on a 32,768 Processor IBM Blue Gene/L Supercomputer," *Proceedings of the 2008 European Modeling and Simulation Symposium*, pp. 284–292, 2008.

[20] R. Brown, "Calendar queues: a fast 0(1) priority queue implementation for the simulation event set problem," *Communications of the ACM*, vol. 31, no. 10, pp. 1220–1227, 1988.

[21] D. D. Sleator and R. E. Tarjan, "Self-adjusting binary search trees," *Journal of the ACM*, vol. 32, no. 3, pp. 652–686, 1985.

[22] I. Parulkar, A. Wood, S. Microsystems, and S. Mitra, "OpenSPARC : An Open Platform for Hardware Reliability Experimentation," *Fourth Workshop on Silicon Errors in LogicSystem Effects SELSE*, 2008.

[23] S. Palnitkar, *Verilog Hdl: A Guide to Digital Design and Synthesis*. SunSoft Press, 2003. [Online]. Available: http://books.google.com/books?id=fCSIpgsqkhkC

[24] J. Levine, T. Mason, and D. Brown, *Lex & Yacc*, ser. A Nutshell Handbook. O'Reilly & Associates, 1992. [Online]. Available: http://books.google.com/books?id=YrzpxNYegEkC

[25] (2009, January) Mpi, message passing interface. [Online]. Available: http://www.mcs.anl.gov/research/projects/mpi/

[26] J. Fu *et al.*, "Scalable parallel i/o alternatives for massively parallel partitioned solver systems," in *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, april 2010, pp. 1 –8.

[27] M. D. Hutton, J. S. Rose, and D. G. Corneil, "Automatic generation of synthetic sequential benchmark circuits," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 21, no. 8, pp. 928–940, Aug. 2002.

[28] C. Carothers and K. Perumalla, "On deciding between conservative and optimistic approaches on massively parallel platforms," in *Simulation Conference (WSC), Proceedings of the 2010 Winter*, Dec. 2010, pp. 678–687.

ACKNOWLEDGEMENT

This material is based on research sponsored by Air Force Research Laboratory under agreement number FA8750-11-2-0065. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Research Laboratory or the U.S. Government.