Optimising Simulation Data Structures for the Xeon Phi

Mozhgan K. Chimeh
School of Computing Science
18 Lilybank Gardens, University of Glasgow
Glasgow, G12 8RZ, UK
Email: Mozhgan.kabiri@gmail.com

Paul Cockshott
School of Computing Science
18 Lilybank Gardens, University of Glasgow
Glasgow, G12 8RZ, UK
Email: William.cockshott@glasgow.ac.uk

***Abstract*—In this paper, we propose a lock-free architecture to accelerate logic gate circuit simulation using SIMD multi-core machines. We evaluate its performance on different test circuits simulated on the Intel Xeon Phi and 2 other machines. Comparisons are presented of this software/hardware combination with reported performances of GPU and other multi-core simulation platforms. Comparisons are also given between the lock free architecture and a leading commercial simulator running on the same Intel hardware.**

***Index Terms*—Xeon Phi; many integrated core (MIC); Gate-level simulation; Parallel logic simulation**

I. Introduction

Basing our work on a low-cost SIMD multi-core machine, we describe experiments which use a lock-free architecture to accelerate logic simulation. These experiments verify that the proposed data structures allow SIMD acceleration, particularly on machines with gather instructions (Section VIII-A). Furthermore, on sufficiently large circuits, we can achieve substantial performance gains from multi-core parallelism (Section VIII-B). Moreover, the experimental results show that a simulator using this approach surpasses that of an existing commercial simulator on a standard workstation (Section VIII-D). Besides, these experiments show that the performance on a cheap Xeon Phi card is competitive with results reported elsewhere on much more expensive supercomputers (Section VIII-F).

In clocked synchronous circuits, the utility of simulators depends upon the maximal size of a circuit that can be simulated in addition to its simulation time. Let n be the number of gates in the circuit and t the time in seconds to simulate the circuit for one clock cycle, the raw performance will be n/t.

The performance of a logic gate simulation system is generally expressed in terms of e the number of gate evaluations per seconds. During the simulation, since not every gate will undergo a transition each clock cycle of the simulated machine, it is usually the case that $e < n/t$. The difference between these two quantities is exploited by event-based simulation.

When simulation is performed on single processor machine, the method is known to be effective. Event-based simulation depends on the maintenance of queues. Parallelisation would lead to potential lock contention for the queues. Message passing models of parallelism have queues, for messages, built into their basic communication mechanism. As such message passing parallelism has been successfully used to accelerate event-based simulation.

Massively parallel discrete-event execution on several thousand of processors has been extended to simulating very large numbers of circuits in detailed hardware simulations of microprocessors [1], [2].

In contrast to dynamically scheduling the behaviour of the model during the simulation, static scheduling of all the computations would be well suited for massively parallel machines. This technique is called oblivious simulation. In this paper, we describe a lock-free architecture that allows contention free parallelism on low-cost SIMD multi-core accelerator boards.

II. Related Work

The initial research work on logic simulation started around 1980, where the concept of oblivious (cycle based) and even based simulation was first addressed [3], [4], [5]. Research on parallel simulation algorithms bloomed around the same time targeting both platforms with distributed memory [6], [7], and multiprocessors with shared memory [8].

There are various existing studies on parallelizing logic simulation targeting various platforms such as supercomputers and workstations with accelerators such as GPUs.

Parallelising simulation algorithm targeting SIMD (Single Instruction Multiple Data) hardware systems was first done by [9]. The compiled code logic simulation targeting GPUs did not achieve an ideal performance. Due to the communication overhead between CPU and GPU and not optimizing the data transfers between host and device, the CPU outperformed GPU. Moreover, they did not use the general purpose parallel programming model CUDA.

In contrast to our work, Chatterjee, Deorio, and Bertacco [10], [2], [11], Sen, Aksanli, and Bozkurt [12], and [13], [14] use circuit partitioning algorithms to achieve fast simulation.

Chatterjee et al. propose both oblivious and event-based simulation algorithm. Although their oblivious simulator [10] was simple and statically optimize-able, unfortunately, the size of the circuits that can be simulated by this simulator is limited due to the size of the local shared memory as well as a number of multiprocessor on the GPU. In their event based simulator [2], they partition the design to macro-gates. During the simulation, one or more macro-gates is/are assigned to a multiprocessor. The number of concurrent thread blocks in a multiprocessor can determine the number of macro-gates that are simulated together.

The hybrid simulator (GPU event-based simulator) by Chatterjee et al. [11] uses event-based simulation as a coarse granularity and oblivious simulation within each coarse grain group. Limited amount of shared memory on GPU that is shared among threads in a block, puts a constraint on the size of these micro-gates.

Similar to our work, Chatterjee et al. used lookup table for gate evaluation, whereas Sen et al. used AIGs (And Inverter Graph) representation where all the logic gates in the circuit were `AND` gates. AIG is a way of representing Boolean function manipulation. The technique has been widely used in technology mapping, logic synthetic, and verification [15], [16], [17], [18], [19].

There have been other research works on parallel logic simulation targeting platforms other than GPUs. Gonsiorowski, Carothers, and Tropper [1] study parallel logic gate simulation on supercomputers. They mainly focus on parallel simulation of a OpenSPARC T2 crossbar. Gonsiorowski et al. use Parallel Discrete Event Simulation (PDES) simulation kernel ROSS [20] framework that is built on Jefferson's Time Warp [21] and is designed for PDES.

Similar to our work, Gonsiorowski et al. use gate level netlist with basic Boolean gates and also considers unit-delay model (each gate has one clock cycle delay).

There are existing papers on the acceleration of digital circuit simulation using GPUs [22], [13], [23], [24], [25], [26] and multicores [1], [27]. In our work we use the Intel Xeon Phi [28] to run digital logic simulation.

III. MIC architecture (Intel Xeon Phi coprocessors)

Intel makes alternative technology to GPUs, the Many Integrated Core (MIC) Architecture. The first version is the 22 nm Knights Corner chip, sold under the name of Xeon Phi.The Intel Xeon Phi coprocessors are SMPs that plug into the host (Intel Xeon processor) via PCI Express. The Xeon Phi is configured as a daughter card which runs an independent copy of Linux. It relies on the host motherboard for power and communication. This makes the Xeon Phi physically similar to GPUs.

Xeon Phi cores are based on the Pentium architecture. It has 57 to 61 cores clocked at around 1GHz. There are 4 hardware threads per core, that results in roughly 240 logical cores. Every core has 512-bit wide vector registers, in addition to the standard x86 registers.

Interconnection among cores is based on a ring network model that allows the L2 caches for each core to be accessible by all others. In all, a total coherent cache of over 30MB is available. A Xeon Phi has from 6 to 16 GB of GDDR5 RAM giving around 170 GB/s bandwidth. Each core has its own 32KB L1 cache only accessible locally.

The Xeon Phi is equipped with a new set of instructions, the Intel Initial Many-Core Instructions (IMCI) that is supported by Vector Processing Units (VPUs) within each core. Each VPU supports 512 bit SIMD vectors.

An advantage of using MIC rather than GPUs is that the same code written for a multicore CPU can run on the Xeon Phi coprocessors. This contrast with the way CPU application code needs alterations in algorithm and syntax when ported to a GPU using CUDA. An application written for the MIC architecture using the Intel C or Vector Pascal compilers runs unaltered not only on the Xeon Phi, but also on computers with standard Intel processors [29].

IV. SIMD requirements

SIMD seems initially attractive for simulation since it would allow a single instruction to perform a logical operation on 512 bit worth of data. But this tantalizing vision faces two serious obstacles;

1) All of the bits in the 512 bit word must perform the same operation : AND, OR etc. This seems to imply that a SIMD simulator would have to segregate logic gates into blocks of ANDs ORs etc.
2) If one represents simulated logic signals using packed single bits in a CPU register one needs a means of redistributing the outputs of the previous logic layer into the appropriate packed bit representation. Whilst this can readily be coded using shift and OR instructions, this would impose a serialization that would offset the gains from SIMD parallelisation. No machine currently supports an instruction that loads a packed bit format from a vector of other bit addresses.

A weaker approximation to this type of instruction does exist. The Xeon Phi `vgather` instruction loads a SIMD register `rx` with 16 doublewords such that `rx[i]` is loaded from `mem[ry[i]]` where `ry` is another SIMD register, and `i` is in range 0..15.

Although, a single instruction performs operation on 16 logic signals instead of 512, it is still 15 more than what can be done without SIMD. With 60 cores on the chip, there is a potential for a simulation parallelism level of 960.

Comming up with a data structure that allows both uses the `vgather` instruction and also ensures good cache locality, is the main challenge. Note that the technique is applicable to machines other than Xeon Phi, as similar gather instructions are being made available on AVX-512.

V. Data Structure

A. Levels of logic

We restrict ourselves to simulating synchronous state machines and make further simplifying assumptions as bellow:

- All gates are two input, NOT is represented by a NAND with duplicate inputs, 3 input ANDs made up of pair of 2 input ones etc.

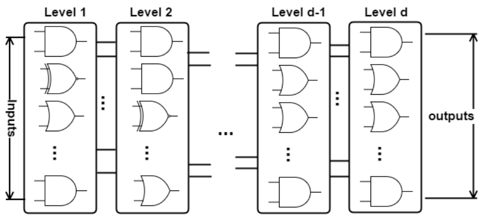

Fig. 1. Levelisation example in a circuit, each of the coloured blocks can be simulated in parallel

- All two input gates have same gate delay, t.

Working backwards from the rising edge of the system clock, the state latches can be affected either by external inputs that feed them directly or by logic gates. No change to an input to a logic gate occurring after a time $-t$ can affect an input to a latch, it follows that there can be no dependencies in the last t of the machine cycle in the set of signals that either feed the latches or between the gates the generate signals that feed the latches. Thus all of these gates can in principle be simulated in parallel. Call this set of signals level N. Clearly we can, by induction, apply the same argument to the signals which feed these gates which we will call level $N-1$. Given a netlist we levelise it as follows:

Step 1. form set of all signals feeding the latches or outputs.
Step 2. push gates whose outputs generate this set onto a stack
Step 3. form set of all signals feeding the set of gates on the top of the stack
Step 4. if this set is empty goto step 5 otherwise goto step 2
Step 5. set n=0
Step 6. pop the stack and label all gates with level n
Step 7. if stack empty terminate, otherwise set n=n+1 and goto step 6

By levelisation (Fig. 1), we can perform parallel independent calculations of whole levels and only force synchronization at the end of each level's simulation [30], [10], [9].

VI. SIMD Based Simulator Architecture

In order to benefit from SIMD architectures, the same operation should be applied to a large number of data elements. Different logic gates perform different boolean functions. However, all can be represented as truth table lookup. We can thus perform AND, OR, NAND, etc. in parallel using SIMD instructions which read an aggregate look up table of size 16x4, which in turn holds truth tables for all binary logic gates. A simplified version of lookup table is as bellow:

```
char lut[24] = {
    0,0,0,1,
    0,1,1,1,
    1,1,1,0,
    1,0,0,0,
    0,1,1,0,
    1,0,0,1
};
```

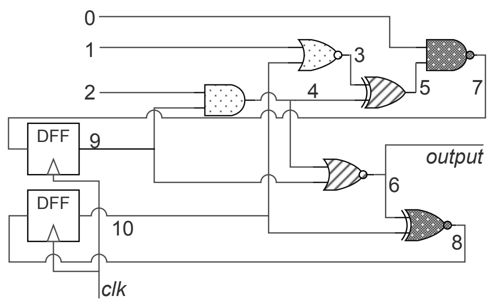

Fig. 2. An example of a circuit with label. Logic gates of the same level are shown in the same color.

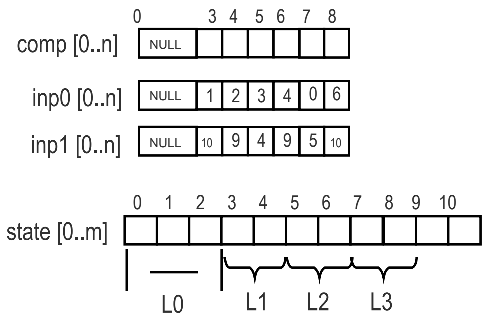

Fig. 3. An example of the array data structure used in the simulator. In practical examples the vectors would be much longer.

To keep the lookup table simple, we have used only basic two input logic gates. Larger circuits are broken down to the level of two input logic gates.

To allow efficient parallel access, we represent the circuit as 4 contiguous vectors. The first three vectors hold the date related to the structure of the circuit: `comp` holds logic types, `inp0, inp1` identify the two inputs to the logic gate. The final vector called `state`, holds the time varying information of the simulation. In other words, this final vector array contains the current state values of all the signals. To update the state of Flip Flops, a separate set of arrays are used, that contain the location of input and output signal of each Flip Flop in the `state` array.

Fig. 3, shows an example of the array data structure used for the given circuit (Fig. 2). The circuit netlist contains all the information related to the gate levels, inputs and outputs, and etc. `state` array holds the signals values from each level close to each other to ensure data locality. Level 0 in the state array contains all the values of input signals to the circuit.

Each location i in arrays `inp0` and `inp1`, contains the ID number of the input signals to the logic gate in location i in array `comp`, where its type is stored. The current state value of its output signal is stored at index i in `state` array.

During each clock cycles, the simulation function is called up to the maximum depth of the circuit, to simulate logic gate of the same level all together 1. Listing 2, line7, shows how the value of current state signals is calculated.

The use of lookup table in this calculation, and the way the data is stored in the arrays, allow this calculation to run

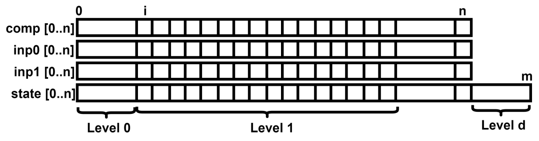

Fig. 4. Example of performing SIMD operation on 512-bits of data in the integer array. Each block is 32 bits.

in SIMD. In order to have an aligned memory access, all the relevant pieces of data for a block of 16 components $i..(i+15)$ stored at the same block of indices in the arrays. This induces adjacent memory access which accelerates the running of the calculation in SIMD. To ease the read and write access pattern, we store chunks of information related to each level in the circuit, next each other in the arrays.

```
int main (int argc, char* argv[]){
 ...
 generate_input (state);
 int index = inputNum;
 for (int level = 1; level < maxLevel; level ++)
      {
    simulation(state, shape_vec[level],index);
  // points to the start of comps in the next Level
    index += shape_vec[level] ;
 }
 ...
}
```

Listing 1. Pseudocode for simulator program. Iteration through levels is sequential Note that the outer for loop start from level 1 as the primary inputs are considered to be at level 0.

```
void simulation (int *state,int glev_Num, int
     index) {
 int i;
 // last index in state that the last gates in this
     level resides
 int last_index = glev_Num+index;
 #pragma omp parallel for simd
 for (i = index; i<last_index ; i++ ) {
    state[i]  = lut[comp[i]*4
                + state[inp0[i]]*2
                + state[inp1[i]]];
    }
```

Listing 2. The function simulates logic gates at a given level with a lookup table using multi-core and SIMD parallelism

Intel Xeon Phi has 128, 512-bit SIMD registers on each of its cores (32 per thread). Components of the same path depth will be simulated as 512 bits chunks of data (Fig. 4). In other words, the load/store, read/write, as well as calculations are done in SIMD on 512-bit of data at the same time (Fig. 4).

Given an array of size N, on Intel Xeon Phi with 240 threads, each physical thread is allowed to process **N/240** elements of the array. On top of this, vectorization allows 16 simultaneous calculations. So, each arithmetic unit only has to do **N/3840** calculations.

During the simulation, logic gates of the same level are divided among threads. The amount of workload for each thread depends on the shape of the circuit (distribution of logic gates per level). Thus, the workload for each thread varies at each level. Each thread performs calculations on an equal piece of data. This ensures work balancing. Fig. 5, shows the workload among thread at each level, during the simulation. The curved lines in the figure symbolize the synchronization between threads.

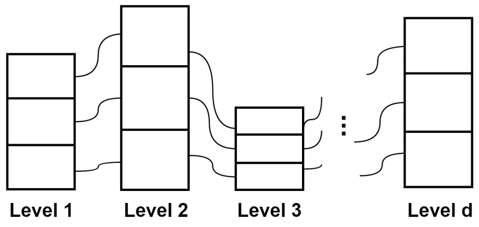

Fig. 5. An example of workload among the threads per level simulation

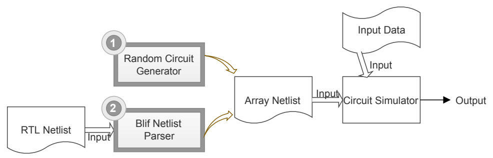

Fig. 6. Process of transforming Netlists to gate Array Netlist for the Simulator

.

VII. Experimental data

A. Test sets

To evaluate our proposed parallel SIMD circuit simulator, we have used test circuits from IWLS benchmark suit, in addition to synthetic circuits.

We took the benchmarks available in BLIF (Berkeley Logic Interchange Format)[1] We used a parser to flatten the circuit and generate the netlist array.

Due to the absence of large benchmark designs (for confidentiality concerns in industry), in addition to the IWLS benchmarks suit, we generate synthetic circuits to be used as benchmark suit for the experiments. The construction algorithm we used is that published in [32] which has been well validated for the way the circuits it builds are representative of real circuits.

The process of generating the test circuits and the simulation itself is shown in Fig. 6.

B. Experimental Setup and Benchmark

We used an Intel Xeon, an Xeon Phi coprocessor as our primary platforms, to evaluate performance of our parallel SIMD simulators.

To assess the performance, the circuit simulator was ran on two different architectures for a varying number of cores over different sizes of circuits. The Intel Xeon Phi 5110S coprocessor with 60 cores, each operating at 1.053 GHz, an Intel Xeon E5-2620 processor operating at 2 GHz, and Intel core i7-2630QM were used. Table I shows the detail specification for the architectures.

Throughout the experiments, we focus on the total physical elapsed time. We also used other metrics such as event rate. We

[1]BLIF [31] is used to represent combinational and sequential logic circuits in logic synthesis and verification tools such as Quartus.

TABLE I
SPECIFICATION OF PROCESSORS USED IN THE EXPERIMENTS

Parameter	Intel Xeon Phi Coprocessor	Intel Xeon Processor	Intel i7
Core, Threads	60, 240	6, 12	4, 8
Clock Speed	1.053 GHz	2 GHz	2 GHz

have used a counter to measure the number of gate transitions. Regardless of how many cores are used, this metric would give us the number events that can be computed per seconds.

For comparison, we also used a commercial simulator (Xilinx) run on our Intel i7 machine.

VIII. RESULTS

In order to validate the effectiveness of our SIMD simulator, we ran the simulator over 1000 clock cycles on two main platforms of Intel Xeon and Intel Xeon Phi.

A. SIMD acceleration

In order to show the effect of vectorization on the performance of the sequential simulator, we ran the simulator on one core with and without SIMD.

With or without the vectorization, the Xeon sequential simulator runs faster the Xeon Phi one. However, the purpose of this experiment was only to show how much improvement the SIMD vectorization would give.

SIMD acceleration increases the speed for up to 10 times on Intel Xeon Phi and up to 4 times on Intel Xeon. We can see that the acceleration falls to 2 for large circuits (Fig. 7). This reduction is due to the size of the circuit and size of the L2 cache.

When the circuit does not fit into the cache, the performance degrades. Note that each logic gate occupies 4 integers (Fig. 3). With 512K L2 cache per core, the maximum size of the circuit that fits the cache is about 32k.

The bit-vector length in Xeon Phi is 512, 256 on Xeon. The expected potential speedup enabling SIMD acceleration: on Intel Xeon Phi is 16, and 8 on Intel Xeon. Meaning 16 logic gates on Xeon Phi and 8 logic gates on Xeon can be evaluated at the same time. However, the achieved vectorization speedup on one core was 10 on Xeon Phi, and 4 on Xeon. This is unsurprising, since Amdahl's law predicts that vectorization speedup would normally be less than the vector length.

B. Multi-core acceleration

The log/log plots in Fig. 8, shows the effect of multi-core parallelism.

On Intel Xeon Phi, as we increase the number of threads (from 1 to 240), we clearly see improvements on larger circuits. From the circuit size of 3 million logic gate, the use of 240 threads shows some speedup.The larger synthetic circuit that was used in these experiment has around 160 million of logic gates. For this circuit size, we achieved the speedup of 10 using 240 threads on Intel Xeon Phi, in comparison to the baseline (the sequential version on Intel Xeon). When

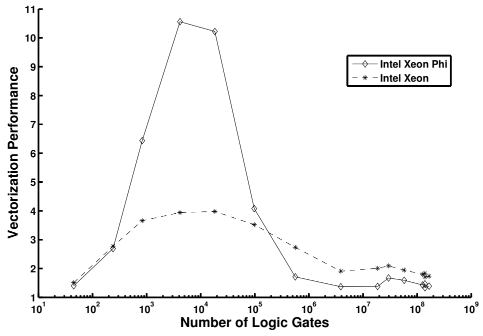

Fig. 7. Acceleration relative to non vectorized code. Both vectorized and non vectorized versions used only 1 core. Note that acceleration gain falls off for larger circuits that do not fit in cache.

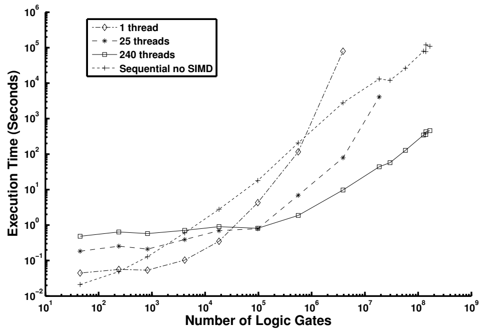

Fig. 8. Performance comparison of multicore SIMD with single core sequential code on Intel Xeon Phi

compared to the sequential version on Intel Xeon Phi, this number increases to 300.

When using multiple threads, not all the resources (hyperthreads) are always available and free to do the simulation. As a circuit grows larger, there will be memory contention (when cores trying to access part of memory that is not accessible) leading to hyperthread stalls. Although smaller circuits fit into cache, they benefit less from multi-threading. We hypothesize that there is not enough work to keep the cores busy and task dispatch overhead degrades the performance.

C. Circuit Connectivity

The results reported so far have used synthetic circuits with random interconnect, random both in terms of which previous layer of logic an input comes from, and in terms of which logic gate within a layer is used. The circuits are thus maximally disordered to provide a worst case.

The same experiments were applied to another set of less random synthetic circuits. In these, the inputs to each logic gate are derived from the immediately previous layer. Randomness is thus significantly reduced, and with the reduction of randomness, we should expect greater locality of access. We call this set the semi-random set.

The observed speedup peaks at 460 as opposed to 300 for maximally random circuits. It shows the effect of data dependency and data locality on the performance. The increased performance is due to the change in write/read pattern. Most

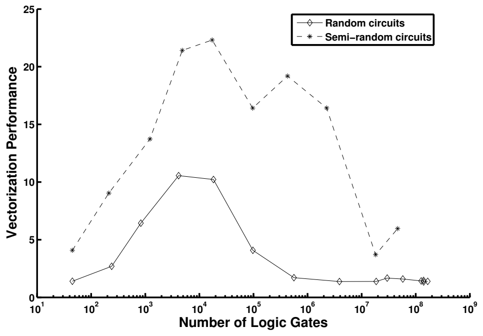

Fig. 9. Semi random synthetic circuits with inputs from previous level only compared with random circuits, both traces using SIMD + multi-cores. Speedup compared to one core non-SIMD on same circuit on Intel Xeon Phi.

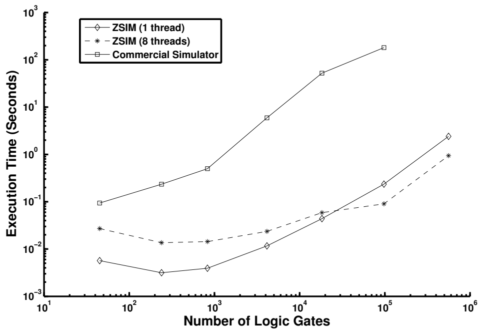

Fig. 10. Performance comparison of our simulator (ZSIM) vs. the commercial simulator on Intel i7 (IWLS benchmark suits)

likely, a lesser number of gather instruction cycles is needed to collect the required data during the simulation.

D. Comparison with a commercial workstation simulator

In this part of the experiments, we used the commercial simulator only on standard Intel chips. So, this comparison did not use the Xeon Phi. The commercial simulator could only take circuits of small size. The circuits are of a size for which our simulator works best with only SIMD and multicore parallelism. The circuits are small and there is not enough work to keeps the cores busy and hide the latency. As we increase the number of threads, the overhead due to thread creation worsen the performance. However, even using one thread on the same machine, our SIMD simulator is much faster than the commercial simulator (Fig. 10)

E. Comparison with simulations on GPUs

There are prior papers on logic circuit simulation on GPUs, though the results reported in the literature are for comparatively small circuits.

In [12], the authors use partitioning and replication in conjunction with levelization in order to handle the problem that the GPUs provide a small amount of shared memory.

It is possible to directly compare the performance of our data structure with the result they report for only two of their circuits, working BLIF representations of the others not being available. Table II shows that when our data structure is run

TABLE II
COMPARISON OF TIME PER GATE SIMULATION FOR INTEL I7 AND NVIDIA QUADRO FX3800 GPU

Design	Time per gate simulation (nano seconds)	
	Alpsen et. al PAR2 (Nvidia GPU)	Chimeh & Cockshott Intel i7
aes-core	3.56	1.88
system-cdes	50.67	1.90

TABLE III
TIME PER GATE SIMULATION AND GATE TRANSITION PER SEC FOR PARALLEL OBLIVIOUS SIMULATION (YUXUAN ET. AL) ON NVIDIA GTX 465

Design	Gates	Nano sec per gate cycle	Transitions per sec
LDPC	60752	6.67	1.50E+8
DES3	52372	2.15	4.65E+8
Or1200	25924	13.3	7.51E+7
OpenSparc	189487	2.17	4.59E+8

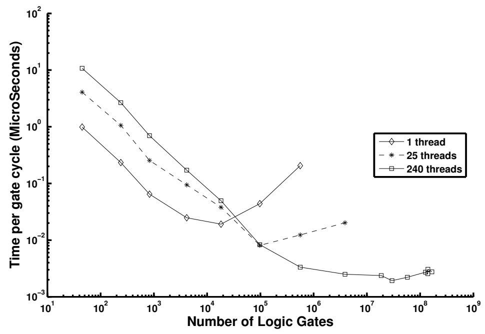

Fig. 11. Comparison of time per gate cycle of multicore SIMD simulator on Intel Xeon Phi. Note that the fast time per gate cycle is around 2.7ns.

even on one core of a standard Intel i7, the performance substantially exceeds the results reported from [12], when we use the metric of nanoseconds per gate simulation.

It is also worth noting that the mentioned paper report results only on comparatively small circuits well under a million gates. So, the applicability of their technique to large circuits is unclear.

Yuxuan et al. [13], introduced a strategy to extract and partition the circuit in order to compile it to GPUs. They presented comparison between the Intel Core Duo T2400 processors with 1.8 GHz frequency and the NVIDIA GTX 465.

They achieved gate cycle times (Table III) comparable to the peak performance of MIC (Fig. 11). This reflects the lower task dispatch cost in CUDA relative to Xeon Phi. The Xeon Phi achieves it best performance on large circuits where the task dispatch cost can be amortized.

Chatterjee et al. report simulation on NVIDIA 8800GT GPU with 14 multiprocessors. Due to no overlap of test circuits, in order to compare the performance of our simulator with the GCS simulator [11], we compare our speedup relative to our sequential commercial simulator mentioned in Section VII-B to the speedup that Chatterjee et al. [11] report relative to a commercial simulator. Their GCS simulator outperforms

TABLE IV
CHARACTERISTIC COMPARISON OF INTEL XEON PHI AND IBM BLUE GENE/L

Parameter	IBM Blue Gene/L	Intel Xeon Phi
Cores	1024	60
Clock Speed	700 MHz/core	1.053 GHz/core
Price	\$0.8m - \$1.3m	\$1600.00 - \$2649.00
Size	2m height x 1m width	24.61cm x 11.12cm x 3.86cm

TABLE V
COMPARISON OF NUMBER EVENTS PER SECOND (IBM BLUE GENE/L VS. INTEL XEON PHI)

Max number of gates $\simeq$ 216 million

Blue Gene/L Cores	Event rate (millions/sec)
64	15
128	20
256	30
512	60
1024	116

Max number of gates $\simeq$ 160 million

Xeon Phi Threads	Event rate (millions/sec)
125	76.8
240	142

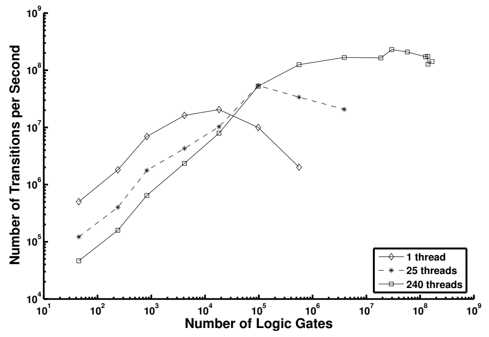

Fig. 12. Comparison of number of transitions per second in parallel simulator on Intel Xeon Phi. Note that these actual transitions in contrast to Fig. 11 which shows time per gate cycle.

their commercial simulator by between 4 to 44 times with an average speedup of 13. Our simulator running SIMD parallelism on one core Intel i7, outperforms our commercial simulator by an average factor of 356.

1) Conclusions relative to GPUs: GPUs can achieve comparable gate cycle per second rates to the Xeon Phi. But this is only been demonstrated on the GPUs for the relatively small circuits. Although it is not explained in the literature why small circuits have been used in GPU experiments, we hypothesize that the relatively small local memory on GPUs motivates experiments to select problems that are easier to map to the local memory. It is clear from our results that Xeon Phi can be extended to the circuits of around 100 millions of gates.

The ring architecture, in which memory accesses by each core are satisfied in the priority: local cache, cache of other cores on ring, GDDR ram, seems very effective. It obviates the need for the programmer to schedule transfers to local memory whilst still giving very good performance even with maximally random circuits.

F. Comparison with simulation on the IBM Blue Gene

Gonsiorowski et al. [1] used a discrete event simulation framework that allows simulations to be run in parallel, called ROSS (Rensselaer Optimistic Simulation System), a modular time wrap system. The paper reports the performance of this framework executing parallel event based simulation (based on the time wrap protocol) using a message passing interface on Blue Gene/L.

1) Blue Gene/L Architecture: The experiments were done on two machines (IBM Blue Gene/L, and Intel X5650). The Blue Gene/L has up to 1024 cores, each performing at 700 MHz clock rate.

To evaluate the simulation performance, we compare the number of gate transitions per second between our simulator and [1].

We are comparing the event metric for our largest circuit (with over 160 millions of gates) with their 216 million gates circuit. On Blue Gene/L with 1024 cores, they achieved an event rate of 116 million events per second.Our simulator achieved an event rate of 141 million events per second (Fig. 12). Table IV, compares some of the characteristics of both Intel Xeon Phi and Blue Gene/L, in terms of the price per rack and the size, in addition to the number of available cores. Table V compares the event rate data taken [1] with the event rate measured in our simulator. We are achieving better performance on many fewer cores at much lower cost. The MIC clock speed is slightly higher than that of the Blue Gene, but the main gain comes from the ability of our data structure to handle both SIMD and multi-core parallelism with low synchronization overhead.

IX. CONCLUSION

In this paper, we proposed a lock-free architecture for accelerating logic gate simulation that allows targeting a low cost SIMD multi-core machine.

We used and applied a data structure on the state of art, Intel Xeon Phi technology. This data structure minimizes the synchronization overhead as well as maximizing the possibility of SIMD and parallel operations. The combination of this data structure and the Xeon Phi chip is a cost effective solution for simulation acceleration.

We have shown that this combination is far faster than, and can handle much bigger circuits than, a widely used commercial simulator running on a workstation. We have shown that the Xeon Phi is competitive with simulation on GPUs and allows the handling of much larger circuits than have been reported for GPU simulation. We also presented results which show that it gives comparable simulation performance to the IBM Blue Gene supercomputer at very much lower cost.

In future publications, we will address the portability of our simulator to other programming languages and to parallel machines by different manufacturers.

References

[1] E. Gonsiorowski, C. Carothers, and C. Tropper, "Modeling Large Scale Circuits Using Massively Parallel Discrete-Event Simulation," in *Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2012 IEEE 20th International Symposium on*, Aug 2012, pp. 127–133.

[2] D. Chatterjee, A. DeOrio, and V. Bertacco, "Event-driven gate-level simulation with GP-GPUs," in *Design Automation Conference, 2009. DAC '09. 46th ACM/IEEE*, July 2009, pp. 557–562.

[3] G. Meister, "A survey on parallel logic simulation," University of Saarland, Department of Computer Science, Misra J, Tech. Rep., 1993.

[4] L. Soulé and T. Blank, "Parallel logic simulation on general purpose machines," in *Proceedings of the 25th ACM/IEEE Design Automation Conference*, ser. DAC '88. Los Alamitos, CA, USA: IEEE Computer Society Press, 1988, pp. 166–171. [Online]. Available: http://dl.acm.org/citation.cfm?id=285730.285757

[5] W. Baker, A. Mahmood, and B. Carlson, "Parallel event-driven logic simulation algorithms: tutorial and comparative evaluation," *Circuits, Devices and Systems, IEE Proceedings -*, vol. 143, no. 4, pp. 177–185, Aug 1996.

[6] Y. Matsumoto and K. Taki, "Parallel logic simulation on a distributed memory machine," in *Design Automation, 1992. Proceedings., [3rd] European Conference on*, Mar 1992, pp. 76–80.

[7] N. Manjikian and W. M. Loucks, "High performance parallel logic simulations on a network of workstations," *SIGSIM Simul. Dig.*, vol. 23, no. 1, pp. 76–84, Jul. 1993.

[8] H. K. Kim and S. M. Chung, "Parallel logic simulation using time warp on shared-memory multiprocessors," in *Proceedings of the 8th International Symposium on Parallel Processing*. Washington, DC, USA: IEEE Computer Society, 1994, pp. 942–948.

[9] A. Perinkulam, "Logic Simulation using Graphics Processors," Master's thesis, University of Massachusetts Amherst, January 2007.

[10] 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.

[11] D. Chatterjee, A. Deorio, and V. Bertacco, "Gate-Level Simulation with GPU Computing," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 16, no. 3, pp. 30:1–30:26, Jun. 2011.

[12] A. Sen, B. Aksanli, and M. Bozkurt, "Speeding Up Cycle Based Logic Simulation Using Graphics Processing Units," *International Journal of Parallel Programming*, vol. 39, no. 5, pp. 639–661, 2011.

[13] Z. Yuxuan, W. Tingcun, K. Yaowen, F. Xiaoya, Z. Meng, and Z. Lili, "Logic simulation acceleration based on GPU," in *Mixed Design of Integrated Circuits and Systems (MIXDES), 2011 Proceedings of the 18th International Conference*, June 2011, pp. 608–613.

[14] T. Hashiguchi, Y. Mori, M. Toyonaga, and M. Muraoka, "Yapsim: Yet another parallel logic simulator using gp-gpu," in *The 19th Workshop on Synthesis And System Integration of Mixed Information technologies*. SASIMI 2015, 2015.

[15] R. Brayton and A. Mishchenko, "Abc: An academic industrial-strength verification tool," in *Proceedings of the 22Nd International Conference on Computer Aided Verification*, ser. CAV'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 24–40.

[16] A. Mishchenko, R. Brayton, and S. Jang, "Global delay optimization using structural choices," in *Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '10. New York, NY, USA: ACM, 2010, pp. 181–184.

[17] A. Mishchenko, S. Chatterjee, and R. Brayton, "Dag-aware aig rewriting a fresh look at combinational logic synthesis," in *Proceedings of the 43rd Annual Design Automation Conference*, ser. DAC '06. New York, NY, USA: ACM, 2006, pp. 532–535.

[18] Q. Zhu, N. Kitchen, A. Kuehlmann, and A. Sangiovanni-Vincentelli, "Sat sweeping with local observability don't-cares," in *Design Automation Conference, 2006 43rd ACM/IEEE*, 2006, pp. 229–234.

[19] S. Chatterjee, A. Mishchenko, R. Brayton, X. Wang, and T. Kam, "Reducing structural bias in technology mapping," in *Computer-Aided Design, 2005. ICCAD-2005. IEEE/ACM International Conference on*, Nov 2005, pp. 519–526.

[20] C. D. Carothers, D. Bauer, and S. Pearce, "Ross: A high-performance, low memory, modular time warp system," in *Proceedings of the Fourteenth Workshop on Parallel and Distributed Simulation*, ser. PADS '00. Washington, DC, USA: IEEE Computer Society, 2000, pp. 53–60.

[21] D. R. Jefferson, "Virtual time," *ACM Trans. Program. Lang. Syst.*, vol. 7, no. 3, pp. 404–425, Jul. 1985.

[22] M. Chimeh, C. Hall, and J. O'Donnell, "Optimisation and parallelism in synchronous digital circuit simulators," in *Computational Science and Engineering (CSE), 2012 IEEE 15th International Conference on*, Dec 2012, pp. 94–101.

[23] Y. Zhu, B. Wang, and Y. Deng, "Massively Parallel Logic Simulation with GPUs," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 16, no. 3, pp. 29:1–29:20, Jun. 2011.

[24] K. Gulati and S. P. Khatri, "Towards acceleration of fault simulation using graphics processing units," in *Proceedings of the 45th Annual Design Automation Conference*, ser. DAC '08. New York, NY, USA: ACM, 2008, pp. 822–827.

[25] M. Li and M. S. Hsiao, "Fsimgp2: An efficient fault simulator with gpgpu," in *Proceedings of the 2010 19th IEEE Asian Test Symposium*, ser. ATS '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 15–20.

[26] M. A. Kochte, M. Schaal, H.-J. Wunderlich, and C. G. Zoellin, "Efficient fault simulation on many-core processors," in *Proceedings of the 47th Design Automation Conference*, ser. DAC '10. New York, NY, USA: ACM, 2010, pp. 380–385.

[27] J. Wang, D. Jagtap, N. Abu-Ghazaleh, and D. Ponomarev, "Parallel Discrete Event Simulation for Multi-Core Systems: Analysis and Optimization," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 25, no. 6, pp. 1574–1584, June 2014.

[28] J. Reinders. (2012) An Overview of Programming for Intel Xeon processors and Intel Xeon Phi coprocessors . [Online]. Available: http://download.intel.com/newsroom/kits/xeon/phi/pdfs/overview-programming-intel-xeon-intel-xeon-phi-coprocessors.pdf

[29] M. Chimeh, P. Cockshott, S. B. Oehler, A. Tousimojarad, and T. Xu, "Compiling Vector Pascal to the XeonPhi," *Concurrency and Computation: Practice and Experience*, 2015.

[30] A. Sen, B. Aksanli, M. Bozkurt, and M. Mert, "Parallel Cycle Based Logic Simulation Using Graphics Processing Units," in *Parallel and Distributed Computing (ISPDC), 2010 Ninth International Symposium on*, July 2010, pp. 71–78.

[31] B. L. Synthesis and B. Verification Group, University of California. Berkeley Logic Interchange Format (BLIF). [Online]. Available: https://www.ece.cmu.edu/ ee760/760docs/blif.pdf

[32] M. Hutton, J. Rose, and D. 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.