EQUIPE: Parallel Equivalence Checking with GPU's

Debapriya Chatterjee, Valeria Bertacco
Department of Computer Science and Engineering, University of Michigan
{dchatt, valeria}@umich.edu

ABSTRACT

Combinational equivalence checking (CEC) is a mainstream application in Electronic Design Automation used to determine the equivalence between two combinational netlists. Tools performing CEC are widely deployed in the design flow to determine the correctness of synthesis transformations and optimizations. One of the main limitations of these tools is their scalability, as industrial scale designs demand time-consuming computation. In this work we propose EQUIPE, a novel combinational equivalence checking solution, which leverages the massive parallelism of modern general purpose graphic processing units. EQUIPE reduces the need for hard-to-parallelize engines, such as BDDs and SAT, by taking advantage of components well-suited to concurrent implementation. We found experimentally that EQUIPE outperforms commercial CEC tools by an order of magnitude on average, on a wide range of industry-strength designs.

1. INTRODUCTION

Combinational equivalence checking (CEC) is one of the most popular formal methods in the context of digital circuit design. The goal of combinational equivalence checking tools is to consider two distinct versions of the combinational netlist of a design and prove or disprove that they are functionally equivalent. These tools are widely adopted in the industry and commonly applied to determine the correctness of intermediate synthesis transformations and optimizations. Several solutions are available both commercially and as research tools; however, for most problem instances, their scalability is still a major limitation.

The technologies available today for CEC can be grouped into two main families: one paradigm is to produce a canonical functional representation for the outputs of the two netlists and then compare the functions obtained (in constant time). In this context, Binary Decision Diagrams (BDD) [4] is most often the data structure of choice. This solution is powerful in coping with netlist pairs that are radically different from each other, however constructing the BDDs for large circuit netlists can prove challenging and time consuming. The second family of solutions poses the problem as a satisfiability problem by constructing a miter circuit which connects corresponding inputs of the two netlists and feeds corresponding outputs to *XOR* gates, which are then *OR*ed together. If the SAT problem corresponding to the miter circuit is satisfiable, it can be derived that the two netlists are not equivalent. In most practical solutions, these two approaches are complemented by a number of structural and signature-based techniques whose main purpose is to reduce the complexity and number of SAT or BDD computations. Structural techniques attempt to prune the netlist portion to be analyzed by finding corresponding internal nodes whose fanin cone of logic is structurally similar, while signature-based approaches generate internal nodes signatures in simulation to rule out the potential equivalency of candidate node-pairs.

Most combinational equivalence checking tools execute on general purpose processors, leveraging over one or a handful of program threads. The advent of massively parallel graphics processors, brings the opportunity for aggressively parallelizing such a computation intensive and widespread application, with the potential to compress the time to market or further optimize industrial-strength digital designs. Graphics processors are suited to execute not only graphics related computation, but also algorithms with a high degree of parallelism and involving a large amount of localized computation. To this end, several vendors have recently developed general purpose programming interfaces that enable users to develop software applications targeting their GP-GPUs (for instance, AMD, NVIDIA, Intel). While parallel solutions for BDD computations and SAT solvers have experienced only mixed success so far, reaching a limited amount of speedup over single-threaded execution, other components in CEC are prone to aggressive parallelization. These include signature-based analysis and structural techniques, which constitute a preponderant fraction of the computation, particularly when the netlists are derived from one another by synthesis transformations, often applied locally. When these latter components can operate effectively, the need for BDD and SAT engines is much reduced, thus limiting the fraction of computation spent on sequential tasks.

1.1 Contributions

In this work we present EQUIPE, a combinational equivalence checker accelerated by GP-GPUs' massive parallelism. EQUIPE includes distributed algorithmic solutions for signature-based analysis and for structural matching. In addition, it relies on a host-based SAT solver for those rare situations where equivalence (or lack thereof) between internal netlist nodes cannot be established with the former techniques.

EQUIPE operates on the two netlists to be analyzed in a levelized fashion, determining which pairs of nodes are equivalent at each logic level, and then using this information in the subsequent levels. When the equivalence of a pair of nodes cannot be determined using the GPU-based engines, a SAT instance is generated and offloaded to the host. The pair of nodes is speculatively assumed to be equivalent and the computation continues on the GPU while the host runs the SAT solver. Once the host has generated an answer, results are updated and propagated as needed. Based on our experience the fraction of SAT instances that find a candidate pair not to be equivalent is extremely small for most benchmarks. This is particularly true in our case, where the benefits of concurrent execution allow us to aggressively leverage signature-based and structural engines, greatly reducing the need for SAT solver calls. When the SAT solver must be invoked, we find that instances are usually small, leading to quick completion.

We implemented our solutions on an NVIDIA CUDA GP-GPU connected to an Intel quad-core host machine and we found that the performance of EQUIPE is 19 times better on average than that of a commercial equivalence checking tool running on the same system.

2. RELATED WORK

The problem of combinational equivalence checking (CEC) has been explored by researchers from the early days of digital design automation. It is commonly used after synthesis transformation and optimization to verify that the functionality of the optimized circuit is preserved. Traditional combinational equivalence checking leverages BDDs to compare the output functions' equivalence.

Many early solutions have explored many variations and improvement over this fundamental idea to achieve extended applicability [7, 15, 2, 12]. However BDD-based approaches suffer from exponential memory usage, hence structural similarity [6, 16] and network cuts through equivalent nodes [13] have been been complementing the mainstream BDD technology.

An alternative approach to equivalence checking is to construct a miter circuit, joining the outputs that needs to be verified for equivalence with an *XOR* gate [3], and the problem of checking equivalence is reduced to testing a stuck-at-0 fault at this gate's output. This can be posed as a SAT problem by considering the CNF form of the circuit and deciding whether the output of the *XOR* gate can be asserted to be 1, which will imply non-equivalence [8]. Recent research in combinational equivalence checking [19, 14] suggests that using a combination of several formal engines, such as BDD and SAT, augmented with techniques such as structural and functional hashing and simulation signatures, may lead to improved performance for this problem. And-Inverter graphs (AIG) are also used as circuit representations to support fast analysis. Functional reduction of AIG's can lead to a AIG representation where each node represents a unique Boolean function. This process can be performed by simulation via a signature-based functional classification followed by the use of a SAT solver to establish functional equivalence [18]. This functional reduction process can be used to perform combinational equivalence checking [19] when applied to the miter of the two versions of the circuit. Identifying large structural isomorphic sub-graphs to establish equivalence has also been suggested [23].

Only very recently the possibility of using general purpose graphics processors to solve complex problems in digital design automation has been explored by researchers. In this domain, [9] attempts parallel fault simulation, and [5] proposes distributed logic simulation on GP-GPUs.

3. INTRODUCTION TO CUDA

General purpose computing on graphics processing units enables parallel processing on commodity hardware. NVIDIA's Compute Unified Device Architecture (CUDA) is a hardware architecture and complementing software interface to design data parallel programs executing on the GPU. According to the CUDA model, GPU is a co-processor capable of executing many threads in parallel. A data parallel computation process, known as a kernel can be offloaded to the GPU for execution. The model of execution is known as single instruction multiple thread (SIMT), where thousands of threads execute the same code operating on different portions of data. Each thread can identify its spatial location by thread ID and thread-block ID, and thus can access its corresponding data.

The CUDA architecture [20] (Figure 1) consists of several multiprocessors (14-30 in current generations) contained in a single GPU chip. Each multiprocessor is comprised of 8 stream processors and can execute up to 512 concurrent threads, all running exactly the same code. The block of threads contained in one multiprocessor has access to 16KB of shared memory, at an access latency of 1 clock cycle. All multiprocessors have access to a global memory which can be 256 MB to 1 GB, known as device memory, and has higher access latency (300-400 cycles). It is possible to amortize the cost of accessing global memory by coalescing accesses from several threads. It is also possible to transfer data from main memory to device memory, possibly in large blocks of data, since the communication is through DMA.

4. EQUIPE OVERVIEW

EQUIPE is a distributed CEC algorithm that leverages the massive parallelism available in graphic processing units. The algorithm is organized in three phases, as illustrated in Figure 2. Two netlists to be compared are provided to the system; typically one is derived by optimization from the other. In our setup we considered synthesized combinational netlists expressed in structural verilog, and applied a number of synthesis optimizations using ABC[1] to obtain the second version. The two netlists are internally converted to AIG form by EQUIPE to check their equivalence.

In the first phase, *Signature generation*, simulation signatures are generated for each circuit node in both the reference and the implementation netlist by running a distributed logic simulation algorithm. In the second phase, *Signature analysis*, these simulation signatures are analyzed to identify potential functionally equivalent nodes between the two netlists through a hashing process, and a database of *candidate equivalent node pairs* is populated.

Finally, in the third phase of EQUIPE, each candidate equivalent node pair is considered to make a full determination of whether the candidate nodes are actually equivalent of not. During this phase each node pair is assigned to a distinct thread in the GPU. When a thread completes its task, it moves on to the next pair. Node pairs are processed by netlist level, starting from the level closest to the primary inputs, and a synchronization among all threads is executed at the completion of each level. This guarantees that when the next level is processed, all the equivalence information of the previous level is readily available and completed. The processing for each node pair consists of performing *2-level matching*, that is, the functional matching of the fan-in cone of the two nodes in the pair, up to 2 levels deep, leveraging the previously computed information on the equivalence among the inputs of the 2-level cone of logic. When 2-level matching is inconclusive, a SAT instance is generated by the thread and transferred to the host CPU for solving, while the GPU thread completes its task by speculatively declaring the nodes equivalent.

In following sections we discuss each of these phases in detail.

4.1 Signature Generation

EQUIPE considers as input two combinational structural netlists. Netlists are converted internally to AIGs and corresponding inputs and outputs are matched by name.

The goal of signature generation is to create signature values for each internal node of the netlist under study using logic simulation. Nodes whose signatures are different are bound not to be equivalent.

In order to exploit the parallelism available we implemented the

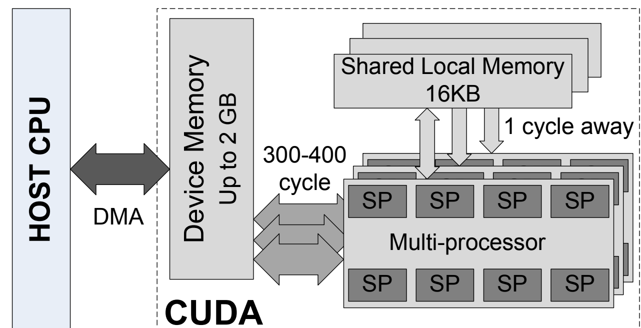

Figure 1: The NVIDIA CUDA architecture The GPU contains an array of multi-processors, all multiprocessors have access to the global device memory and individual exclusive shared memory.

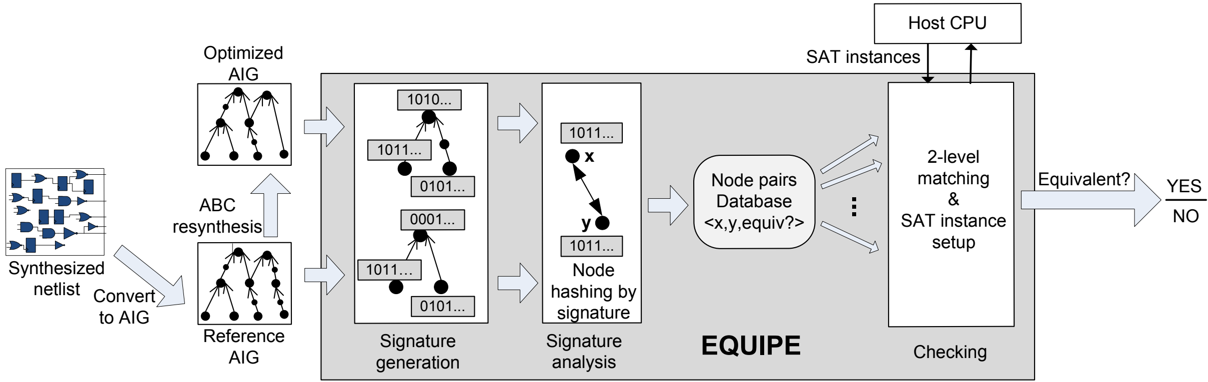

Figure 2: EQUIPE algorithm overview. EQUIPE considers two combinational designs represented by And-Inverter graphs and determines if they are functionally equivalent. Input designs are generated from a synthesized netlist, which is converted to AIG form. The netlist is subsequently optimized and transformed and these two versions of the design constitute EQUIPE's inputs. The algorithm starts by generating simulation signatures for all internal netlist nodes using a distributed logic simulation solution. Then signatures are analyzed to identify potentially equivalent node pairs. These pairs are stored in the *node pairs database*. Finally, individual GPU threads operate on one node pair at a time to determine the equivalence of the two nodes. This step is accomplished by first using *2-level matching*, a mixed structural/functional approach, and then, in case of failure, by creating an appropriate SAT instance. The SAT problems generated in this fashion are solved on the host CPU while work progresses concurrently on the GPU, which speculatively assumes them successful.

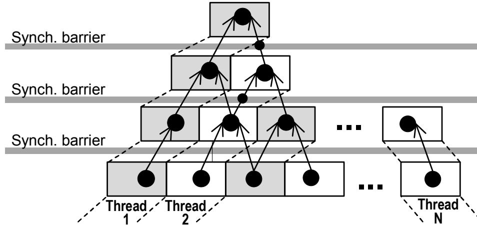

Figure 3: Signature generation. The And-Inverter graph is levelized and simulated concurrently in the GPU: each thread simulating one netlist node at a time. Threads synchronize at the completion of a level, and then proceed to simulate a node in the next level. Once the process completes, a new set of inputs is generated randomly, and the netlist is simulated for another cycle. Signatures are collected by storing the simulation values at the outputs of each nodes over the entire simulation.

simulator with an architecture resembling that of [5]: the netlist is levelized and each level is simulated concurrently. Individual threads simulate one *AND* gate each of the AIG graph. Figure 3 shows a schematic of this process, highlighting how individual execution threads move from one gate simulation to the next after synchronizing at the barrier. As the Figure suggests, it is typical to observe that at the lower simulation levels the netlist is wider, thus requiring more threads for simulation, while at levels closer to the output fewer threads are needed for simulation. We chose not to optimize the data organization to compensate for this phenomenon as [5] suggested: indeed, in our case, the simulation accounts for only a very small fraction of the equivalence checking effort and such optimization would have no impact on the overall performance.

In order to minimize accesses to the global device memory in the GPU, generated signature values are stored in shared memory at first (this occupies one bit per internal node). When shared storage is exhausted the data is transferred to global memory in one single batch. In contrast, the data structure representing the netlist itself is stored in global memory using an array organized by level: this organization allows the GPU to optimize access by executing requests to transfer contiguous blocks of memory from global memory to a same shared memory unit.

The inputs used for the logic simulation are random vectors, which could be generated in the GPU as a separate kernel. The GPU alternates the execution of the input generator kernel with that of the main simulation kernel – simulating one clock cycle of the full netlist each time.

In our experimental evaluation we varied the length of the signature generated to determine a value that would distinguish most nodes. We found that for all of our designs a value of 32 bits was ideal. In most cases, doubling this length would only allows us to distinguish one or two more nodes in a pool of hundreds of thousands. While the time to double the signature length is minimal, the storage space required to storage signatures would double, leading us to settle for the shorter length.

As a case study, consider one of the circuits used in our evaluation, namely the *ldpc* circuit, the reference netlist has 218890 AIG nodes, with 32 bit simulation signatures, 218530 nodes obtain an unique signature while increasing signature length to 64 bits, produces only additional 148 unique signatures, a rather minute improvement. This was the case for other circuits as well.

4.2 Signature Analysis

Signature analysis considers all the signatures collected and determines which pair of nodes from the two input netlists are potentially equivalent. These pairs are added to the node pairs database, residing in global device memory, for further processing.

The process is executed in a distributed fashion by first adding nodes to a hash table based on their signature, and then considering the pool of nodes with a same hash for detailed comparison. We found that, most often, a same signature is only associated with one node per netlist. When multiple nodes have the same signature, we simply consider one from each netlist, and disregard the others. The selection is made by attempting to choose a pair node that belong to close levels in the two netlists.

4.3 Checking

At this stage, candidate node pairs can be considered independent from each other. Individual execution threads in the GPU retrieve one node pair at a time from the database, evaluate their equivalence, and return the updated information to the database. Pairs are analyzed concurrently on a per-level basis. At the end of each level, threads are synchronized and then the operation at the

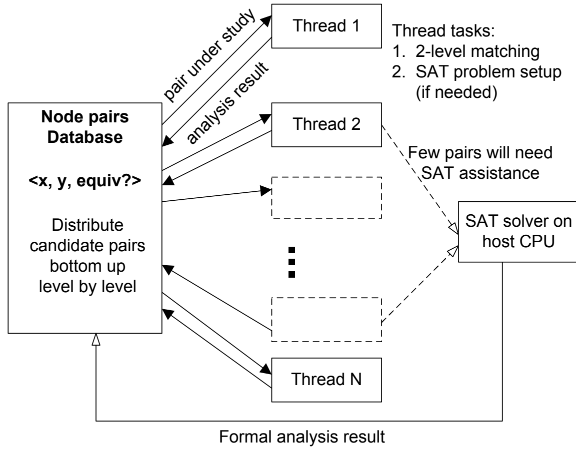

Figure 4: 2-level Matching Candidate equivalent node pairs are analyzed by individual threads concurrently. Each thread builds the 2-level AIG in the fan-in cone of each node and attempts to determine their equivalence based on the AIG structure and the equivalence of the input nodes in the AIG. If this analysis is inconclusive, the thread expands the AIGs and builds a SAT instance to determine the nodes equivalence. The SAT problems generated by the threads are off-loaded to the host CPU, while the node pair is deemed "speculatively equivalent". Computation continues, organized by netlist levels so that, equivalency of fanin nodes is known when operating on a node pair. When the host CPU completes execution, results are updated in the database. Misspeculated equivalencies trigger the re-computation of node pairs in their fan-out.

following level begins. This approach guarantees that the fan-in of a pair of nodes under consideration has been already checked for node equivalency.

The operation in each thread during this phase consists of (i) considering the fanin cone of both nodes in a pair, two levels of logic deep, and (ii) building the corresponding AIG. These two small AIGs are compared to determine their equivalence based on the equivalence of their input nodes and their own structure. Three outcomes may occur: (i) if the inputs are equivalent and the AIGs are equivalent, then the two nodes are deemed also equivalent. (ii) If the inputs are equivalent, and the AIGs are not, then the two nodes are definitely not equivalent. (iii) Finally, if inputs of the AIGs are not equivalent, the local information available is not sufficient to make a final determination. The thread proceeds by expanding the AIG of the fan-in cone of both nodes, until a cut of equivalent nodes is found. These AIGs are then enclosed in a miter circuit and converted to clausal normal form (CNF) for solution by a SAT solver. The SAT problem is off-loaded to the host CPU, and the thread deems the pair of nodes to be ”speculatively equivalent”, that is, the pair is assumed to be equivalent in the subsequent computation, until the host CPU returns with a final answer. Since the vast majority of the time, the SAT solver finds the nodes to be indeed equivalent, operating speculatively under this assumption leads to a minimal amount of re-computation. Node pairs are tagged with the decision determined by these three outcomes, and the database is updated upon the completion of the checking phase for each thread.

4.3.1 2-level Matching

2-level matching leverages the structural information of the 2-level AIGs and the knowledge of functional equivalence of the AIGs' inputs, that is, the four grandchildren of each of the nodes in the node pair. This approach is adapted from [14], however this process under EQUIPE framework is far more powerful as it also

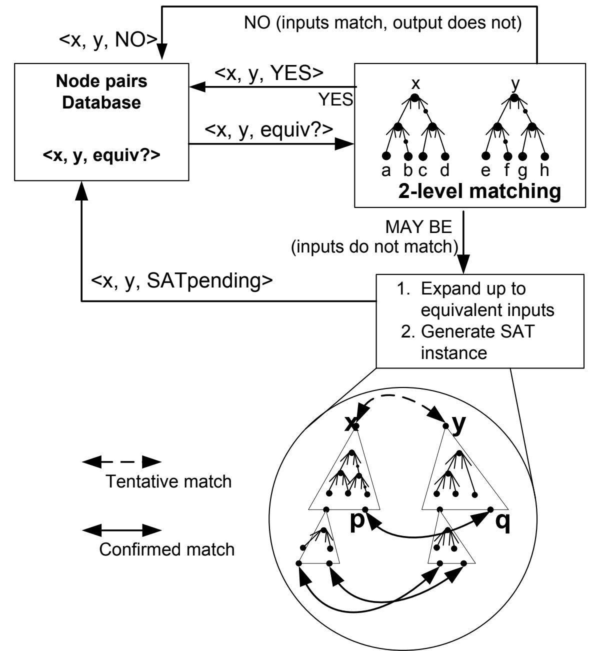

Figure 5: Checking procedure. During the checking phase of EQUIPE node pairs are assigned to individual threads, which in turn process them first through 2-level matching and then, in case of failure, through pruned miter construction. At completion of this task, the thread updates the entry in the database indicating if the pair is equivalent, not equivalent, or pending on the SAT solver.

takes into consideration nodes that were proven to be functionally equivalent by SAT, and not just functionally matched nodes. If the grandchildren are pairwise equivalent and the AIGs are functionally equivalent, then we can declare the node pair equivalent, too. If only AIGs are equivalent, but not the input grandchildren, then we can guarantee that the node pair is not equivalent. If neither of these situations occurs, we need to resort to the next technique, that is, using a SAT solver, as shown in Figure 5.

This method is applied to each node pair at each level, even if we build AIG structures that cross more than one level of logic. This design decision enables us to establish the equivalence of a greater set of candidate pairs without recurring to the SAT solver. Indeed, with a 2-level matching algorithm, we can decide the functional equivalence of nodes that are structurally different in the top two levels of logic. Since 2-level matching may be executed in a distributed fashion, while the SAT solver runs on a sequential thread, we derive a performance advantage from this choice. We also considered extending this approach using 3 or 4 levels of logic in the AIG. However, we found that the number of different functions and of different ways to build those functions impaired the performance of this phase more than the benefit derived by generating fewer SAT instance to solve. The method described is illustrated in the top part of Figure 5, showing the 2-level matching task, and the possible outcomes of that analysis. The next section discusses the activity corresponding to the lower part of the figure, when 2-level matching is inconclusive.

4.3.2 Pruned miter construction

When equivalence can be determined by 2-level matching, a thread

resorts to setting up a SAT problem instance to be off-loaded to the host CPU. This phase proceeds in three steps: first the AIGs are expanded until a cut of equivalent input nodes can be found, then a miter circuit with the two AIGs is built, and finally the circuit is converted into a SAT instance. The fan-in cone of nodes in the pair is expanded, breadth first, beyond the two levels. For these node pairs, the cone of logic below them is explored level by level in a BFS manner: if two functionally equivalent nodes are found in the two cones, the logic below those nodes is not further explored and they are considered primary inputs of the miter circuit to be built. The two logic cones thus explored are joined to form a miter circuit, to decide the equivalence of the corresponding candidate pair. The lower part of Figure 5 (portion in the bubble) illustrate this phase with an example: assume that node p is in the fan-in cone of x at k^{th} level and that q is in the fan-in of y at the same level, and that $< p, q, YES >$ (that is, p and q are equivalent) from previous analysis. Then the logic below p and q need not be explored further and the two nodes can be considered as a single joined input for the SAT solver. The SAT instance problems generated in this phase are typically fairly simple and thus quick to solve for the sequential SAT solver. A quantitative analysis of the complexity of the instances generated is reported in Section 7.2.

Note that it is possible that matched nodes at the inputs of the miter circuit are not independent of each other. As a result, if the SAT solver deems the node pair as non equivalent, the pair may still be equivalent. For these situations, only if the inputs of the miter circuit are not primary inputs, we construct a new miter circuit that includes the complete fan-in cone of the node pair and solve a new SAT instance that will indeed be able to provide a definitive evaluation of equivalence. In our experimental evaluation with a number of complex designs we only encountered this situation a handful of times (a tiny fraction of the overall SAT instances run) and always with node pairs in the lower netlist levels. As a result the full fan-in SAT instance did not have an adverse effect on performance.

5. OVERLAPPING EXECUTION

Once the SAT instances are generated during the checking phase, we solve them using a sequential SAT solver in the host CPU. We made this decision because SAT solver algorithms have proven challenging to distribute over multiple threads, and state-of-the-art concurrent SAT solvers do not provide a significant performance improvement [22, 10]. Moreover, while the host solve the SAT instances generated, we can use the GPU to compute the checking phase of the next level of logic in the netlist. However we have multiple independent SAT instances after finishing each level, which can be distributed among multiple general purpose cores, each running a separate SAT solver process. We report run-times for a 4-core general purpose processor in section 7.1.

As discussed before, node pairs pending SAT decision are speculatively deemed equivalent, and the GPU can advance its analysis to the next level making this assumption for the speculative nodes. When the SAT solver completes, it can update node pair equivalence status in the background. If a node pair was found non-equivalent and its miter did not exclusively rely on netlist's primary inputs, a new instance is generated and provided to the host. Once all the results are in, if any node pair is determined non equivalent, the analysis dependent on it at the higher level are re-run and updated.

Figure 6 shows a schematic of this process, where execution on GPU and CPU is shaded differently. As the figure shows, while the GPU is processing the checking phase for level k, the CPU is solving the SAT instances generated at level $k-1$. Upon completion, the mis-speculated node pairs (frequently none) are updated and the level k analyses depending on them are re-evaluated.

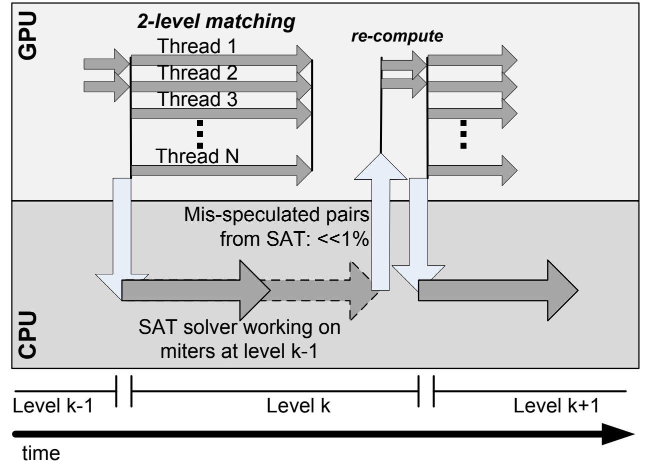

Figure 6: Overlapping execution of GPU and CPU during the Checking phase. The GPU performs the Checking phase on node pairs at level k, while the host is solving SAT instances at level $k-1$. The host may complete before or after the GPU, however, they must synchronize before proceeding to the next level and update potential mis-speculations.

6. EXPERIMENTAL SETUP

To evaluate the performance of the combinational equivalence checking tool, we used a broad range of designs, from purely combinational circuits, such as an LDPC encoder, to the complex OpenSPARC core, comprising almost half a million AIG nodes in its representation. The designs were collected from several sources, the first 5 designs in Table 1 were obtained from the OpenCores section of the IWLS 2005 benchmarks [11], while the next two: b18 and b19, are the largest synthetic designs from the ITC99 suite. The rest of the designs were obtained from OpenCores [21] and the Sun OpenSPARC project [24]. Design parameters are reported in Table 1, which reports number of inputs, output and latches, as well as the number of AIG nodes and netlist levels found after AIG levelization.

Design	inputs	outputs	latches	AIG nodes	levels	simil.
des_perf	122	64	1984	45449	29	61.2
systemcaes	260	129	670	11141	79	63.3
usb_funct	142	121	1739	20532	67	67.1
wb_conmax	1130	1416	770	55675	36	61.2
vga_lcd	99	109	17057	120140	37	55.1
b18	38	151	3320	94813	203	61.4
b19	38	302	6640	196375	208	63.1
3x3 NoC routers	326	324	13434	87618	48	68.1
4x4 NoC routers	578	576	23875	155286	54	68.7
5x5 NoC routers	902	900	37334	467717	55	69.1
JPEG decompressor	36	91	20740	130515	436	56.3
UofT raytracer	544	422	13986	157074	165	57.1
LDPC encoder	1723	2048	0	218961	21	58.4
OpenSPARC core	926	938	62001	385811	273	72.2

Table 1: **Testbench designs for evaluation of EQUIPE**

To generate input design pairs for EQUIPE, we synthesized our testbenches with Synopsys' Design Compiler targeting the GTECH library. We construct a structural verilog netlist of the circuit from the synthesized version. This will be used as the first input netlist for EQUIPE. We then apply a number of synthesis transformations and optimizations using ABC: a synthesis tool chain from the Berkeley logic synthesis and verification group [1]. ABC's re-

synthesis script `resyn` is used to perform these transformations. The optimizations in the script include disjoint support decomposition, refactoring, renoding, sweeping and redundancy removal. The optimized netlist is used as the second input netlist for EQUIPE. To provide an estimate of functional similarity between two versions we also report the percentage of AIG nodes that were found to be functionally identical in the AIG corresponding to the optimized netlist w.r.t. to the original AIG, this is presented in last column of Table 1.

7. EXPERIMENTAL RESULTS

7.1 Performance Evaluation

We compared the performance of EQUIPE against a commercial equivalence checking tool as well as ABC's internal CEC solution. The experimental hardware included a CUDA-enabled 8800GT GPU with 14 multiprocessors and 512MB of device memory, operating at a clock frequency of 600 MHz for the cores and 900MHz for the memory. The host-based portion of the algorithm ran on 2.4 GHz Intel Core 2 Quad CPU. The MiniSat 2 SAT solver [17] was used for deciding the satisfiability of the CNF corresponding to the miters. The baseline EQUIPE setup uses a single thread instance of MiniSat running on the CPU to check the generated SAT instances. In addition, the commercial equivalence checking solution and the ABC tool-chain are also executed on the same host.

Table 2 reports the execution time in seconds for EQUIPE, the commercial tool and ABC for performing CEC between the two versions of the designs. The relative performance speedup for our single-threaded host implementation over commercial solution and ABC is also reported. Note that EQUIPE outperforms the commercial tool by a factor of 3 to 68 times, and shows a considerable performance benefit over the state-of-the-art CEC solution built in ABC tool-chain.

Note that the host-based SAT solver must run several small SAT instances, thus it could benefit from many threads running concurrently on the host. We distributed SAT solving to 4 threads running MiniSat, on a 4-core CPU. Our task distribution algorithm performs the estimates of the SAT instance execution based on the number of clauses in the instance. Table 3 reports the speedup achieved when the host ran 4 SAT solving threads. As the table reports, the performance benefit is improved up to 89 times over the baseline commercial solution for the *NoC-5x5* benchmark, and twice as fast compared to ABC, for certain benchmarks. Note that this multi-threaded scheme can be extended to 16 or more threads on CPU, when future generation of massively parallel general purpose processors become available.

One of the major performance bottle-necks, however remains the fact that at each level the index list of the nodes corresponding to the generated miter structures must be transferred to the host, so that the corresponding CNF can be constructed. These lists can be fairly large for large miters. This aspect is the main limiting factor in the performance boost against ABC, accounting for at least 30 percent of our execution time. This problem can be possibly solved by a better indexing scheme, which can convey the same amount of information in smaller amount of space. Also when a miter CNF instance is found to be satisfiable, it is not possible to confirm non-equivalence unless a large miter up to primary inputs is constructed. This can be circumvented by performing simulation with more patterns for the nodes in question and thus limiting non-equivalence issues by leveraging the faster simulation.

7.2 SAT solver calls

To understand the extent to which the the SAT solver is invoked,

Design	comm.(s)	ABC(s)	EQUIPE(s)	speedup vs. comm.	speedup vs. ABC
des_perf	53.1	5.9	4.1	12.95	1.44
systemcaes	21.3	3.1	2.8	7.61	1.11
usb_funct	33.7	2.5	2.1	16.05	1.19
wb_conmax	29.3	3.3	2.7	10.85	1.22
vga_lcd	176.5	8.1	7.3	24.18	1.11
b18	114.5	24.3	13.1	8.74	1.85
b19	197.3	91.0	47.1	4.19	1.93
NoC-3x3	302.1	9.6	7.3	41.38	1.32
NoC-4x4	621.3	21.2	15.2	40.88	1.39
NoC-5x5	2342.8	49.1	34.3	68.30	1.43
DJPEG	543.4	55.1	28.4	19.13	1.94
RayTracer	613.2	56.3	37.7	16.27	1.49
LDPC	980.3	487.8	257.6	3.81	1.89
SPARC	1807.4	401.2	321.5	5.62	1.25

Table 2: Performance comparison of EQUIPE with the commercial equivalence checking tool and ABC

Design	comm.(s)	ABC(s)	EQUIPE(s) 4 cores	speedup vs. comm.	speedup vs. ABC
des_perf	53.1	5.9	3.4	15.62	1.74
systemcaes	21.3	3.1	2.3	9.26	1.35
usb_funct	33.7	2.5	1.9	17.74	1.32
wb_conmax	29.3	3.3	2.4	12.21	1.38
vga_lcd	176.5	8.1	6.2	28.47	1.31
b18	114.5	24.3	11.3	10.13	2.15
b19	197.3	91.0	34.3	5.75	2.65
NoC-3x3	302.1	9.6	6.1	49.52	1.57
NoC-4x4	621.3	21.2	13.3	46.71	1.59
NoC-5x5	2342.8	49.1	26.2	89.42	1.87
DJPEG	543.4	55.1	21.1	25.75	2.61
RayTracer	613.2	56.3	28.3	21.67	1.99
LDPC	980.3	487.8	208.5	4.70	2.34
SPARC	1807.4	401.2	287.7	6.28	1.39

Table 3: Performance comparison of EQUIPE with the commercial equivalence checking tool and ABC using 4 SAT solving threads.

we provide a case study for the LDPC encoder design. Figure 7 shows how many node pairs where analyzed in each logic level and what fraction of these triggered the generation of a SAT instance. As it can be noted, the fraction of SAT solver calls is very small compared to the node pairs resolved by 2-level matching.

In addition, we studied the average size of individual SAT instances generated. Small SAT instances are usually solved much more quickly than large ones. Figure 8 reports the average number of clauses in the SAT instances generated for LDPC, again clustered by logic level. As it can be noted, the average SAT instances includes less than 100 clauses in the lower logic levels, where the number of SAT problems is high. 100 clauses constitute an extremely small problem for modern SAT solvers. At the higher logic levels the average size is greater and the count is smaller.

7.3 Non-equivalent designs

One of the disadvantages of EQUIPE as mentioned in Section 4.3.2 is the fact, when a miter CNF instance is satisfiable it does not necessarily imply non-equivalence, due to pruned miter construction. In order to determine non-equivalence we need to construct a non-pruned miter up to the primary inputs, which can degrade the performance of our speculative scheme to a considerable degree. When designs are indeed equivalent, this does not pose a serious problem as nodes matched by signature are rarely functionally non-equivalent. To evaluate how non-equivalence affects the

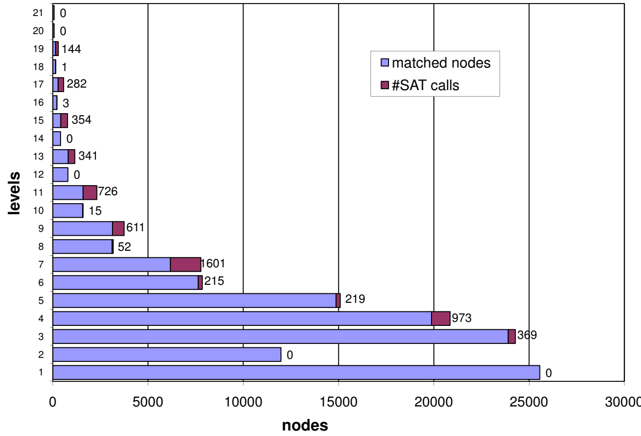

Figure 7: **Node pairs analyzed** per level in LDPC design.

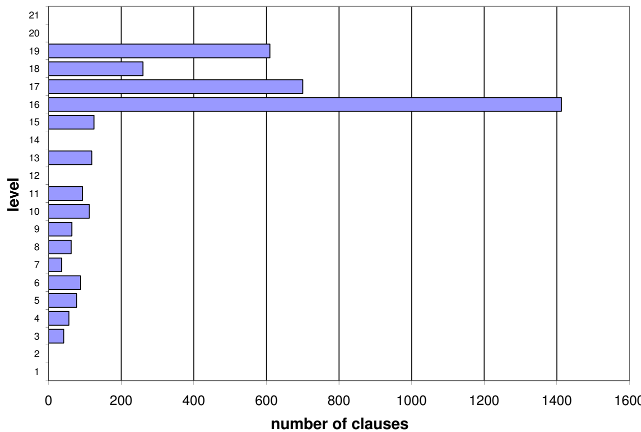

Figure 8: **Average number of clauses in SAT** for LDPC design

performance of EQUIPE we conducted the following experiment: non-equivalence between two designs is introduced by introducing a change in the AIG corresponding to the circuit. We selected 10 random AIG nodes for each level of a circuit to perform this study. A case study is shown in Figure 9 for the LDPC circuit. As it can be noted, most of the non-equivalent cases (all but a few errors at level 9 and 11) were detected by a difference of simulation signatures at one of the primary outputs of the netlist, by fast simulation. However for a few errors in level 9 and 11, simulation signatures were not able to detect the functional non-equivalence: in these cases, we build a large miter to ascertain non-equivalence of two nodes with matching signatures. This results in a considerable performance degradation as can be seen on the worst-case run-times for those levels.

We performed this study for few of the larger circuits in the benchmark set. In Table 4 we report the worst case execution time among all generated non-equivalent circuit pairs. Notice that for the NoC circuits all possible non-equivalent cases were detected by simulation alone, hence the worst-case time is actually smaller than the time needed to establish equivalence. However for the rest of the designs, the non-equivalent case required the construction of one or more large miters, which dominated the solution time.

8. CONCLUSIONS

We presented a novel distributed equivalence checking algorithm, called EQUIPE, that leverages the parallelism of modern graphic

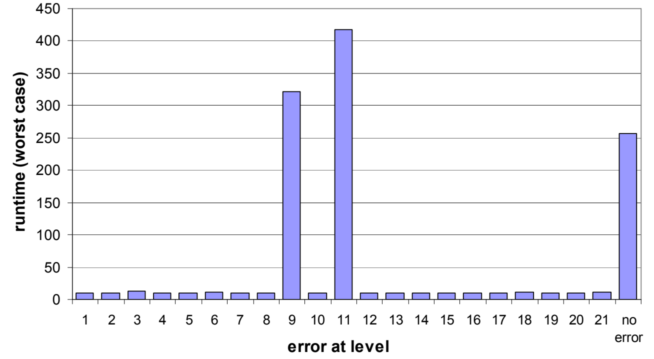

Figure 9: **Effect of non-equivalence**

Design	eqv	eqv 4 cores	non-eqv worst case	non eqv 4-cores
NoC-3x3	7.3	6.1	5.3	4.7
NoC-4x4	15.2	13.3	6.1	5.2
NoC-5x5	34.3	26.2	8.9	7.1
DJPEG	28.4	21.1	32.2	26.4
RayTracer	37.7	28.3	43.2	38.7
LDPC	257.6	208.5	417.3	397.8
SPARC	321.5	287.7	401.2	367.4

Table 4: **Performance for non-equivalent testbenches**

processing units to boost the performance of equivalence checking. EQUIPE includes a novel distributed matching mechanism that uses structural and functional information to determine if internal node pairs are equivalent. This is complemented by parallel solutions for signature generation and analysis. Finally, we developed a speculative mechanism to overlap the execution in the GPU with the SAT solving problems off-loaded to the host CPU. We found experimentally that, on average, EQUIPE delivers a 19 times performance speedup over commercial solution, and can double this result when using multi-core hosts for SAT solving. In the future we plan to leverage more powerful GPU functional units to integrate the two components of EQUIPE (GPU-based and CPU-based) in the GPU.

9. REFERENCES

[1] ABC:a system for sequential synthesis and verification, Berkeley logic synthesis and verification group. http://www.eecs.berkeley.edu/ alanmi/abc/.

[2] P. Ashar, A. Ghosh, and S. Devadas. Boolean satisfiability and equivalence checking using general binary decision diagrams. In *ICCD*, 1991.

[3] D. Brand. Verification of large synthesized designs. In *ICCAD*, 1993.

[4] R. Bryant. Graph-based algorithms for Boolean function manipulation. *Computers, IEEE Transactions on*, C-35(8):677–691, Aug. 1986.

[5] D. Chatterjee, A. DeOrio, and V. Bertacco. High-performance gate-level simulation with GP-GPUs. In *Proc. DATE*, 2009.

[6] C. A. J. v. Eijk and G. Janssen. Exploiting structural similarities in a BDD-based verification method. In *TPCD*, London, UK, 1994. Springer-Verlag.

[7] M. Fujita, H. Fujisawa, and N. Kawato. Evaluation and improvement of Boolean comparison method based on Binary Decision Diagrams. In *ICCAD*, Nov 1988.

[8] E. Goldberg, M. Prasad, and R. Brayton. Using SAT for combinational equivalence checking. In *DATE*, 2001.

[9] K. Gulati and S. Khatri. Towards acceleration of fault simulation using graphics processing units. *Proc. DAC*, 2008.

[10] Y. Hamadi, S. Jabbour, and L. Sais. ManySAT: a parallel SAT solver.

Journal on Satisfiability, Boolean Modeling and Computation, 6:245–262, 2009.

[11] IWLS 2005 benchmarks. http://www.iwls.org/iwls2005/benchmarks.html.

[12] S.-W. Jeong, B. Plessier, G. Hachtel, and F. Somenzi. Extended BDD's: trading off canonicity for structure in verification algorithms. In *ICCAD*, Nov 1991.

[13] A. Kuehlmann and F. Krohm. Equivalence checking using cuts and heaps. In *DAC*, 1997.

[14] A. Kuehlmann, V. Paruthi, F. Krohm, and M. Ganai. Robust Boolean reasoning for equivalence checking and functional property verification. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 21(12):1377–1394, Dec 2002.

[15] S. Malik, A. Wang, R. Brayton, and A. Sangiovanni-Vincentelli. Logic verification using Binary Decision Diagrams in a logic synthesis environment. In *ICCAD*, Nov 1988.

[16] Y. Matsunaga. An efficient equivalence checker for combinational circuits. In *DAC*, 1996.

[17] MiniSat. www.minisat.se.

[18] A. Mishchenko, S. Chatterjee, and R. Brayton. Fraigs: A unifying representation for logic synthesis and verification. Technical report, 2005.

[19] A. Mishchenko, S. Chatterjee, R. Brayton, and N. Een. Improvements to combinational equivalence checking. In *ICCAD*, 2006.

[20] NVIDIA. *CUDA Compute Unified Device Architecture*, 2007.

[21] Opencores. http://www.opencores.org/.

[22] S. Plaza, I. Kountanis, Z. Andraus, V. Bertacco, and T. Mudge. Advances and insights into parallel SAT solving. In *IWLS*, 2006.

[23] S. Ray, A. Mishchenko, and R. Brayton. Incremental sequential equivalence checking and subgraph isomorphism. In *IWLS*, 2009.

[24] Sun Microsystems OpenSPARC. http://opensparc.net/.