CN43-1258/TP
ISSN1007-130X
Computer Engineering and Science
ComputerEngineering& Science
Vol.47, No.2, Feb.2025

Article number:1007-130X(2025)02-0191-09

Research on RTL simulation acceleration technology based on loop expansion on GPU *

Qian Tian, Dan Li, Yue Cheng, Yan Pi, Hongji Zhou

(School of Computer Science, National University of Defense Technology, Changsha 410073, Hunan, China)

Abstract: With the development of open source and agile hardware design methodologies, it is increasingly important to provide efficient RTL simulation support for them. The parallel capability of GPU makes it possible to accelerate RTL simulation by using the structural and stimulus level parallelism of RTL simulation. However, due to the existence of feedback loops in timing design, how to achieve data-level parallelism within a single testbench is still a big challenge. A new method of accelerating RTL simulation using GPU is proposed. The core technology of this method is the identification and expansion of feedback loops in RTL design, as well as the RTL circuit partitioning technology based on this. Circuit partitioning and loop expansion play the parallel capability of GPU to accelerate RTL simulation from two aspects: structural parallelism and data parallelism within a single testbench. Experimental results show that the proposed GPU-accelerated RTL simulation method is 1.2~107.1 times faster than the traditional GPU-based RTL simulation method, and 2.2~14 times faster than the fastest RTL simulator ESSENT. Keywords: RTL simulation; GPU acceleration; PyRTL; hardware construction language; loop expansion

GPU-acceleratedRTLsimulationwithLoopunrolling

TIANXi,LITun,CHENGYue,PIYan,ZOU Hongji

(CollegeofComputerScienceandTechnology,NationalUniversityofDefenseTechnology,Changsha410073,China)

Abstract:Withthedevelopmentofopen-sourceandagilehardwaredesignmethodologies,providing efficientRTL(register-transferlevel)simulationsupporthasbecomeincreasinglyimportant.Theparal-lelcapabilitiesofGPUsenabletheaccelerationofRTLsimulationsbyleveragingstructural-leveland stimulus-levelparallelism withinRTLsimulations.However,duetothepresenceoffeedbackloopsin timingdesigns,achievingdata-levelparallelismwithinasingletestbenchremainsasignificantchallenge. ThispaperproposesanovelmethodforacceleratingRTLsimulationsusingGPUs.Thecoretechnolo-giesofthismethodinvolvetheidentificationandunfoldingoffeedbackloopsinRTLdesigns,aswellas RTLcircuitpartitioningtechniquesbasedonthisapproach.Circuitpartitioningandloopunfoldinghar-nesstheparallelcapabilitiesofGPUstoaccelerateRTLsimulationsthroughbothstructuralparallelism anddataparallelism withinasingletestbench.Experimentalresultsdemonstratethattheproposed GPU-acceleratedRTLsimulationmethodexhibitsaspeeduprangingfrom1.2to107.1timescompared totraditionalGPU-basedRTLsimulationmethods,andaspeedupof2.2to14timescomparedtothe fastestRTLsimulatorcurrentlyavailable,ESSENT.

Keywords:RTLsimulation;GPU-accelerated;Pythonregistertransferlevel(PyRTL);hardware constructionlanguage(HCL);loop-unrolling

* Received: 2023-08-18; Revised: 2023-12-07 Funded by: National Natural Science Foundation of China (U19A2062) Corresponding author: Li Tun (tunli@nudt.edu.cn) Address: College of Computer Science and Technology, National University of Defense Technology, Changsha 410073, Hunan, PR China

1 Introduction

With the continuous development of VLSI (Very Large Scale Integration) technology and the increasing complexity of chip integration, reducing time to market and design costs has become a huge challenge. Inspired by the development of software design technology, agile hardware design methods HADM (Hardware Agile Design Methodology) and open source hardware OSH (Open Source Hardware) design [1] have emerged in recent years . Hardware construction language HCL (Hardware Construction Language) is a domain-specific language DSL (Domain Specific Language) embedded in a high-level programming language, which is one of the key factors for the success of HADM and OSH. Typical HCLs include Chisel on Scala [2] and PyRTL (Python Register Transfer Level) on Python [3]. Using HCL, designers can directly perform register transfer level RTL (Register Transfer Level) design modeling in Scala or Python.

In the design process of HADM and OSH framework, RTL simulation is still an important tool for design verification and plays an important role in the design verification process. Generally speaking, RTL simulation is divided into two types according to the scheduling method used: event-driven and cycle-based simulation. In each cycle, the former will dynamically schedule the corresponding logic unit to be executed according to the change of its input signal value, while the latter schedules all logic units to be executed. Corresponding to the two simulation scheduling methods, researchers have carried out extensive research and achieved rich results. For example, SSIM (Software levelized compiled-code SIMulator) [4] and LECSIM (LEvelized event driven compiled SIMulator) [5] are event-driven simulators, while Verilator and ESSENT (Essential Signal Simulation Enabled by Netlist Transformation) [6] [7,8] are cycle-based simulators.

Compared with CPU, GPU can execute thousands of threads in parallel. Although it has less storage space and control unit, it has a large number of cores for computing. The characteristics of GPU provide researchers with the opportunity to use the parallelism in circuits to accelerate RTL simulation. Therefore, there are a lot of research focuses on how to use GPU to accelerate simulation (including RTL level and gate level simulation, event-driven or cycle-based simulation, etc.) and has achieved certain results.

Chatterjee et al. [9] designed and implemented the first prototype system of event-driven gate-level simulator based on GPU. They used a new macro-gate partitioning algorithm to partition the entire netlist with uniform granularity and regarded macro-gates as events to be scheduled. Zhang et al. [10] improved the work of Chatterjee et al. and proposed a more adaptive partitioning strategy to perform coarse-grained parallel simulation on GPU. Zhu et al. [11] first utilized the power of GPU to

The large-scale parallelism of the CMB (Chandy-Misra-Bryant) [12,13] algorithm was explored . Holst et al. [14] proposed a high-throughput event-driven gate-level simulator using GPU. Here, high throughput means that the simulator can process multiple cycles of data simultaneously with pre-given initial inputs and pseudo inputs. At the same time, references [15] and [16] adopted a cycle-based simulation method at the gate level, using heuristic clustering technology to reduce synchronization overhead by dividing the netlist into macro gates across multiple logic layers, and using GPU to accelerate simulation by converting the netlist into a directed acyclic graph. However, in the above work, the input dependency caused by

the presence of a large number of feedback loops in the sequential circuit hinders the parallel simulation of continuous cycles of the circuit on the GPU, which restricts the further development of the GPU acceleration capability.

“Re-simulation” [17] makes it possible to break the limitations of the feedback loop because during re-simulation, all signal changes can be obtained from the previous RTL simulation results. Therefore, the acceleration effect of using GPU for logic re-simulation is very significant [18,19]. However, due to the lack of re-simulation signal change information during RTL simulation, re-simulation technology cannot be applied to RTL simulation.

QIAN et al. [20] proposed the first event-driven GPU-accelerated RTL simulation framework, which extended the method proposed by ZHU et al. [11] to RTL simulation. However, using GPU to accelerate event-driven simulation may not be a good choice, because all existing works, whether gate-level simulation or RTL simulation, face the problem of function scheduling and communication overhead involved in event-driven. RTLFlow [21] is the latest cycle-based Verilog RTL simulation flow on GPU. Inspired by the research of ZHANG et al. [22] , RTLFlow achieves high-throughput RTL simulation by simulating inputs from different stimuli simultaneously.

The following conclusions can be drawn from existing

work: 1) Existing automated processes for converting RTL designs into CUDA (Computer Unified Device Architecture) programs are mostly targeted at hardware description languages, such as Verilog, and existing methods cannot be directly applied to the conversion of HCL designs into CUDA programs. 2)

For data within the same testbench, due to the limitations of the feedback loop in the sequential logic circuit, most existing methods only pass in one cycle of data within the same testbench for simulation at a time.

To address the above problems, this paper proposes a new GPU-accelerated RTL simulation method. By using loop expansion and the Maximum Fanout Free Cones (MFFC) [8] partitioning algorithm, the RTL circuit described by the partitioned HCL is mapped to the GPU to form a CUDA program for simulation. The experimental results show that when the number of simulation cycles is 2, the method based on HCL is more [17] This paper proposes efficient than the method that only explores the circuit structure parallelism without using the same

The data parallel method in the testbench can achieve a speedup of up to 107.1 times. Compared with the most advanced open source simulator ESSENT, the acceleration ratio can reach up to 14 times.

The main work of this paper is summarized as follows:

1) A method for converting HCL descriptions into CUDA programs is proposed. Automatic process, and realize the data in the same Testbench Run simulation.

2) A new RTL graph partitioning for GPU is proposed This method uses loop expansion to maximize the use of GPU parallel capabilities.

3) This work is the first basic A GPU-accelerated RTL simulation method based on RTS cycle.

2 Background knowledge

2.1 GPU and CUDA

CUDA is a general-purpose parallel computing platform and programming framework The part that can be calculated in parallel is called the kernel function. CUDA can be used to implement parallelism on NVIDIA GPUs. By setting parameters, thousands of threads can be mobilized to perform calculations. Execute computing tasks simultaneously.

Assume there are 2 independent operations, each of which needs to be processed 64 cycles of input, use "kernel<<<2,64>>>()" to adjust 128 threads are used to execute these two independent computing tasks simultaneously. These threads form a grid and are divided into 2 blocks. Each A block contains 64 threads that can process multi-cycle inputs in parallel.

When using GPU to simulate logic circuits, Independent logic units set up a block to simulate. Multiple threads can simultaneously process the multi-cycle inputs of the logic unit and And these threads can communicate quickly through shared memory.

In actual operation, the GPU is scheduled in groups of 32 threads. Even if there are less than 32 threads, the same as 32 threads will be used. The same hardware resources are used to schedule the threads. When choosing the number of threads, try to choose an integer multiple of 32. In addition, blocks also have a maximum The maximum number of threads in a block is limited. The number of threads is usually 1024. Therefore, when programming, you need to ensure that each The number of threads in a block does not exceed this upper limit.

In summary, by using CUDA for GPU programming, Developers can fully utilize the parallel computing capabilities of GPUs; Set the number of threads and blocks appropriately and use shared memory for communication It can improve the computing efficiency.

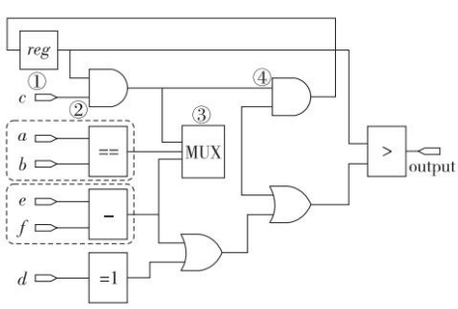
2.2 PyRTL

This paper develops a hardware construction language PyRTL based on Research on simulation acceleration on GPU. PyRTL is an agile hardware Description language, which provides a series of RTL design, simulation, Tracking and testing tasks, through two built-in data structures Structure - Block class and WireVector class, from bottom to top In PyRTL, the Block class stores Basic logical unit, each basic logical unit is stored as a LogicNet object, the interconnection information between units is represented by WireVector Object storage. All information stored in the Block object is reflected in In the intermediate format IR (Intermediate Representation) of PyRTL, Figure 1a shows a simple Logic circuit,Figure 1b shows the corresponding logic circuit diagram.

```
a = Inputbitwidth =3, name ='a')
b = Input (bitwidth =3, name ='b')
c = Input (bitwidth =3, name ='c')
d = Input (bitwidth =3, name ='d')
and = Input (bitwidth =2, name ='and')
f = Input (bitwidth =2, name ='f')
reg = Registerbitwidth =3, name ='reg')
output = Output (bitwidth =1, name ='output')

equal = a == b
tmp 1= reg &c
tmp 2= and f
m = mux (equaltmp 1, tmp 2)
reg . next <<= m &tmp 1
n =~ d
sum = tmp 2| n
the = sum | m
output <<= reg > the
```

a PyRTL design example



b Logic circuit diagram

Figure1 ExampleofdesignedbyPyRTLand itscorrespondinglogiccircuitgraph

Figure 1 PyRTL design example and its corresponding logic circuit diagram

Obviously, the two logical units marked by the dotted boxes in Figure 1b The inputs of the two operations “” in Figure equal = a == b "and “ tmp 2= and f 1a are both from the initial input (Input Therefore, these two operations are independent of each other in the simulation process. The simulator can obtain these two operations in multiple cycles at the same time input data (for example, 64 cycles of input data), and then As mentioned above, in CUDA, "kernel<<<2,64>>>()" sets the number of threads to 64. Blocks are used to implement parallel simulation of the structure of these two operations and 64 cycles

Data parallel simulation of the period.

3 Research Motivation and Overall Idea

In existing GPU-based parallel simulations,
In the Testbench, the input data of each cycle is usually sent in sequence.
The parallel capability of GPU is fully utilized to generate
In this simulation process, some special
Special logic elements (such as registers) are usually split into two
Nodes - Pseudo-input and Pseudo-output. This behavior is called
Figure 2 shows how this strategy can be used to accelerate the GPU.
Simulated situation.

Take the design in Figure 1b as an example to introduce the breaking operation.
Point 9 corresponds to a register in the logic circuit. In Figure 2, node
9 is divided into two parts: out_9 and in_9.
becomes the pseudo input of the circuit, and in_9 points to the pseudo output.
Although all input data in the Testbench is
All of them have been transferred to the GPU before, but they are performed on the GPU
During simulation, only one cycle of input data is passed in each time.
The kernel function contains all the logical units that can be calculated currently.
Therefore, the number of blocks declared by each kernel function in Figure 2 represents the logic
The number of logic units is increased to realize parallel computing of logic units.
The minimum unit of scheduling on the GPU is 32 threads, so even if only
One active thread is required for calculation, and each block still contains 32
Figure 2 marks the completion times on the kernel function scheduling times axis.

The total number of kernel function calls required for the simulation of a circuit cycle
This assumes that the number of threads that can be run simultaneously in a block is less than
The number of processes is less than 1024 (the actual situation may be smaller).
Note that in the case of a broken loop, a total of 4 calls are required.
Kernel function to complete the simulation.
* 32 threads, but only 10 are actually used
The thread usage is 1/32.

From the above analysis, we can find that to give full play to the parallelism of GPU
There are two main issues to be addressed:

1) How to make the best use of all threads in a block?

2) How to maximize the number of threads that can be computed simultaneously in a block
Lots of land?

In order to solve these two problems, this paper proposes a new strategy
Loop expansion: Identify loops in a circuit and extract them.
In this way, the thread waste is limited to
The ring allows the remaining logical nodes to fully utilize the lines in the block.
In this case, you can increase the number of threads in the block to make the
Multiple threads can perform calculations simultaneously.

Figure 3 shows the GPU acceleration using the ring expansion strategy.
In the dotted oval box, node 9 represents the storage
Nodes 0, 4, and 6 are affected by register feedback and affect the register
These four nodes form the ring mentioned in this article.
Finally, the ring is encapsulated into a new node—Macro node.
The kernel function corresponding to the Macro node is looped by an active thread

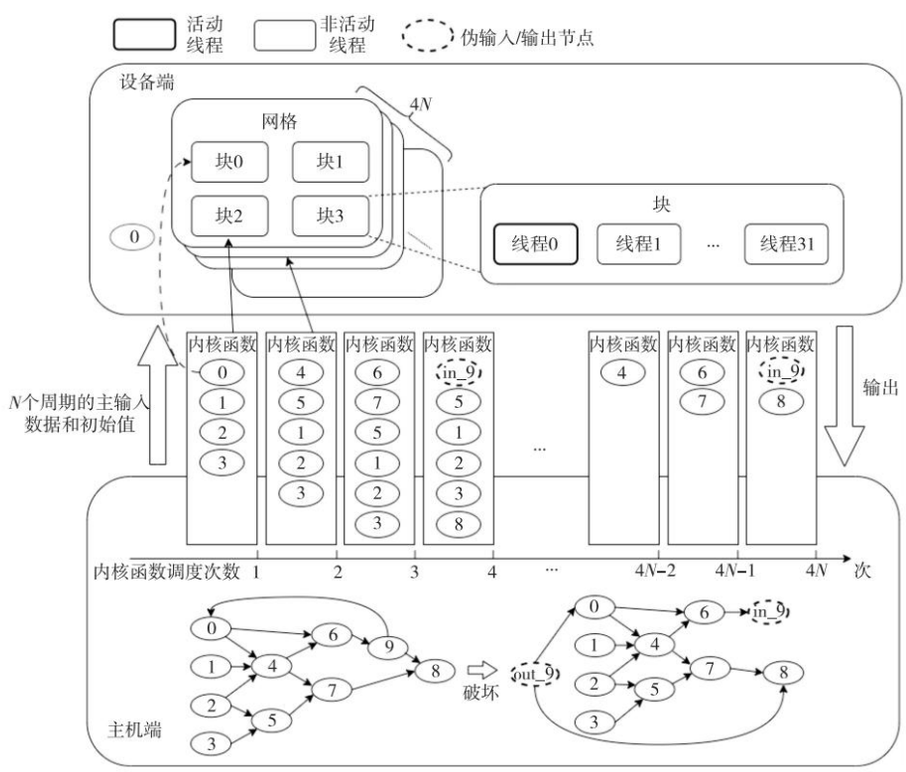


Figure2 GPU-acceleratedRTLsimulationmethodwithloopbreaking

Figure 2 GPU accelerated RTL simulation method using a loop breaking strategy

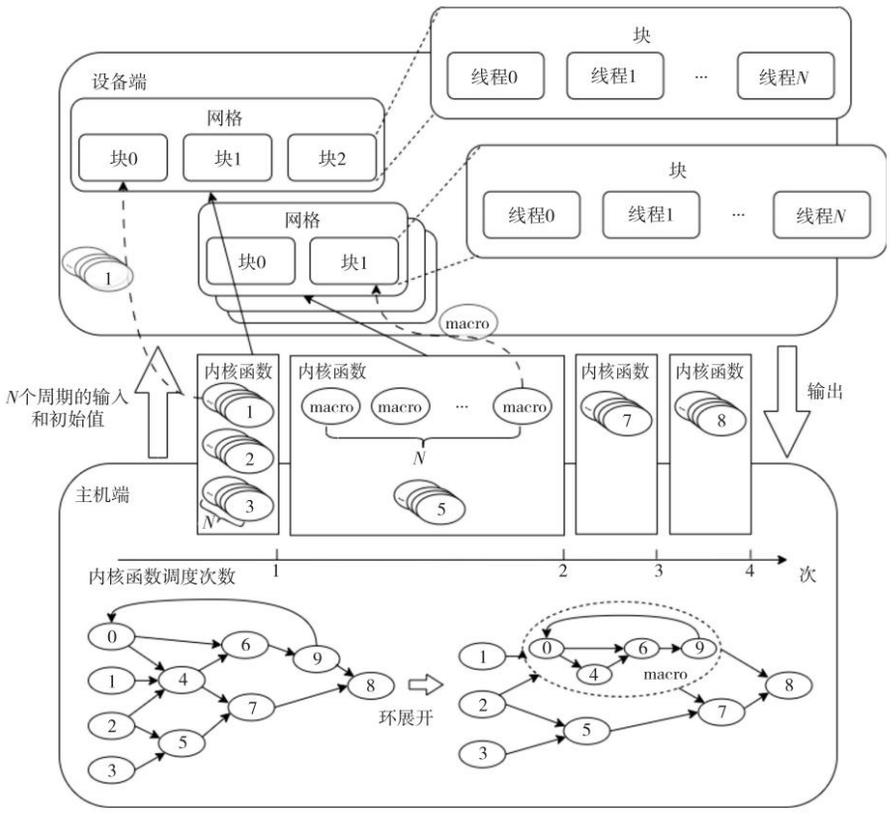


Figure3 GPU-acceleratedRTLsimulationmethodwithloopunrolling

Figure 3 GPU accelerated RTL simulation method using loop expansion strategy

The simulation tasks of the nodes in the remaining N kernel functions are Scheduled to different blocks to achieve structural parallelism. Points (except the Macro nodes mentioned here) can get the same The input or intermediate output of a cycle in a Testbench. Each thread in a block can independently process a cycle Data, so each logical node corresponds to the thread in the block are fully utilized, thus achieving data parallelism.

When using the ring expansion strategy, the total number of declared threads is reduced to 7 N , the number of threads actually used is N+1. The utilization rate is 6/7+1/(7), which is obviously greater than when using the destruction strategy. 1/32 of thread utilization.

Therefore, using ring expansion can achieve both structure and data dimensions. degree of parallelism and further obtain higher thread utilization, that is, In the same testbench simulation, using loop expansion can better play The potential of GPUs.

4 GPU simulation acceleration method based on ring expansion

Figure 4 shows the GPU based ring expansion proposed in this paper. The steps of simulation acceleration method: 1) According to the intermediate format of PyRTL The RTL circuit is constructed as a DGL (Deep Graph Library) [23] graph. 2) Find all feedback loops starting from the register and The found ring is encapsulated as a Macro node. After the ring is encapsulated, the DGL graph will be 3) Perform MFFC partition on the reconstructed acyclic graph. In this step, the ring encapsulated in the previous step is treated as a separate partition. Each MFFC partition will also be encapsulated as a Macro node, and reconstruct the DGL graph according to the partitioning results. After the entire graph is divided into layers according to the topological logic, Into a static CUDA program. When compiling the generated CUDA program After that, you get a GPU-accelerated RTL simulator.

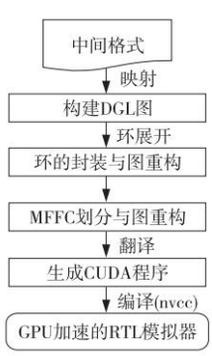


Figure4 FlowgraphofGPU-acceleratedRTLsimulation

Figure 4 GPU accelerated RTL simulation flow chart

4.1 Constructing DGL Graph from IR

Extract the connection relationship between each node in the circuit and abstract the circuit into A directed graph is the first step of the whole process, according to PyRTLIR The following information can be obtained:

1) The connection relationship of each LogicNet;

2) The topological order of each LogicNet;

3) Register nodes that may form feedback loops. As shown in Figure 5, a DGL graph with node features is constructed based on the information of types 1) and 2) contained in the IR of Figure 1a. The node features can ensure that the dependencies between the initial LogicNets are not lost during the reconstruction of the DGL graph.

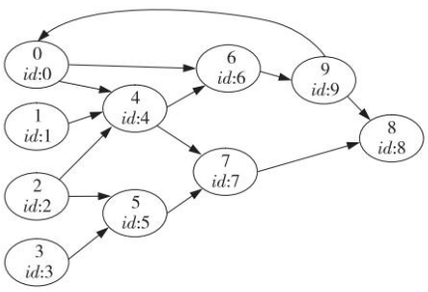


Figure5 ExampleofDGLgraph

Figure 5 DGL graph example

4.2 Ring Encapsulation and Graph Reconstruction

The first step in ring encapsulation is to identify the ring. The purpose of identifying the ring is to find the node set that affects the register logic unit. The classic method to solve similar problems is the cone of influence (COI) algorithm [24], but the set that this paper needs to find is actually a subset of COI. Therefore, this paper uses the Tarjan algorithm, which is a classic algorithm for finding strongly connected components (SCC). Based on the three types of information in Section 4.1, starting from the register node and moving toward the initial input direction, Tarjan can identify all SCCs in the graph. Each SCC contains all the nodes that affect the change of the register value, which is the node set required by this paper.

Each SCC found will be encapsulated as a new Macro node. In terms of implementation, to represent the Macro node, this paper defines the Macro class to save the input and output information of the Macro node, as well as the node characteristics of all nodes in the encapsulated SCC (such as id: 0, 4, 6, 9 in Figure 5). The input and output information of the Macro node comes from the connection relationship between the internal and external nodes of the Macro, as well as the connection relationship between the nodes in the ring and the initial input and initial output. In Figure 6, the internal node of the Macro receives the input of "tmp 0/1W" and "tmp 2/3W" Enter, through "tmp from external nodes 1 and 2, and outputs "reg/3W" and "reg/3R" to external nodes 7 and 8, and the node 0 in the ring has an initial " c input /3I", so through encapsulation, we can get a node as shown in Figure 6. Macro node.

Figure 7 shows the reconstruction result of the DGL graph after the ring is encapsulated. Node 3 is the new Macro node. Node 3 is added to the DGL graph instead of Figure 6. Its node feature id is 10, which is the number of nodes in the initial DGL graph plus 1. If more than one ring is encapsulated, the node feature value of the next encapsulated Macro id node is 11. When there are multiple SCCs in a graph, the above process

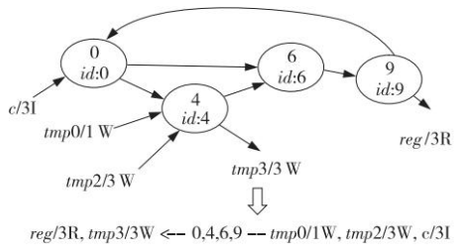


Figure 6 Packaging of loop

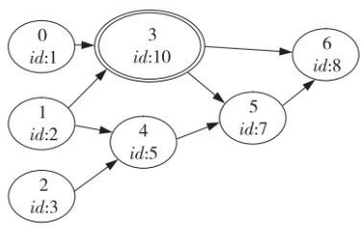


Figure 7 Reconstitution of DGL graph Figure 7 Reconstitution of DGL graph

The process will be repeated until all SCCs are packaged.

After the ring is encapsulated, the entire graph is transformed into a directed acyclic graph. For this article, a directed acyclic graph is the graph that can realize structure and data parallel simulation.

4.3 MFFC Partitioning and Packaging

The purpose of MFFC partitioning is to divide the entire graph into a directed acyclic graph consisting of many coarse-grained units. In this graph, the size of each unit is as consistent as possible, so that the simulation overhead of each thread is as close as possible to give full play to the computing power of the thread.

When performing MFFC partitioning, it is first necessary to calculate the MFFC partition of each node. The MFFC partitioning algorithm will divide the entire circuit into non-intersecting acyclic partitions based on the calculated MFFC partitions of each node. In recent studies, researchers from ESSENT [7,8] used this technology to assist in the partitioning of RTL circuits. As indicated by the dashed box in Figure 8, through MFFC partitioning, several logic nodes are partitioned into a coarser-grained basic simulation unit. In this partitioning process, since the Macro node (node 3) already contains multiple logic nodes, it itself is treated as a basic simulation unit and does not participate in the partitioning. Each partitioned MFFC partition is also encapsulated as a Macro node, and the DGL graph is reconstructed again.

4.4 Two-dimensional parallel CUDA program

When generating a CUDA program, as shown in Figure 9, the basic simulation units of each layer are arranged in the same kernel function, and different blocks are allocated through switch/case statements to simulate different units. Threads executing the same kernel function will first enter different cases according to the block number to which they belong, and then enter different cases according to the block number within the case.

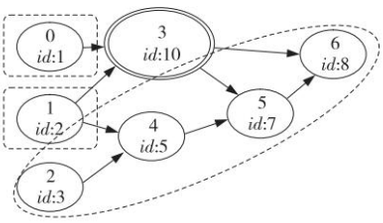


Figure8 MFFCpartition

Figure 8 MFFC division

Its thread number accesses different addresses to obtain input in different cycles.

The above process is performed simultaneously, so the same kernel function

The simulation task is two-dimensionally parallel.

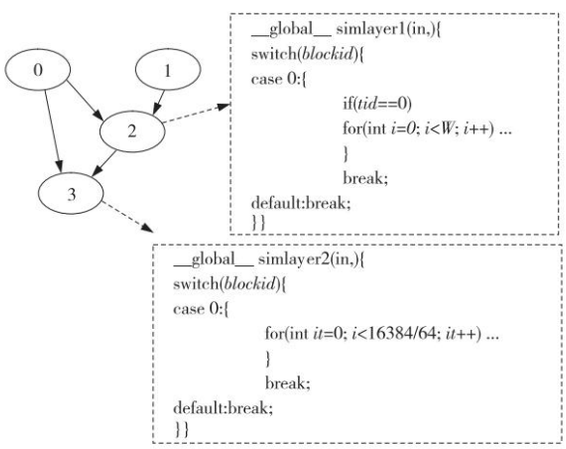


Figure9 Schedulingcodewithinalevel

Figure 9: Scheduling code within a hierarchy

However, as mentioned in Section 2.1, the maximum number of threads in a block

The number cannot exceed 1024. In actual situations, this maximum limit may be

Therefore, when the number of parallel cycles is greater than the number of blocks set

When the number of threads is greater than 1, group simulation will be performed. As shown in Figure 9,

When the MFFC partition is internally divided into blocks containing 64 parallel threads,

When the number of simulation cycles for each group is 16384 (2 14), the simulation

It is planned to be completed in 256 cycles, that is, each block is parallel and the block

The internal threads complete 256 simulation cycles in parallel.

The simulation is performed by an activity in the block.

The thread is implemented by looping 16384 times.

Simultaneous simulation of basic simulation units in layers, and MFFC partitions

Simultaneous simulation of multiple consecutive cycles.

Obviously, the simulation needs to wait for the input data to be transferred to the GPU.

The transmission of output data to the CPU needs to wait for all modules to complete.

The same as above is done, but the input and output data are copied between different groups.

Therefore, as shown in Figure 10, this paper uses

Three CUDA streams are used to schedule simulation and data transfer.

Overlapping of data copies in two directions is achieved.

5 Experiments and results analysis

This paper uses PyRTL to implement RTL for OR1200 [25]

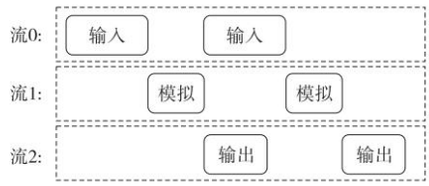


Figure10 Schedulingofusestream

Figure 10 Using flow scheduling

Description, then the 12 submodules and the entire microprocessor

The CPU randomly generates 131072 (2 17) cycles of input

The simulation method in this paper is evaluated by using the simulation results.

At the same time, this paper also uses PyRTL for FFT (Fast Fourier Transform)

[26], RocketCore [27], TLSPI [28] and

TLT2C [28] performed RTL description, also using random generation

The input method is similar to one of the most advanced open source simulators currently available -

ESSENT was compared. The number of simulation cycles increases in sequence.

The addition reached 2 21, 2 22 and 2 24.

This paper uses the following simulator configuration in the experiment and compares

Simulation time cost:

1) BaseLine: The traditional destruction method mentioned in this article

GPU-based simulator.

2)ESSENT: ES with all optimization options

SENT simulator.

3) GPUSim: This paper proposes a loop expansion based on

A simulator for GPUs.

The GPU operating environment is as follows:

1) Operating system: Ubuntu 18.04.1X86 64-bit.

2) Graphics card model: Tesla-V10032GiB.

The experimental results of OR1200 microprocessor design are shown in Table 1.

Table1 Executiontime& GPUSim's

speedupsoverBaseLineon2 17cycles

Table 1 GPUSim and BaseLine simulation 2^{17} The time taken for each cycle and the speedup ratio

test Use Cases	node quantity	whether With ring	GPUSim/s (-O3)	BaseLine/s (-O3)	Speedup
go	654	None	0.011469	1.229154	107.1
genpc	32	is	0.035963	0.993545	27.6
cfgr	267	None	0.006499	0.708643	109.0
ctrl	2106	None	0.019422	1.589940	81.8
expect	956	is	0.334450	1.281243	3.8
freeze	76		0.027681	0.598541	21.6
if	126		0.019843	0.574560	28.9
oper	145		0.015172	0.526842	34.7
sprs	1512	is	0.054054	1.529784	28.3
lsu	730	is	0.034788	1.142487	32.8
wb	53	None	0.003711	0.307451	82.8
rf	5794	is	1.882997	2.571856	1.3
cpu	13047	is	7.395821	9.171025	1.2

The number of nodes in the second column of Table 1 lists the number of LogicNets generated after each test case is modeled using PyRTL; the third column lists whether there is a loop in each test case; it indicates the number of parallel threads on the GPU. time

Summarizing the speedup ratio data shown in the last column of Table 1, we can draw the following conclusions: Using loop expansion for parallel simulation can achieve 1.2~107.1 times speedup compared to single-cycle simulation. The main factor affecting the speedup ratio is the existence of loops in the design. When there is no loop in the design, two-dimensional full parallel simulation can be achieved, which can achieve an acceleration effect of more than 81 times. When there is a loop in the design, although the speedup ratio is only 1.2 times at the minimum and 34.7 times at the maximum due to the different sizes of the loops, the acceleration effect can still be seen even when the loop size reaches the maximum, indicating that using loop expansion to make the rest of the parallel is still an effective

acceleration method. The experimental results of the remaining experimental circuits are shown in Table 2. The number of simulation cycles in Table 2 is the maximum number of cycles that can be simulated for each test case. Since the simulator will transfer all initial inputs to the GPU, the number of initial inputs will limit the maximum number of cycles that can be simulated for the test case. According to the data in the last column of Table 2, compared with ESSENT, the simulator proposed in this paper can achieve 2.2~14.0 times speedup, which further illustrates the feasibility of the acceleration method proposed in this paper.

Table 2 Execution time & GPUSim's speedups over ESSENT on 2 21, 2 22 and 2 24 cycles

Table 2 GPUSim and ESSENT simulation 2 21, 2 22 and 2^{24} The time taken for each cycle and the speedup ratio

Test Cases	Simulation cycles	Number of nodes	Is there a ring	GPUSim/s (+64)	ESSENTIAL / s	Speedup
FFT 2	2^{24}	15605	Yes	47.4	661	14.0
Rocket 2	2^{21}	9537	is	100.0	221	2.2
TLSPI 2	2^{22}	2798	Yes	13.7	100	7.3
TLT2C 2	2^{22}	1953	Yes	9.9	78	7.9

6 Conclusion

This paper introduces a new RTL simulation acceleration method on GPU based on loop expansion for HCL. This work is the first method for HCL to use GPU to accelerate RTL simulation. The partitioning technology based on loop expansion in this method enables RTL simulation to maximize the parallel computing potential of GPU. For further acceleration, this paper uses CUDA streams for data copying and kernel function scheduling, which can execute data copy in two directions at the same time. Experimental results show that compared with the single-cycle simulation method using loop breaking, the simulation method using multi-cycle parallel simulation using loop expansion achieves 1.2~34.7 times acceleration in loop circuits. Compared with ES-SENT, one of the most advanced open source simulators, the simulation method in this paper can also achieve a 2.2~14.0 times acceleration effect. However,

the existence of the loop has a great impact on the simulation speed. Therefore, the next research will focus on the method of optimizing the simulation in the loop. Since the nodes in the loop are affected by feedback, only a single cycle can be simulated. Therefore, the work of the researchers mentioned above on exploring parallelism is also of great reference value for optimizing the simulation in the loop.

References:

[1] JOHNM LK.Agilehardwaredesign[J].IEEE Micro,2020,40(4):4-5.

[2] BACHRACHJ,VO H,RICHARDSB,etal.Chisel: ConstructinghardwareinaScalaembeddedlanguage [C]ÿDAC Design Automation Conference2012, 2012:1212-1221.

[3] CLOWJ, TZIMPRAGOSG, DANGWALD, etal.A pythonicapproachforrapidhardwareprototypingand instrumentation[C]ÿ201727thInternationalConfer-enceonFieldProgrammableLogicandApplications, 2017:1-7.

[4] WANGLT,HOOVER N E,PORTER E H,etal. SSIM:Asoftwarelevelizedcompiled-codesimulator [C]ÿ24th ACM/IEEE Design AutomationConfer-ence,1987:2-8.

[5] WANGZC,MAURER P M.LECSIM:Alevelized event-driven compiledlogicsimulation[C]ÿ27th ACM/IEEEDesignAutomationConfer-ence,1990: 491-496.

[6] SNYDER W.Verilator4.0:Opensimulationgoesmulti-threaded[EB/OL].[2018-10-22].https://veripool. org/papers/Verilator_v4_Multithreaded_ Or-Conf2018.pdf.

[7] BEAMERS.AcaseforacceleratingsoftwareRTL simulation[J].IEEE Micro,2020,40(4):112-119.

[8] BEAMERS,DonofrioD.Efficientlyexploitinglow activityfactorstoaccelerate RTLsimulation[C]ÿ 2020 57th ACM/EDAC/IEEE Design Automation Conference,2020:1-6.

[9] CHATTERJEED,DEORIO A,BERTACCO V. Event-driven gate-level simulation with GP-GPUs [C]ÿ200946thACM/IEEEDesignAutomationCon-ference,2009:557-562.

[10] ZHANGYX,WEITC,KAIY W,etal.Logicsim-ulationaccelerationbasedonGPU[C]ÿProceedings of the 18th International Conference Mixed Design of Integrated Circuits and Systems, 2011: 608-613.

[11] ZHU Y H,WANGB,DENG YD.Massivelyparal-
lellogicsimulationwithGPUs[J].ACM Transac-
tionsonDesignAutomationofElectronicSystems,
2011,16(3):ArticleNo.:29.
[12] CHANDYK M,HOLMES V,MISRA J.Distrib-utedsimulation
ofnetworks[J].Computer Net-
works,1979,3(2):105-113.
[13] Simulationofpacketcommunicationsarchitecture
computer system[EB/OL].[2023-01-12].https://
dspace.mit.edu/handle/1721.1/149478.
[14] HOLSTS,IMHOF ME,WUNDERLICH H J.
High-throughputlogictiming simulation on GP-GPUs[J].ACM
TransactionsonDesignAutomation
ofElectronicSystems,2015,20(3):ArticleNo.:37.
[15] SEN A,AKSANLIB,BOZKURT M,etal.Parallel
cyclebasedlogicsimulationusinggraphicsproces-
singunits[C]ÿ20109thInternationalSymposium
onParallelandDistributedComputing,2010:71-78.
[16] CHATTERJEED,DEORIO A,BERTACCO V.
Gate-levelsimulation with GPU computing[J].
ACM TransactionsonDesign AutomationofElec-
tronicSystems,2011,16(3):ArticleNo.:26.
[17] ZHANG Y Q,REN H X,KELLERB,etal.Prob-
lemC:GPUacceleratedlogicre-simulation:(invited
talk)[C]ÿ2020IEEE/ACM InternationalConfer-
enceonComputerAidedDesign,2020:1-4.
[18] ZHANGY Q, REN HX, SRIDHARAN A, et al.
GATSPI:GPUacceleratedgate-levelsimulationfor
powerimprovement[C]ÿProceedingsofthe59th
ACM/IEEEDesignAutomationConference,2022:
1231-1236.
[19] ZENGC,YANG F,ZENG X.Acceleratelogicre-simulationon GPU
viagate/eventparallelism and
statecompression[C]ÿ2021IEEE/ACM Interna-
tionalConferenceonComputerAidedDesign,2021:
1-8.
[20] QIAN H,DENG YD.AcceleratingRTLsimulation
withGPUs[C]ÿProceedingsoftheIEEE/ACMIn-
ternationalConferenceonComputer-AidedDesign,
2011:687-693.
[21] LINDL,REN H X,ZHANGYQ,etal.FromRTL
toCUDA:A GPUaccelerationflowforRTLsimu-
lationwithbatchstimulus[C]ÿProceedingsofthe
51stInternational Conference on Parallel Proces-
sing,2022:1-12.
[22] ZHANGY Q, REN HX, KHAILANYB.
nities for RTLandgatelevelsimulationusingGPUs
(invitedtalk)[C]ÿ2020IEEE/ACM International
ConferenceonComputerAidedDesign,2020:1-5.
[23] dgl.ai[EB/OL].[2022-12-12].https://www.dgl.
eat.
[24] CLARKEE M, GRUMBERG O, PELED D A.
Modelchecking[M].2nded.Massachusetts:The
MIT Press, 2000.
[25] OR1200[CP/OL].[2023-01-10].https://github.
com/openrisc/or1200.
[26] FFT[CP/OL].[2023-06-13].https://github.com/
ucb-art/fft.
[27] RocketCore [CP/OL].[2023-06-13].https://
github.com/freechipsproject/rocket-chip.
[28] sifive-blocks [CP/OL].[2023-06-13].https://
github.com/sifive/sifive-blocks.

About the Author:

Tian Qian (1999-), female, from Cangzhou, Hebei, master's degree Student, research direction is electronic design automation. E-mail: tx0913wy@163.com

TIANXi,bornin1999,MScandidate, herresearchinterestincludeselectronicde-sign automation (EDA).

Li Tun (1974-), male, from Chenzhou, Hunan, PhD, Professor, Doctoral Supervisor, CCF Member (14884S), Research His research direction is electronic design automation. E-mail: tunli@nudt.edu.cn

LItun, born 1974, PhD, professor, PhDsupervisor,CCF member(14884S),hisresearchinter-estincludeselectronicdesignautomation(EDA).

Cheng Yue (1997-), female, from Yueyang, Hunan, master's degree Student, research direction is electronic design automation. E-mail: chengyue@nudt.edu.cn

CHENG Yue, born in 1997, MScandi-date,herresearchinterestincludeselec-tronicdesignautomation(EDA).

Pi Yan (2000-), male, from Xiangtan, Hunan, Master Student, research direction is electronic design automation. E-mail: piyan@nudt.edu.cn

PIYan,bornin2000,MScandidate, hisresearchinterestincludeselectronicde-sign automation (EDA).

Zou Hongji (1999-), male, from Xiaochang, Hubei, PhD Master's degree student, research direction is electronic design automation. mail:1359261542@qq.com

ZOU Hongji, born in 1999, PhD candidate date,hisresearchinterestincludeselectron-icdesignautomation(EDA).