Improving Parallelism in System Level Models by Assessing PDES Performance

Emad Malekzadeh Arasteh
Center for Embedded and Cyber-physical Systems
University of California, Irvine
Irvine, California, USA, 92697
emalekza@uci.edu

Rainer Dömer
Center for Embedded and Cyber-physical Systems
University of California, Irvine
Irvine, California, USA, 92697
doemer@uci.edu

***Abstract*—For effective embedded system design, transaction level modeling (TLM) must explicitly expose any available parallelism in the application. Traditional TLM in SystemC utilizes channels for communication and synchronization between concurrent modules, whereas modern TLM-2.0 emphasizes address-accurate communication via explicit interconnect and memories. In both modeling styles, the choice of synchronization mechanisms has a significant impact on the available parallelism in the model which can be exploited by parallel discrete event simulation (PDES).**

In this work, we propose and analyze a set of non-invasive standard-compliant modeling techniques to increase parallelism in IEEE SystemC TLM-1 and TLM-2.0 models. We measure the performance of aggressive out-of-order PDES in the Recoding Infrastructure for SystemC (RISC) and analyze the parallelism in the models. Our case study on six modeling styles of a state-of-art deep neural network (DNN), namely the GoogLeNet image classification algorithm, demonstrates the impact of varying synchronization mechanisms with simulator run time reduced by 38% compared to a synchronous parallel reference model on a 16-core host machine. Our study also suggests that increased parallel simulation performance indicates better models with higher amounts of parallelism exposed.

***Index Terms*—system modeling, model parallelism, SystemC, transaction-level modeling, neural networks, parallel simulation**

I. Introduction

Transaction level modeling (TLM) explicitly exposes inherent parallelism in the application by modeling concurrency, hierarchy, synchronization and timing. TLM guidelines use different methods to model communication of concurrent modules in the design. TLM-1 focuses on modeling communication using channels and TLM-2.0 focuses on modeling address-accurate communication using memory-mapped buses. The choice of synchronization and communication mechanisms in TLM models affect the available level of parallelism. Parallel Discrete Event Simulation (PDES) is an attractive approach to measure parallelism of TLM models and compare simulation performance between models using different parallelization techniques.

Exponential growth of computational requirements of new emerging applications such as deep learning, puts an extra demand on finding parallelism opportunities and simulation performance. To this end, fast and yet standard-compliant simulation of design candidates will enable rapid design space exploration and hence, shorter time to market.

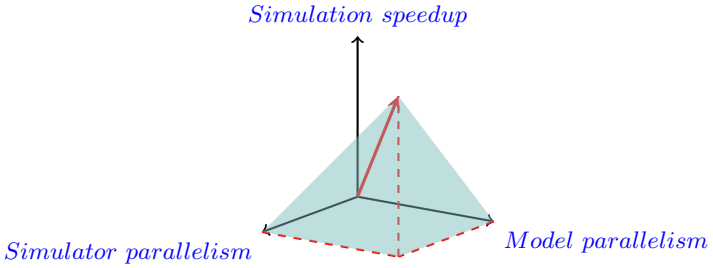

Fig. 1: Simulator parallelism, model parallelism and simulation speedup forms a 3-dimensional space

In this paper, we propose a set of IEEE standard-compliant modeling techniques to increase available parallelism in SystemC TLM-1 and TLM-2.0 models for parallel discrete event simulation. As shown in Figure 1, we illustrate simulator parallelism, model parallelism and simulation speedup in a 3-dimensional space. As the red arrow indicates, both higher model parallelism and simulator parallelism achieve the maximum simulation speedup. Moreover, by increasing model parallelism opportunities in one dimension, the simulator can better leverage its parallelization capabilities for the maximum simulation speedup. In particular, we demonstrate our proposed techniques on SystemC TLM models of a DNN using out-of-order parallel simulation.

Our key contributions are as follows:

(1) A systematic analysis of parallelism opportunities in SystemC TLM-1 and TLM-2.0 models of a representative DNN (GoogLeNet) for parallel simulation

(2) A proposal of less restrictive communication mechanisms and transaction types for enhanced parallelism with out-of-order parallel simulation of TLM-1 and TLM-2.0 models

(3) Experimental results that demonstrate the improved parallelism in a given reference model [1] with simulator run time reduced by 38%

978-1-6654-1825-6/21/$31.00 © 2021 IEEE

II. Background

In this section, we provide a brief background on Parallel Discrete Event Simulation (PDES) of SystemC models and in particular out-of-order PDES implemented in the Recoding Infrastructure of SystemC (RISC). Furthermore, we introduce the application driver used in this study.

A. Parallel SystemC Simulation

Earlier works on Parallel Discrete Event Simulation (PDES) such as [2] focused on distributed simulation hosts. Fujimoto presented the first initial work on parallel and distributed hosts [7]. [5] presented the first synchronous parallel SystemC kernel. Out-of-Order parallel SystemC simulation was first proposed in [3] to increase the simulation speed. [10] presented the first implementation of OoO PDES. [4] extended the RISC compiler with Socket Call Path technique to support safe parallel simulation of TLM-2.0 models.

The Out-of-Order PDES (OoO PDES) approach proposes to allow threads in different cycles to run in parallel if those threads do not have potential data or event conflicts [3]. OoO PDES maximizes multi-core CPU utilization by localizing global simulation time for each thread and performing conservative analysis of potential conflicts among the active threads. This approach is realized in Recoding Infrastructure for SystemC (RISC) that performs parallel SystemC simulation in maximum compliance with the IEEE standard semantics using a dedicated SystemC compiler that automatically analyzes existing conflicts in the model [9].

Despite the fact that RISC maximizes the number of threads to run in parallel, we optimize TLM models such that they exhibit further parallelism opportunities so that RISC achieves even faster simulation.

B. Deep Learning and Convolutional Neural Network

Deep Learning (DL) is a known technique in machine learning to extract useful features from input data, perform data transformations, and arrive at a final meaningful representation. One of the main application areas of DL is visual recognition and in particular, image classification which is the problem of assigning a descriptive label to an input image from a fixed set of categories.

A convolutional neural network (CNN) mainly consists of alternating convolution layers and pooling (sub-sampling) layers. Choosing a state-of-the-art deep CNN for TLM modeling enables the means to investigate parallelism opportunities with a real-world industry-strength application. Therefore, we select GoogLeNet [12], a deep CNN for image classification and detection, and start with a reference model in SystemC [1].

GoogLeNet is 22 layers deep when counting only layers with parameters. The overall number of layers (independent building blocks) is 142 distinct layers. The main constituent layers are convolution, pooling, concatenation and classifier. GoogLeNet includes two auxiliary classifiers that are used during training to combat the vanishing gradient problem. Further details on the GoogLeNet structure can be found in [1] and [12].

III. Related work

Static analysis of SystemC models and TLM modeling techniques for parallel SystemC simulation has been studied in other works. The SystemC-clang framework [8] analyzes SystemC models at register-transfer level and transaction-level with support for some TLM 2.0 constructs. Authors in [13], [14], and [15] propose modified parallel SystemC simulation kernels that require users to manually translate their sequential models into safe parallel models. [11] provides SystemC designers with a set of primitives to manually parallelize SystemC tasks for loosely-timed models. These techniques require the designer to manually instrument the model for safe parallel simulation. In contrast to prior works, our approach leverages from the complete automatic parallelization in the RISC to increase PDES simulation performance in a safe and standard-compliant fashion.

Automatic generation of a set of RTL primitives by analyzing CNN architecture and parameters to be used on FPGA has been carried out in [16]. To the best of our knowledge, there is no similar work on improving the parallelism in SystemC TLM for CNN.

IV. Parallelism in TLM-1 modeling

Following the distinction between simulator parallelism and model parallelism introduced in Figure 1, we propose alternative channel constructs to increase parallelism opportunities. We also analyze how the number of buffers inside channels can increase parallel simulation performance.

A. Channel type

Starting from the model proposed in [1], we improve the communication and synchronization mechanism in this work. To this end, we propose three channel types according to the channel synchronization mechanism:

1) *Blocking channel*: In a blocking channel, synchronization is handled using a set of two `wait` statements in read and write access functions.
2) *Non-blocking channel*: In a non-blocking channel, the write access function does not block and synchronization is handled using only one `wait` statement in the read access function.
3) *SystemC FIFO channel*: This channel is built on the predefined primitive channel `sc_fifo` with default read and write member functions which use the `request_update` mechanism.

In a blocking channel two `sc_event`s ensure synchronization between each consumer and producer. In a non-blocking channel, we design a synchronization scheme between producer and consumer that uses only one `wait` statement and one `sc_event`. Lastly, we design SystemC channels that do not require any calls to the `wait` construct. The improved communication techniques increase the potential that an out-of-order PDES simulator schedules threads more aggressively.

B. *Buffering scheme*

The TLM-1 model of a data processing application can be considered as a graph data structure with modules as nodes and channels as edges connecting neighboring modules. Each module continuously fetches data from its input channel(s), processes the data and writes its result(s) into output channel(s). These data processing modules often form a pipeline structure that execute in parallel while buffers in channels hold intermediate results between pipeline stages. The more buffers exist in the channels, the more possibilities there are for pipelining of data in the graph. This gives a parallel simulator more freedom to schedule even more parallel threads at the same delta cycle.

In particular, having only a single buffer inside blocking channels, modules can only process data in every other delta cycle. However, with double buffers inside channels, a producer can write to the back buffer while a consumer can read from the front buffer. This results in more active threads that perform their tasks in fewer delta cycles. This increase in the level of parallelism gives the parallel simulator more opportunities to aggressively schedule threads and minimize simulation run time.

Figure 2 illustrates the inception module, the main building block of GoogLeNet. The inception module forms an unbalanced graph structure with four parallel tracks, each running a different workload. Note that the four parallel tracks in each inception module contain (2, 4, 4, 3) modules to process, respectively. In the absence of a balanced graph topology, the number of buffers in channels should address the imbalance to enable the maximum number of modules to run in parallel.

A TLM-1 modeling diagram of the inception module with double buffering scheme is shown in Figure 3. As shown, modules read/write data from/to channels via their input port(s) and output port(s). Note that the output channels for **relu_1x1** and **relu_pool_proj** keep 4 and 3 buffers, respectively, to compensate for the unbalanced graph structure.

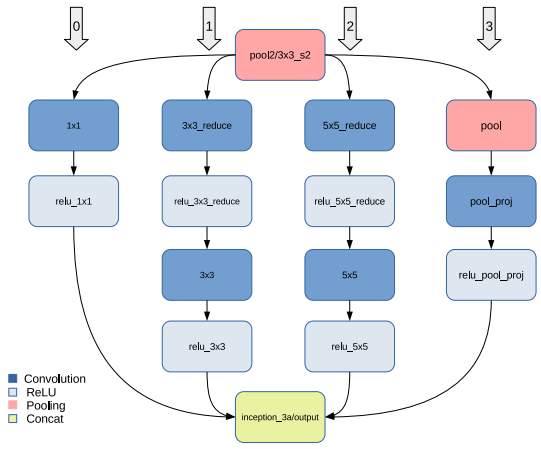

Fig. 2: Inception module in GoogLeNet

In the case of non-blocking channels, the `write` access method does not incur any `wait` statement. Therefore, the

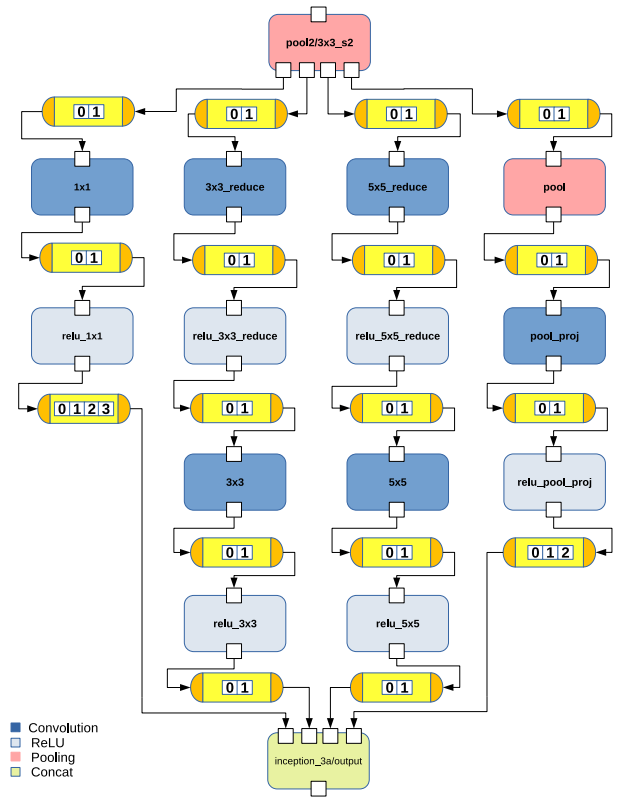

Fig. 3: TLM-1 model diagram of inception module in GoogLeNet

number of buffers in non-blocking channels needs to be increased to avoid any buffer overflow.

Overall GoogLeNet forms a graph with a depth of 62 layers. In the worst case scenario, all producer layers in each level of the graph write to the channel before consumer layers read the data. To dimension the channel sizes for this worst case scenario, non-blocking channels should have space for the maximum depth of the graph plus one for the stimulus module, namely 63 buffer elements.

V. Parallelism in TLM-2.0 Modeling

While TLM-1 gives early feedback on parallelism and local communication, it is not specifically intended for bus modeling, interoperability and architectural exploration. SystemC TLM-2.0 introduces generic payload and core transport interfaces for the abstract modeling of memory-mapped buses. However, the notion of channels from TLM-1 has disappeared from TLM-2.0 modeling and each module instead uses pointers to access memory locations in other modules. The lack of an encapsulating channel construct allows simulation threads to directly access data of other modules, making synchronization of such accesses a difficult task for parallel simulators in a standard-compliant fashion [4] and has been identified as an obstacle for safe and fast parallel simulation [6]. On the other hand, TLM-2.0 models feature address-accurate memories.

In this section, we introduce a feed-forward events mechanism for TLM-2.0 modeling of data processing applications such as DNN that can be used for synchronous parallel simulation. Later, we propose the back-pressure events mechanism devised for safe out-of-order parallel simulation.

A. Feed-forward events mechanism

In TLM-2.0 modeling, a socket should be instantiated within each initiator and each target for every transaction level connection. Therefore, module input and output ports in TLM-1 models are replaced with an initiator socket. The generic payload captures the information to pass with each bus transaction between initiator and target. The initiator module instantiates the generic payload transaction object and sets its attributes before passing a reference to this object to a target module via its transport interface.

In our proposed model, we equip each module with an initiator socket connected to a target socket on a shared TLM-2.0 model of a memory. Each module in this model has a dedicated address space inside the memory to read and write its data. Figure 4a shows the connections of initiator sockets of the first convolution and ReLU layers in GoogLeNet to target sockets of shared TLM-2.0 memory.

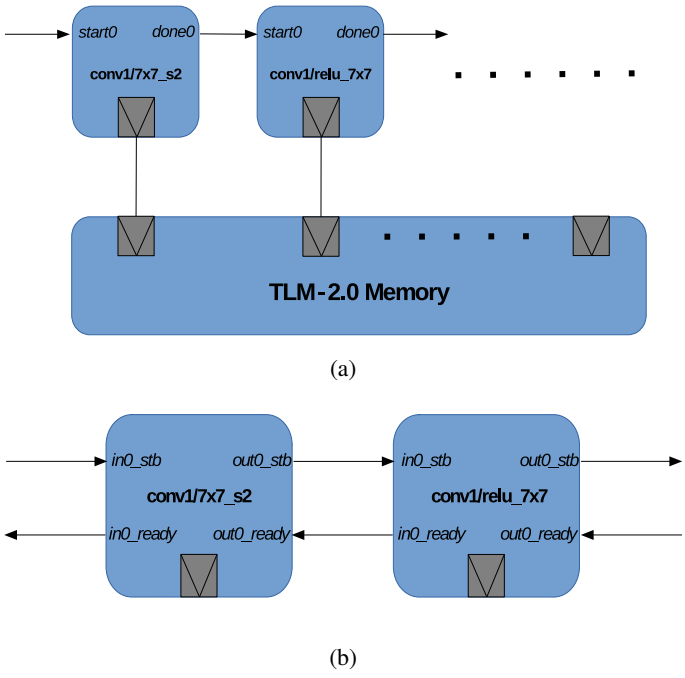

Fig. 4: (a) TLM-2.0 feed-forward model connections (b) TLM-2.0 back-pressure events connections

To minimize the memory footprint, adjacent modules share a common buffer inside the memory. For example, the output of the `conv1_7x7_s2` module is written to a buffer that the `conv1/relu_7x7` module uses for its input. To guarantee correct synchronization and avoid data race conditions, the producer notifies the consumer via an event once it completes its write transaction. On the other end, the consumer waits for the event notification from the producer before it can start its read transaction. Therefore, each module contains one `sc_event` for each input (`start`) and each output (`done`). For example, as seen in Figure 4a, the `start` event of `conv1/relu_7x7` is connected to `done` event of `conv1_7x7_s2`.

In the feed-forward model with only a single buffer between neighboring modules, modules only accept and process data every other delta cycle. By increasing the number of buffers between modules, modules can instead process data every delta cycle. This gives the parallel simulator the opportunity to schedule more threads in each delta cycle, utilizing available parallelism in the processor for minimizing the simulation run time. Moreover, such a model achieves its maximum theoretical throughput, generating output every delta cycle.

As already illustrated in Figure 2, each inception module in GoogLeNet has four parallel tracks and tracks containing (2, 4, 4, 3) modules to process, respectively. Since events occur at precise points in simulation time, the proposed TLM-2.0 model must guarantee that inputs from different tracks arrive at the exact same cycle to the output of the inception module, namely `inception_3a/output`. To guarantee correct event synchronizations, delay elements must be inserted in those tracks with less modules to form a balanced graph structure. This means 2 delay elements in track 0 and 1 delay element in track 3 are required. Furthermore, the output of the last modules in tracks with less modules require extra buffers to store results generated during those delay cycles. This means for supporting double buffering, the last module in track 0, `relu_1x1`, and the last module in track 3, `relu_pool_proj`, require 4 and 3 output buffers, respectively. These extra buffers ensure a continuous stream of data in every delta cycle into the design, increasing model parallelism and maximizing model throughput.

B. Back-pressure events mechanism

The model without back-pressure mechanism is not safe for aggressive out-of-order scheduling. Only a conservative in-order scheduling approach will execute the feed-forward TLM-2.0 model correctly due to the missing back-pressure mechanism. Therefore, we devise a back-pressure events mechanism to safely execute the TLM-2.0 model in the aggressive OoO parallel simulation for maximum speedup.

As event connections for the first convolution and ReLU layers in GoogLeNet are depicted in Figure 4b, each module has a set of two `sc_events` for each input and output. The `stb` event is always notified once a module has valid data inside the memory to be read and the `ready` event is always notified once a module is ready to read a new data. By connecting events between all subsequent modules, the model forms a robust back-pressure mechanism that safely controls the flow of data inside the pipeline.

Support for the back-pressure events mechanism should be extended to all neighboring modules in the TLM-2.0 model. Furthermore, the double-buffering scheme guarantees a continuous stream of data inside the design pipeline, maximizing

model parallelism and model throughput with the minimum number of buffers in the memory.

VI. Parallelism Direction

To demonstrate degrees of freedom for parallel simulators to find parallelism opportunities in TLM-1 and TLM-2.0 models, we create an XY chart with communication mechanism and buffering scheme on x- and y-axes, respectively. As depicted in Figure 5, we map the number of buffers on the x-axis and communication mechanism on the y-axis. On the x-axis, *min* refers to a single buffer, *mul* to double buffer taking into account that certain layers requires multiple buffers due to the imbalanced graph structure, and *max* refers to the total depth of TLM-1 model graph. On the y-axis, we map communication mechanisms from the most restrictive type for parallel simulation, namely, *TLM-1 blocking*, to the least restrictive type, namely, *TLM-2.0 back-pressure*. Increasing the number of buffers and utilizing communication with less restrictive synchronization mechanisms creates more freedom for out-of-order simulators to schedule threads in different delta cycles. This maximizes multi-core utilization and hence results in shorter simulator run time.

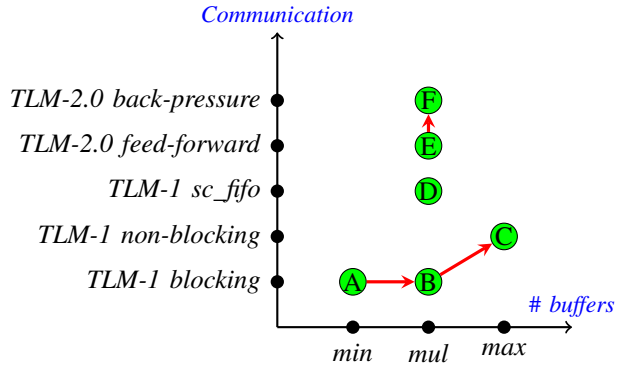

Fig. 5: Communication mechanism versus number of available buffers in TLM models

Given the proposed communication mechanisms and buffering schemes, we have designed a set of TLM-1 and TLM-2.0 models for GoogLeNet. As marked in Figure 5, the reference Model **A** [1] uses blocking channels with only a single buffer in channels. We designed Model **B** with a double-buffering scheme with blocking-channels. In Model **C**, we replaced blocking channels with non-blocking channels with buffer size of 63 elements, the total depth of the graph. We designed Model **D** using `sc_fifos` with a double-buffering scheme. Model **E** is a TLM-2.0 model that uses the feed-forward events mechanism as an inter-module communication and modules have double-buffers inside a shared memory. Finally, we designed Model **F** using our back-pressure mechanism to guarantee safe communication between modules for aggressive out-of-order scheduling with double buffers for maximum parallelism and maximum throughput. Table I summarizes the properties of all designed TLM-1 and TLM-2.0 models.

TABLE I: TLM models summary

Model name	Standard	Communication	Buffers
Model A [1]	TLM-1	Blocking channels	Single buffer
Model B	TLM-1	Blocking channels	Double buffers
Model C	TLM-1	Non-blocking channels	Buffer size of 63
Model D	TLM-1	SystemC FIFO channels	Double buffers
Model E	TLM-2.0	Feed-forward	Double buffers
Model F	TLM-2.0	Back-pressure	Double buffers

TABLE II: Platform specification

Platform name	16-core (Phi)	32-core (Phi HT)
OS	CentOS 6.10	CentOS 6.10
CPU Model name	Intel Xeon E5-2680	Intel Xeon E5-2680
CPU frequency	2.7 GHz	2.7 GHz
#cores	8	8
#processors	2	2
#threads per cores	1	2

Since each SystemC module has specific attributes based on its layer type and its corresponding TLM modeling style, writing module definitions by hand is a tremendously error-prone task. Furthermore, interconnecting all modules at the top level using either queues or events is a tedious task. Therefore, we have extended the generator tool from Model A [1] to automatically generate all the other TLM-1 and TLM-2.0 models based on modeling style, communication type and buffer architecture. In the case of TLM-2.0 models, the Python 3 generator automatically produces an address map file based on buffer architecture and supports memory address generation for multiple buffers for any layer in the network.

VII. Experimental Measurements and Results

Parallelism opportunities introduced in transaction-level SystemC models can be quantified and measured using a SystemC parallel simulator. To exploit the available parallelism in our TLM-1 and TLM-2.0 models of GoogLeNet, we describe our extensive measurement results using RISC and provide valuable insights gained from analyzing the results.

A. Simulation setup

We use a 16-core host computer platform with hyper-threading technology (HTT) to benchmark the simulations. The specifications of the platform are shown in Table II. To have reproducible experiments, the Linux CPU scaling governor is set to 'performance' to run all cores at the maximum frequency, and file I/O operations, i.e. *cout*, are minimized.

B. Simulation Results

For benchmarking, we measure simulator run time using Linux */usr/bin/time* under CentOS 7. Measurements are reported for the sequential SystemC simulation using Accellera SystemC and for the parallel simulations using RISC simulator V0.6.2 in three modes: synchronous (SYN), non-prediction

(NPD) and out-of-order (OOO) parallelism. For reliability of the results, each measurement is performed three times. Later, if the distance of each recorded value from its median is greater than $\pm 2\%$, that entire measurement is ignored.

We analyze the measurement results obtained from the simulations of six TLM models. We create various heat map tables to identify the relevant results regarding parallelism in transaction types and transaction level modeling as follows:

1) Less restrictive transaction types enable higher parallelism: Table III shows the elapsed time of the models in SYN, NPD and OOO simulation modes using RISC V0.6.2.

	Phi		
Elapsed time	SYN	NPD	OOO
Model A	266.66	194.77	193.94
Model B	229.04	193.63	193.08
Model C	231.79	220.32	200.29
Model D	197.02	197.93	194.90
Model E	198.31		
Model F	198.66	199.31	170.44

	Phi HT		
Elapsed time	SYN	NPD	OOO
Model A	276.04	198.40	197.17
Model B	237.69	196.73	197.47
Model C	235.12	224.39	202.90
Model D	201.13	195.35	195.40
Model E	203.29		
Model F	203.17	204.89	170.68

TABLE III: Measurements of elapsed time for parallel simulations (color scale red-to-green means slow-to-fast)

Considering the 16-core machine (phi), model A uses blocking channels with a single buffer. The elapsed time of Model A for the SYN mode is the highest. Model B uses multiple buffers to increase the potential for pipelining. Model C removes wait statements in the write function to let the OOO scheduler schedule multiple threads together. Model D uses SystemC FIFOs to implement channels. SystemC FIFO forces synchronous simulation. Hence, the elapsed time of Model D for SYN, NPD and OOO are almost identical as reflected in the fourth row. Model E and Model F are TLM-2.0 models without any usage of primitive channels. As previously stated, model E is not safe for out-of-order parallel simulation, so elapsed time for NPD and OOO simulations are not reported for this model. As can be seen in the second to six rows, the elapsed time for SYN simulation mode decreases steadily. However, the aggressive OOO simulation exploits the maximum parallelism introduced in each model and reports the shortest elapsed time for model F. The exact same pattern applies to the other TLM-1 and TLM-2.0 models on machines with a higher number of logical cores.

Notably, our efforts on increasing the potential of parallelism in the models pay off with a significant simulation speedup. Comparing the synchronous reference TLM-1 model (276.04s with hyper-threading enabled) with the OOO simulation of the TLM-2.0 model with safe back-pressure (170.68s), shows the simulator run time reduced by 38%. Note that this applies despite the higher workload the TLM-2.0 models carry, as we will show in the next section.

2) Abstract TLM-1 models carry less workload than memory accurate TLM-2.0 models: Table IV shows the heat map table for elapsed time of all six models in sequential simulation mode. The last two rows for TLM-2.0 models indicate longer elapsed time than the first four rows for TLM-1 models. This can be explained due to the difference in number of memory copies in TLM-1 and TLM-2.0 models. TLM-1 models use shallow copy for storing/loading items in/from channels. However, TLM-2.0 models use two memory copies to read and write from/to the memory module. This distinction shows that the actual simulator workload for the TLM-2.0 models has increased in comparison to the TLM-1 models.

	Phi	Phi HT
Elapsed time	SEQ	SEQ
Model A	949.02	949.56
Model B	940.12	939.61
Model C	941.93	940.93
Model D	945.24	943.98
Model E	956.28	955.81
Model F	956.28	956.01

TABLE IV: Measurements of elapsed time for SEQ execution (color scale green-to-red means increasing workload)

3) Increased parallel simulation performance indicates better models with higher amount of parallelism exposed: Figure 6 illustrates simulation speedups in different parallel simulation modes on our 16-core host (phi). As shown, the maximum simulation speedup (5.6x) is achieved by model F in out-of-order (OoO) parallel simulation mode. This indicates model F has the highest level of parallelism available in comparison with other TLM models. Therefore, model F is the right design candidate for further model refinements and lower-level implementation.

VIII. Conclusion

In this work, the impact of synchronization and communication mechanisms on available parallelism in transaction level modeling (TLM) has been studied. We have demonstrated the impact of varying synchronization mechanisms on the exposed parallelism using six modeling styles of a state-of-art deep neural network (DNN), GoogLeNet. We further have quantified the improved parallelism in the improved SystemC TLM-1 and TLM-2.0 models by measuring the performance of aggressive out-of-order parallel simulation in the Recoding Infrastructure of SystemC (RISC). The experimental results show that our standard-compliant parallelization techniques result in a significantly increased simulation speed up to 5.6x on a 16-core host machine. Notably, the results support the hypothesis that higher speed in aggressive parallel simulation is a significant indicator of higher level of parallelism in design

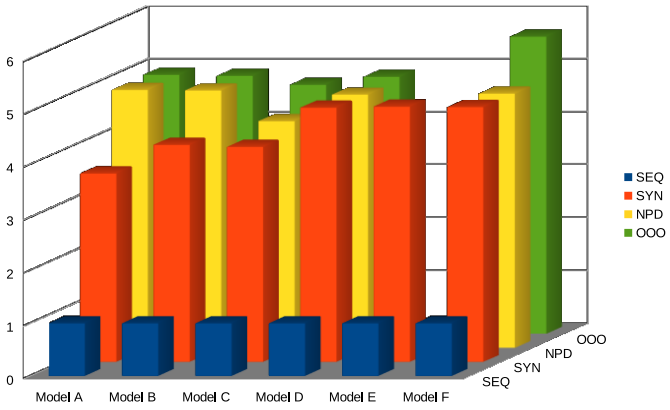

Fig. 6: Simulation speedup for different parallel simulation modes on a 16-core host

models which enables better implementation at later stages in the design flow.

While this work has focused on communication and synchronization aspects, we plan to expand the quantitative analysis of parallelism in future work to computation, explore timed models at lower abstraction, and include a wider range of DNN applications.

REFERENCES

[1] Emad Malekzadeh Arasteh and Rainer Dömer. An Untimed SystemC Model of GoogLeNet. *Proceedings of the International Embedded Systems Symposium*, 2019.

[2] K. M. Chandy and J. Misra. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Transactions on Software Engineering*, SE-5:440–452, 1979.

[3] Weiwei Chen, Xu Han, Che-Wei Chang, Guantao Liu, and Rainer Dömer. Out-of-Order Parallel Discrete Event Simulation for Transaction Level Models. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 33(12):1859–1872, December 2014.

[4] Zhongqi Cheng and Rainer Dömer. Analyzing variable entanglement for parallel simulation of systemc tlm-2.0 models. *ACM Trans. Embed. Comput. Syst.*, 18(5s), October 2019.

[5] Bastien Chopard, Philippe Combes, and Julien Zory. A Conservative Approach to SystemC Parallelization. In *International Conference on Computational Science (4)*, pages 653–660, 2006.

[6] Rainer Dömer. Seven obstacles in the way of standard-compliant parallel SystemC simulation. *IEEE Embededded Systems Letters*, 8(4):81–84, December 2016.

[7] Richard Fujimoto. Parallel Discrete Event Simulation. *Communications of the ACM*, 33(10):30–53, Oct 1990.

[8] A. Kaushik and H. D. Patel. Systemc-clang: An open-source framework for analyzing mixed-abstraction systemc models. In *Proceedings of the 2013 Forum on specification and Design Languages (FDL)*, pages 1–8, 2013.

[9] Guantao Liu, Tim Schmidt, Zhongqi Cheng, Daniel Mendoza, and Rainer Dömer. RISC Compiler and Simulator, Release V0.6.0: Out-of-Order Parallel Simulatable SystemC Subset. Technical Report CECS-TR-19-04, Center for Embedded and Cyber-physical Systems, University of California, Irvine, September 2019.

[10] Guantao Liu, Tim Schmidt, and Rainer Dömer. RISC Compiler and Simulator, Alpha Release V0.2.1: Out-of-Order Parallel Simulatable SystemC Subset. Technical Report CECS-TR-15-02, Center for Embedded and Cyber-physical Systems, University of California, Irvine, October 2015.

[11] Matthieu Moy. Parallel programming with systemc for loosely timed models: A non-intrusive approach. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '13, page 9–14, San Jose, CA, USA, 2013. EDA Consortium.

[12] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015*, pages 1–9, 2015.

[13] N. Ventroux and Tanguy Sassolas. A new parallel systemc kernel leveraging manycore architectures. *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 487–492, 2016.

[14] Jan Henrik Weinstock, Rainer Leupers, Gerd Ascheid, Dietmar Petras, and Andreas Hoffmann. Systemc-link: Parallel systemc simulation using time-decoupled segments. In *Proceedings of the 2016 Conference on Design, Automation and Test in Europe*, DATE '16, page 493–498, San Jose, CA, USA, 2016. EDA Consortium.

[15] Jan Henrik Weinstock, Christoph Schumacher, Rainer Leupers, Gerd Ascheid, and Laura Tosoratto. Time-decoupled parallel systemc simulation. In *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1–4, 2014.

[16] Yufei Ma, N. Suda, Yu Cao, J. Seo, and S. Vrudhula. Scalable and modularized rtl compilation of convolutional neural networks onto fpga. In *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–8, Aug 2016.