NoCeption: A Fast PPA Prediction Framework for Network-on-Chips Using Graph Neural Network

Fuping Li*†, Ying Wang*†§, Cheng Liu*†, Huawei Li*†‡, Xiaowei Li*†
*SKLCA, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
†University of Chinese Academy of Sciences, Beijing, China
‡Peng Cheng Laboratory, Shenzhen, China
§Zhejiang Laboratory, Hangzhou, China
{lifuping20s, wangying2009, liucheng, lihuawei, lxw}@ict.ac.cn

***Abstract*—Network-on-Chips (NoCs) have been viewed as a promising alternative to traditional on-chip communication architecture for the increasing number of IPs in modern chips. To support the vast design space exploration of application-specific NoC characteristics with arbitrary topologies, in this paper, we propose a fast estimation framework to predict power, performance, and area (PPA) of NoCs based on graph neural networks (GNNs). We present a general way of modeling the application and the NoC with user-defined parameters as an attributed graph, which can be learned by the GNN model. Experimental results show that on the unseen realistic applications, the proposed method achieves the accuracy of 97.36% on power estimation, 97.83% on area estimation, and improves the accuracy of the network-level and system-level performance predictor over the topology-constrained baseline method by 6.52% and 4.73% respectively.**

***Index Terms*—network-on-chip, PPA prediction, graph neural network**

I. Introduction

Network-on-Chip (NoC) has been widely adopted as the high-performance interconnect fabric to satisfy the increasing communication bandwidth demand of multiprocessor system-on-chips (MPSoCs) due to its better modularity and scalability than traditional buses.

To achieve better PPA for application-specific NoCs, a vast design space needs to be explored to obtain the optimized micro-architectural and circuit-level parameters such as topology and buffer depth, which is known as an NP-hard combinatorial problem [1]. Most of the existing NoC design automation frameworks, e.g. genetic algorithm-based method [2], need to iteratively evaluate the PPA of the generated intermediate NoC instances. As a result, fast and precise PPA prediction is critical. Though we can have precise PPA by evaluating the NoC system with synthesis and detailed simulation, the computation cost is unaffordable when it comes to thousands of optimization loops. Compared with synthesis and simulation, the model-based PPA estimation methods consisting of analytical model-based and machine learning-based solutions can save orders of magnitude simulation time with a reasonable amount of prediction losses.

To derive a mathematically extractable PPA model, the analytical model-based methods usually make a lot of assumptions, which limits their usage scenarios [3]. Besides, the established analytical models are often highly coupled with the specific NoC architectures. The poor generalizability requires designers to build new models for different NoC architectures and increases the overhead of chip design. In contrast, the popularity of machine learning-based methods solves these problems to a larger extent with model-less end-to-end prediction by fitting the target function with pre-labeled data samples, which helps decouple the predictor from the specific NoC implementation for removal of the model building phase. However, the applicability of prior predictors [3] [4] are seriously limited by conventional ML algorithms designed to analyze **Euclidean** data, which cannot accurately describe and capture features of irregular NoCs. Thus, they are restricted to specific regular network topologies such as mesh.

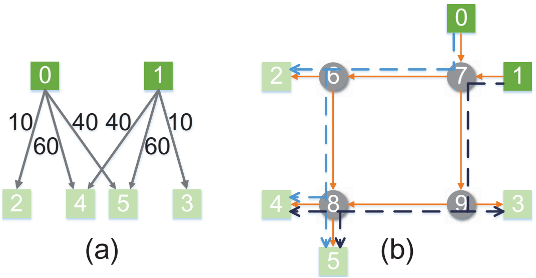

Fig. 1. (a) Example of the task graph of an MPSoC. The darker ones are master cores and the lighter ones are slave cores. (b) Example of an NoC topology graph. Squares are NIs which master cores and slave cores in (a) are connected to (with same number and color) and circles are routers. The dotted lines show the routing paths using (a) as task and the different colors indicate traffic flows coming from different masters.

To address these challenges, in this paper, we develop a GNN-based PPA prediction framework. Unlike traditional machine learning frameworks, GNNs are known powerful in dealing with **non-Euclidean** data structures, i.e. graphs. Therefore, with GNNs, we can model detailed NoC instances as attributed graphs that can be generalized to other NoC implementations easily. In general, this paper makes following contributions:

- We propose NoCeption, a flexible GNN-based framework to estimate power, performance, and area overhead for NoCs with arbitrary irregular topologies and variable design parameters. We design the network architecture and dataset generation flow to train the predictor.
- For the first time, we employ a general representation method to convert task graphs and NoCs to attributed graphs, which can extract vital information, especially

978-3-9819263-6-1/DATE22/©2022 EDAA

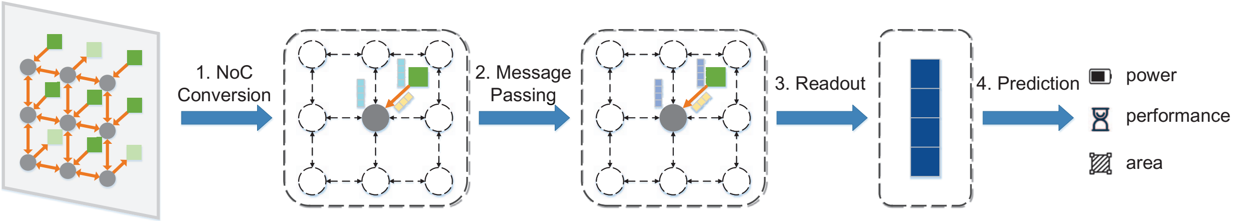

Fig. 2. Framework of NoCeption. Step 1 maps the NoC and its application to an attributed graph, where details of dotted portions are omitted; step 2 performs message passing to aggregate information from neighbors; step 3 generates a graph-level representation of the NoC system; step 4 predicts the PPA of the application-specific NoC.

router architectures and link congestions.

- To evaluate the framework proposed, we test NoCeption on silicon-proof commercial NoC IP [5] using both synthetic and real-world workloads. Experimental results show that the proposed method can estimate the PPA of NoCs accurately and achieves significant improvement over the topology-constrained baseline.

II. Background And Related Work

A. NoC Basics

A classic NoC consists of network interfaces (NIs) and routers. NIs are connected to heterogeneous cores and are responsible for converting standard protocols such as AXI [6] to internal packets of the network. Routers are used to multiplex multiple communication flows over the NoC interconnect. The topology is defined as the connection pattern of NIs and routers. The heterogeneous nature of MPSoCs requires irregular topologies and customized microarchitectures of NoCs, which have great impacts on PPA of NoCs [7] and can not be ignored. Common customization includes synchronizers for clock domain crossing support and variable buffer depth in routers for specific bandwidth demand.

B. PPA Prediction for NoC Design

In pre-silicon chip design, accurate and reliable PPA can be obtained through synthesis and detailed simulation taking tens of wall-clock minutes, which are unacceptable for fast design space exploration. To deal with it, many methods for fast PPA estimation have been proposed. These methods can be roughly divided into two categories: analytical model-based and machine learning-based.

The analytical model-based methods need to establish sophisticated mathematical models. In ORION 2.0 [8], the power and area are estimated by modeling NoCs at circuit level. To estimate performance, researchers model the NoC as a large queuing system and solve it with queuing theory. The model in [9] views channels of the router as M/M/1 queues, denoting exponentially distributed inter-arrival time, exponentially distributed service time, and one server respectively. Some work extends traffic inter-arrival time and service time to following General distribution [10].

Compared with all these methods based on analytical models, machine learning-based algorithms do not need to consider the low-level implementation, which disregards trivial mathematical modeling and has a better generalization capability. By using multivariate adaptive regression splines (MARS), [11] achieves significant estimation error reductions of 85% compared with ORION 2.0. The framework in [4] using SVR can predict the average packet latency accurately while achieving about 120× speed-up over the simulation. In industry, the NoC IP vendor Arteris also presents their machine learning-based PPA predictor [12]. However, instead of graphs, in most of former works, topologies are encoded as type and size vectors, e.g. $[0, 4]$ for a 4×4 mesh where 0 represents the type of mesh, which limits their usages in irregular application-specific NoCs.

C. Graph Neural Networks for NoC Modeling

Graph neural network is a novel neural network architecture for learning hidden patterns by mapping attributed graphs to latent feature space, which is difficult for traditional methods such as convolutional neural network to handle. Then we can use the learned representations known as graph embeddings for tasks such as clustering and classification [13]. For NoCs, the important factors influencing PPA such as router radix, channel contention, and link workload can be encoded in graphs, inspiring us to apply the GNN to the new domain of estimating PPA of NoCs. By leveraging little domain knowledge, we customize a GNN-based model for our problem.

III. Methodology

Fig. 2 depicts the overall flow of our framework. In the stage of NoC conversion, the NoC and the corresponding application are mapped to an attributed graph using basic NoC knowledge. In the message passing stage, the GNN aggregates the information from neighbors within specific hops. In the readout phase, the graph-level representation vector is generated, which can be used to predict PPA lastly.

A. Basic Assumptions and Definitions

As mentioned before, we focus on application-specific NoCs with arbitrary topologies and various architectural parameters. Also, we consider the standard transaction-based interface protocol [6] classifying NIs to master NIs and slave NIs. We target buffered wormhole flow control under deterministic routing algorithms. We also assume a constant packet length for the specific NoC. To formulate the problem, we use definitions similar to [14] and [13].

Definition 1. An application can be represented as a *task graph*, which is a directed graph $AG(C,F)$ with vertex $c_i \in C$ representing a core and edge $f_{i,j} \in F$ representing directed communication from c_i to c_j. Every $f_{i,j}$ has a weight denoted as $comm_{i,j}$ representing the communication volume from c_i to c_j. An example of the task graph is shown in Fig. 1(a).

Definition 2. The topology of NoC is described by the *NoC topology graph*, which is a directed graph $TG(N,L)$. Note that we focus on NoC itself and exclude cores but their connected NIs. Vertex $n_i \in N$ represents an NI or a router, and edge $l_{i,j} \in L$ represents a directed link from n_i to n_j. An example of the NoC topology graph is shown in Fig. 1(b).

We can use the given deterministic routing algorithm to get the ideal total traffic of links according to

$$w_{i,j} = \sum\nolimits_{f_{p,q} \in F} comm_{p,q} \times \psi(p,q,l_{i,j}) \qquad (1)$$

where $w_{i,j}$ is the accumulated traffic of link $l_{i,j}$ and $\psi(p,q,l_{i,j})$ is a Boolean indicating whether the $l_{i,j}$ is part of the routing path from source core p to destination core q. As shown in Fig. 1(b), the blue dotted lines show routing paths from core 0 to core 2, 4, 5. Then $w_{7,6} = 10 + 60 + 40 = 110$. In this paper, we name the NoC topology graph with $w_{i,j}$ as weight as *workload graph* (WG), which indicates the load and congestion of network under the given task.

Definition 3. The input of the GNN is an attributed graph $G(V,E,X,X^e)$, where V and E represent nodes and edges. $X \in R^{|V| \times d}$ is a node feature matrix with $x_i \in R^d$ representing the feature vector of node v_i. And $X^e \in R^{|V| \times |V| \times d^e}$ is an edge feature matrix with $x^e_{i,j} \in R^{d^e}$ representing the feature vector of edge $e_{i,j}$.

B. Converting NoC Descriptions into Attributed Graphs

In this stage, we need to construct target attributed graphs with task graphs and NoC configurations. In part III-B1 to III-B3, we consider the simplest situation with homogeneous NoC parameters, i.e. NIs and routers have the same configuration respectively. In III-B4, we consider NoC with heterogeneous architectural parameters.

1) Power: According to [8], the major router building blocks including input buffers, crossbar switches, and arbiters, take a large proportion of dissipated power of NoCs. The important characteristics of above components are included in the NoC topology directly. For example, router 8 in Fig. 1(b) having two incoming links and two outgoing links is composed of two input buffers, one 2×2 crossbar and two $2-1$ arbiters at output ports. The dynamic power consumed by the buffers, crossbars, and arbiters is also dependent on data transmitting frequency, which can be described by the workload of corresponding links. For instance, $w_{6,8}$, $w_{9,8}$, $w_{8,4}$, and $w_{8,5}$ depict writing and reading activities of input buffers of router 8 in Fig. 1(b).

NIs in the commercial IP [5] are composed of packetizers, depacketizers, and control logics. However, implementations of NIs might have huge differences among NoC designs. Here we simply treat NIs of masters and slaves as two different kinds of routers and use the learning ability of the GNN to fit their implementation-dependent properties.

In summary, the attributed graph G for power prediction derives from workload graph, i.e. $V = N$, $E = L$ and $x^e_{i,j} = [w_{i,j}]^{\mathrm{T}}$. To distinguish different types of nodes, i.e. routers, NIs of masters, and NIs of slaves, we use one-hot encoding as node features X, e.g. $x_0 = [1,0,0]^{\mathrm{T}}$ if v_0 is a router.

2) Area: Compared with power, area prediction does not need to consider the traffic workload, i.e. task graph, because the area of NoC is decided by itself only. For the simplest situation, the attributed graph of area is as same as power, except that the edge feature matrix X^e is empty.

3) Performance: NoC performance is generally measured in terms of network latency [7]. Here we focus on average latency, which is a metric of average-case performance. For the transaction-based NoC systems, the latency of one transaction can be defined as the time between when the first request is presented at master and when the final response is accepted at master. By averaging transaction latencies between master p and slave q over a period of time, we have their network-level end-to-end latency $LAT_{p,q}$. Moreover, we can use (2) to measure the system-level global latency LAT, where $\lambda_{p,q}$ denotes the actual traffic rate. Note that $\lambda_{p,q}$ might be smaller than the ideal value $comm_{p,q}$ and the deviation can be learned by the GNN.

$$LAT = \frac{1}{\sum_{f_{p,q} \in F} \lambda_{p,q}} \sum\nolimits_{f_{p,q} \in F} (LAT_{p,q} \times \lambda_{p,q}) \qquad (2)$$

For realistic NoCs, the latency is decided by not only hops but also congestion. When a packet traverses the router, channel contention happens if it fails for arbitration and waiting occurs if no available downstream buffers due to blockage, both of which cause additional delay. We can still use the workload graph to describe the congestion of network, however, this is insufficient to model channel contention inside routers. An exmaple is in Fig. 1(a), by changing $comm_{0,4}$, $comm_{0,5}$, $comm_{1,4}$, $comm_{1,5}$ to 50, we find the recomputed workload graph identical to the original, e.g. $w_{7,6}$ keeps 110. To solve this, we represent routers in a finer-grained way by separating them into output ports to have *output port graph* (OPG). By applying this method, the components connected to router 8 in Fig. 1(b) can be represented differently under the changed task, as shown in Fig. 3, where P_i^j is the output port of router *i* connected to node *j*. The GNN can learn the probability of contention at P_8^4 is 0.6×0.4 and 0.5×0.5 in (a) and (b) of Fig. 3 after gathering the information from neighbors P_6^8 and P_9^8 and the weighted edges connecting them. We will show the huge improvement by adopting output port graph in experiment.

For end-end-latency which is a local property, we need to extract subgraph from the whole topology before NoC conversion. Firstly, end-to-end latency $LAT_{p,q}$ is influenced by the nodes in the routing path, which decides the number of hops directly, e.g. $\{n_0, n_7, n_6, n_2\}$ in Fig. 1(b) for $LAT_{0,2}$. Secondly, the neighboring routers of nodes in the path also impact the ultimate latency, e.g. $\{n_9\}$ of n_7 and $\{n_8\}$ of n_6 in Fig. 1(b) if we consider one-hop neighbors. So the subgraph for $LAT_{0,2}$ consists of $\{n_0, n_7, n_6, n_2, n_9, n_8\}$ and links among them.

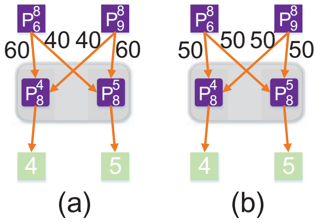

Fig. 3. (a) OPG for router 8 in original Fig. 1(b). (b) OPG for router 8 in Fig. 1(b) after changing task. Different weights of links indicate different channel contentions.

4) Dealing with NoC Heterogeneity: Heterogeneous NoC parameters are quite common for the optimized application-specific MPSoCs. The key to handling heterogeneity is inserting new parameters to node features or edge features appropriately.

The first example is the synchronizer used to transmit data crossing multiple clock domains. The clock domain NIs and routers belonging to can be seen as part of the node attribute and be inserted into node feature vectors directly. The GNN can perceive PPA overhead by gathering clock domain fields in feature vectors of neighboring nodes. The second example is the input buffer for storing flits coming from upstream temporarily. Though the buffer is one of the components of the router, it will be difficult to encode it as a node feature as there is no order for input buffers of general topologies. Since every input channel owns its buffer, we can treat the depth of buffer as part of the edge feature.

C. The GNN Architecture Design in NoCeption

The architecture of GNN model transforming attributed graphs into effective graph embeddings is essential to prediction results. To satisfy the need for directed graphs and edge features learning, we use message passing neural network (MPNN) architecture [15] in NoCeption, which consists of message passing phase and readout phase.

In the message passing phase, node v representing the NI/router and having node feature $x_v \in R^d$ is projected to hidden state $h_v \in R^{d'}$ where $d' > d$ normally. The aggregation of the information from neighboring nodes in iteration t is perfomed according to

$$h_v^{t+1} = \alpha(h_v^t + m_v^{t+1}) \tag{3}$$

where h_v^0 is obtained from x_v through MLP and α is ELU activation function. h_v^t accumulates the message $m_v^{t+1} \in R^{d'}$ produced by neighboring routers/NIs and links indicating radix of v, workload of connected links and congestion of connected channels, etc. We construct messages of two directions, i.e. mi_v for incoming and mo_v for outgoing, to distinguish their impacts from upstream/downstream such as contention/backpressure according to

$$mi_v^{t+1} = \sum_{w \in Ni(v)} Mi_t\left(h_w^t, x_{w,v}^e\right) \tag{4}$$

$$mo_v^{t+1} = \sum_{w \in No(v)} Mo_t\left(h_w^t, x_{v,w}^e\right) \tag{5}$$

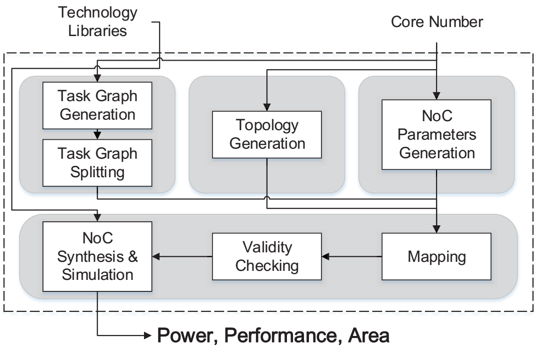

Fig. 4. Process of generating datasets for training.

$$m_v^{t+1} = mi_v^{t+1} \parallel mo_v^{t+1} \tag{6}$$

where Ni and No denote the predecessors and successors of v with Mi and Mo as their message functions separately. Mi has same form as Mo and $Mi_t\left(h_w^t, x_{w,v}^e\right) = \theta\left(x_{w,v}^e\right) h_w^t$, where θ is an MLP that maps edge feature $x_{w,v}^e$ to a $\frac{d'}{2} \times d'$ matrix. This method can help us combine impacts from neighboring nodes and neighboring edges. A simple example is contention probability generation in III-B3. The contention probability of P_8^4 can be obtained by aggregating hidden states of P_6^8/P_9^8 containing total output traffic of P_6^8/P_9^8 and the corresponding traffic from P_6^8/P_9^8 to P_8^4. We then concatenate messages from two directions.

Impacts caused by neighbors located at more than one hop can not be ignored. An instance is packets at router 7 might wait due to contention at router 8 in Fig. 1(b). After the message passing phase runs for T iterations, information from nodes and edges within T hops are aggregated. In the readout phase, we can use the readout function ρ to compress all the hidden states of nodes to a single vector H_G representing the whole graph, that is

$$H_G = \rho\left(\left\{h_v^T | v \in G\right\}\right) \tag{7}$$

In NoCeption, we simply sum all h_v^T for power and area estimation. However, the latency is more imbalanced due to more congestions around hotspot nodes. We use set2set [16] as ρ, which uses the attention mechanism to help us capture hotspot regions in NoCs. With the graph-level representation of NoC, we can predict PPA using fully-connected layers.

D. Model Training of NoCeption

For more effective training, in NoCeption, we share Mi and Mo in iterations to reduce the number of their parameters to $\frac{1}{T}$ of the original. Our optimization goal is fitting the function of predicting PPA, so we choose MSE as loss function and for mini-batch gradient descent, that is

$$MSE = \frac{1}{k}\sum_{i=1}^{k}\left(y_i - \hat{y}_i\right)^2 \tag{8}$$

where k, y_i, and $\hat{y}_i$ are batch size, the label of PPA, and the output of GNN. Grid-search is applied to search for optimized hyperparameters such as values of T and d'. Adam is selected to optimize the parameters of the GNN.

TABLE I
MAPE FOR POWER AND AREA PREDICTION (%)

Indicator	Configuration		
	Homogeneous[1]	Clock[2]	Buffer[3]
Power	2.60	2.41	2.55
Area	3.69	4.06	4.19

[1] NIs and routers have the same configuration, respectively.
[2] Cores connected to the NoC have random clock frequencies.
[3] Routers have random buffer depth at each input channel.

TABLE II
MAPE FOR LATENCY PREDICTION (%)

Configuration	End-to-End			Global		
	SVR-NoC	GNN-WG[3]	GNN-OPG[3]	SVR-NoC	GNN-WG	GNN-OPG
Homogeneous (Mesh)[1]	14.31	14.35	**8.12**	6.98	7.02	**4.42**
Homogeneous[2]	—	14.73	**9.82**	—	7.44	**4.63**
Clock[2]	—	14.86	**10.13**	—	8.99	**5.28**
Buffer[2]	—	13.18	**9.76**	—	8.65	**5.25**

[1] Same as Table I but topology of NoCs in datasets is mesh only to campare NoCeption with topology-constrained SVR-NoC.
[2] Same as Table I.
[3] In NoCeption, use workload graph and output port graph for NoC conversion respectively.

IV. EXPERIMENTAL RESULTS

A. Datasets Generation and Experimental Setup

The process of generating training datasets is shown in Fig. 4. With core numbers as input, synthetic task graphs are generated by TGFF [17]. To prevent an interface from being master and slave at the same time which is not allowed for AXI, those NIs owning both incoming and outgoing traffic are split into two. Random network topologies including regular/irregular ones and architectural parameters such as clocking domain are also generated. Then we map the task graph to the NoC randomly to get routing paths under the selected routing algorithm. Note that some of them might be illegal, e.g. existing cyclic link dependency leads to the risk of deadlock. Here we discard them directly and restart the generation. Lastly, we use a commercial NoC IP [5] with the technology libraries to synthesize and simulate the NoC to get PPA.

In our experiment, the maximum number of cores in generated task graphs is 20 and (20×2) NIs at most after splitting. Regular topologies like torus and irregular topologies like random tree are included. For homogeneous NoC configuration, buffer depths of all routers are 4 and all clock frequencies are 100 MHz. For heterogeneous ones, depths of buffer range from 4 to 12, and clock frequencies of cores can be 100/150/200 Mhz. Shortest path routing is used. We set the packet size to 64 bytes and configure the packet injection to follow Poisson distribution. TSMC 65nm library is used. The generated datasets contain 21,000 samples in total.

B. Experimental Results

To compare the differences in prediction accuracy, the statistical measurement of mean absolute percentage error (MAPE) defined in (9) is adopted, where k, y_i and $\hat{y}_i$ are the number of samples, the label of PPA, and the prediction. For a single specific sample, where $k = 1$, is called APE here.

$$MAPE = 100\% \times \frac{1}{k}\sum_{i=1}^{k}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \tag{9}$$

1) NoCeption on Synthetic Applications: We first evaluate our method on datasets where task graphs and topologies are generated randomly. The results of power and area are shown in Table I, which both achieve high accuracies. For latency prediction, here we use SVR-NoC [4] as the baseline which only uses information of one-hop neighboring routers for prediction and is limited to work with the mesh topology and homogeneous NoC parameters. As seen in Table II, compared with SVR-NoC, our method based on the output port graph achieves 6.19% and 2.56% improvement in end-to-end and global latency, respectively. Note that latency modeling using workload graph performs much worse than using the output port graph, which conforms to our analysis. NoCeption in following sections uses output port graph only. When using heterogeneous architectural parameters across the network, our method can still keep high accuracy close to the homogeneous ones. In Fig. 5, the APE distribution of NoCeption in larger NoCs keeps as in small ones, whereas SVR-NoC does not. This shows our model can generalize to NoCs with different scales.

2) NoCeption on Synthetic Traffics: We test the models trained by synthetic applications on unseen traffic patterns including complement, reverse, shuffle, and neighbor [7], on 4×4 meshes. The results are shown in Fig. 6, where NoCeption predicts more precisely than SVR-NoC, and at a higher injection rate the advantages are more obvious. We found that in Fig. 6 latency levels off at higher packet injection rate. This is caused by the transaction mechanism that huge delay of responses reaching masters slows down the sending of new transactions in busy NoC which alleviates increment of global latency.

3) NoCeption on Real Applications: We test the models trained by synthetic applications on unseen real benchmarks [18] [19] of MMS, MPEG4, VOPD, PIP, and MWD, by mapping them randomly onto NoCs to demonstrate the practical accuracy of NoCeption. Mesh topology and homogeneous parameters are still used to make NoCeption comparable with SVR-NoC. The results are shown in Fig. 7, where the power and area prediction achieve the accuracy of 97.36% and 97.83% on average, and of 93.44% and 95.92% in worst cases. We also see that on real applications, NoCeption shows an improvement of 6.52% and 4.73% on average for the end-to-end and global latency prediction compared with SVR-NoC.

V. CONCLUSION

In this paper, we have proposed a GNN-based framework for PPA prediction of application-specific NoCs with arbitrary

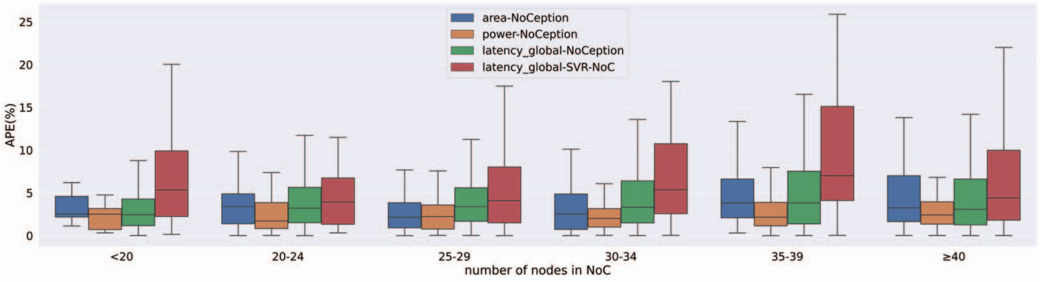

Fig. 5. Boxplot for APE distribution of increasing NoC scale. The outliers are omitted here.

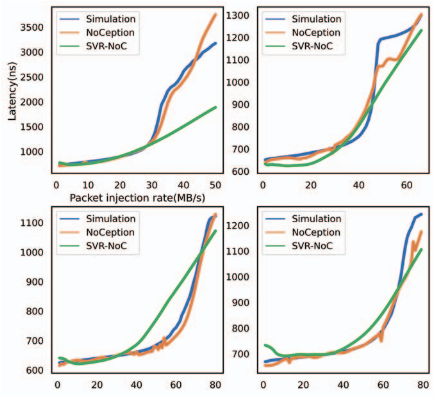

Fig. 6. Latency comparison with simulation, NoCeption, and SVR-NoC under complement, reverse, shuffle, and neighbor traffic pattern.

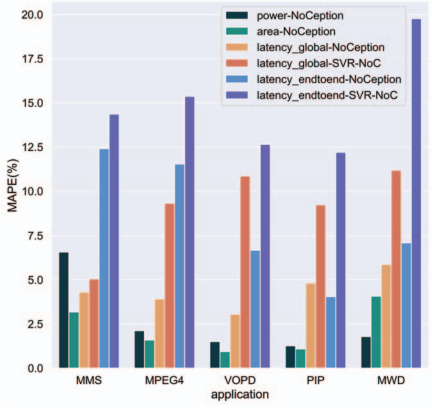

Fig. 7. PPA prediction results comparison on real applications.

topologies and heterogeneous parameters. Given an NoC and the corresponding task, the NoCeption transforms them into an attributed graph and uses it to predict PPA. Validated by datasets generated using a commercial NoC IP, our proposed approach is approved to be effective on various applications.

Acknowledgment

This paper is supported in part by the National Key Research and Development Program of China under grant 2020YFB1600201, and in part by the National Natural Science Foundation of China (NSFC) under grant No.(62090024, 61876173, 61874124). The corresponding authors are Ying Wang and Huawei Li.

References

[1] M. Kreutz *et al.*, “Design space exploration comparing homogeneous and heterogeneous network-on-chip architectures,” in *SBCCI*, 2005, pp. 190–195.

[2] D. Matos *et al.*, “Floorplanning-aware design space exploration for application-specific hierarchical networks on-chip,” in *Proc. NoCArc*, 2011, p. 31–36.

[3] J. Silva *et al.*, “An investigation of latency prediction for noc-based communication architectures using machine learning techniques,” *The Journal of Supercomputing*, vol. 75, August 2019.

[4] Z. Qian *et al.*, “SVR-NoC: A performance analysis tool for network-on-chips using learning-based support vector regression model,” in *DATE*, 2013, pp. 354–357.

[5] “Sonics Inc.” [Online]. Available: https://sonicsinc.com/

[6] “AMBA AXI and ACE Protocol Specification,” February 2013. [Online]. Available: http://www.arm.com

[7] N. E. Jerger *et al.*, *On-Chip Networks: Second Edition*, 2017.

[8] A. B. Kahng *et al.*, “Orion 2.0: A fast and accurate noc power and area model for early-stage design space exploration,” in *DATE*, 2009, pp. 423–428.

[9] Z. Guz *et al.*, “Network delays and link capacities in application-specific wormhole nocs,” *VLSI Design*, vol. 2007, pp. 90 941:1–90 941:15, 2007.

[10] A. E. Kiasari, Z. Lu, and A. Jantsch, “An analytical latency model for networks-on-chip,” *TVLSI*, vol. 21, no. 1, pp. 113–123, 2013.

[11] K. Jeong *et al.*, “Accurate machine-learning-based on-chip router modeling,” *IEEE Embedded Systems Letters*, vol. 2, no. 3, pp. 62–66, 2010.

[12] B. Winefeld, “Using machine learning for characterization of noc components,” *SNUG-2019, Silicon Valley*, 2019.

[13] Z. Wu *et al.*, “A comprehensive survey on graph neural networks,” *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–21, 2020.

[14] S. Murali and G. De Micheli, “SUNMAP: a tool for automatic topology selection and generation for nocs,” in *Proc. DAC*, 2004, pp. 914–919.

[15] J. Gilmer *et al.*, “Neural message passing for quantum chemistry,” in *Proc. ICML*. JMLR.org, 2017, p. 1263–1272.

[16] O. Vinyals, S. Bengio, and M. Kudlur, “Order matters: Sequence to sequence for sets,” *CoRR*, vol. abs/1511.06391, 2016.

[17] R. P. Dick, D. L. Rhodes, and W. Wolf, “Tgff: task graphs for free,” in *Proc. CODES*, 1998, pp. 97–101.

[18] Jingcao Hu and R. Marculescu, “Energy- and performance-aware mapping for regular noc architectures,” *TCAD*, vol. 24, no. 4, pp. 551–562, 2005.

[19] D. Bertozzi *et al.*, “Noc synthesis flow for customized domain specific multiprocessor systems-on-chip,” *TPDS*, pp. 113–129, 2005.