Approximating NoCs for Massively Parallel Error-Resilient Machine Learning Applications

Shilpa Mysore Srinivasa Murthy, Alish Kanani, Mingcong Cao, Deepak Vasudevan

***Abstract*—Network-on-chip (NoC) interconnect performance is a significant bottleneck in compute-intensive and communication-centric massively parallel applications. As parallel workloads become more demanding, it is crucial to prioritize the minimization of NoC traffic as a critical performance metric. This goal motivates techniques to reduce network traffic, such as approximating or compressing data during communication between compute nodes. Approximated data makes the packet size smaller, which reduces contention and data movement across NoCs.**

We present a floating-point precision reduction approximation scheme applied to machine learning workloads on a system with heterogeneous compute nodes connected with a NoC. We analyze machine learning workloads due to their widespread use in modern computing systems and assume that the interconnect's compute nodes are specialized for ML workloads. We see that even with significant reduction in floating-point precision, accuracy loss is negligible while power consumption and network latency improves by $\sim$1.1$\times-$ 2.6$\times$.

I. Introduction

As processing elements (PEs) become more specialized, there's a growing emphasis on efficiently connecting them. This is particularly important because data for processing is often stored in on-chip memory due to increasing data demands. Consequently, there's a need for more effective communication networks to handle the complexity of these PEs and the heterogeneity within the chip. These massively multicore systems are adopting the NoC interconnection model to address these challenges [1]. Unlike traditional bus-based networks, NoCs are asynchronous and peer-to-peer, offering several advantages, especially as on-chip heterogeneity and competition for bus resources increase. However, a significant bottleneck in the overall application performance is still data movement [2]. As packet sizes increase, network traffic surges, putting strain on the links and buffers across the interconnect. Addressing this bottleneck is crucial, especially in the context of AI-focused computing, which deals with incredibly large datasets.

A solution emerging from recent research is the concept of approximate computing [3]. The key understanding is that numerous applications are inherently error resilient to a certain degree, allowing for approximate data processing with minimal loss in accuracy. In the context of NoCs, discarding information in data payloads often requires incorporating extra circuitry to detect and eliminate redundant data transfer, thus optimizing the network's efficiency [4]–[8].

Modern artificial intelligence (AI) and machine learning (ML) applications are well-suited for approximate computing methods [9]. AI-ML algorithms typically process natural data such as images, audio, or text, which can be modified or compressed without sacrificing accuracy, thus enhancing application speed. Given the massive data volumes involved, parallelizing computation is highly advantageous, making multicore and heterogeneous architectures particularly beneficial. Moreover, the end applications are inherently error-resilient, allowing for simple bit-approximation schemes with minimal accuracy loss, making them ideal candidates for approximate computing approaches.

This study will explore the trade-offs within the design space involving accuracy, network traffic, and power consumption. Section II will delve into the approximation scheme utilized, the interconnect framework employed to assess the trade-offs, the selection of various AI-ML workloads as benchmarks, and the network parameters assigned to each benchmark. In Section III, we will conduct an evaluation of the methodology to gauge the effectiveness of the approximation scheme. Section IV provides alternative approaches found in the current literature. Finally, Section V concludes the report.

II. Methodology

In our methodology, we demonstrate the effectiveness of approximation in NoCs by employing it in selected Convolutional Neural Networks (CNNs) workloads. These CNN models have been trained using the CIFAR10 dataset. Typically, CNN architectures consist of layered structures, with each layer performing various common computing tasks such as convolution, batch normalization, and ReLu activation. Among these tasks, convolution and linear layers tend to have the highest computational requirements. Hence, in our approach, we group other layers with either convolution or linear layers. This results in a network structure where each compute layer is connected to the next, with some skip connections included. We assume that each of these layers is mapped to specific compute tiles, enabling efficient layer computation. These tiles are connected using NoC, enabling pipelined data flow architecture. Since each tile is different in compute capacity, we assume that the entire system forms a heterogeneous architecture. Figure 1 illustrates a simplified network configuration.

A. Approximation Method

To identify an approximation scheme suitable for CNN workloads, it's crucial to analyze the operations involved in these algorithms. In CNNs, each layer processes input data and passes activations to the next layer, typically represented as floating-point numbers of varying precision. One straightforward approximation approach is *precision reduction*, which

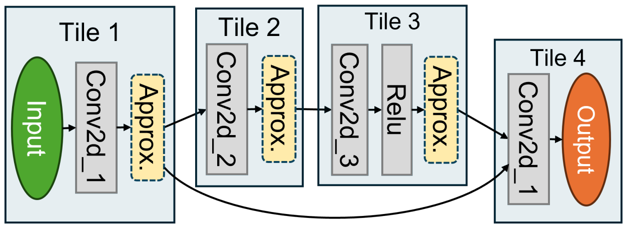

Fig. 1. The layer-tile mapping method maps neural network layers to different interconnect tiles. This example shows how a simple CNN with a skip connection is mapped to 4 tiles.

proves highly error-resilient in these networks. In our study, we introduce bit dropping as an "Approximation block" at the end of each layer as shown in Figure 1 This block effectively reduces the amount of data transmitted to the next layer. We compare the impact of dropping 4, 8, 12, 16, 20, 22, and 24 bits from the least significant bit position across different networks and workloads. By comparing the reduction in accuracy with the decrease in network latency, we assess the efficacy of the approximation in trading accuracy for reduced network traffic. We simulate these network traffic patterns using gem5 HetroGarnet [10], [11]. Additionally, these reductions in network traffic result in energy savings within the NoC, which we estimate using the *Design Space Exploration of Networks Tool* (DSENT) tool [12].

B. Network Simulation using HetroGarnet [11]:

Translating a CNN into tiles within an interconnect involves several steps. First, we map layers to appropriate tiles and model a system of these tiles connected via a NoC that represents a heterogeneous architecture. For this purpose, we utilize HetroGarnet, an interconnect network model within gem5. HetroGarnet leverages gem5's infrastructure, including topology, routing, and flit control, to offer a packet-level simulation framework [11]. Originally it was designed for simulating cache-coherent traffic within gem5 for multicore architectures.

The second step is to provide custom network traffic that accurately represents the workload. We modified HetroGarnet to accept *trace-based traffic* instead of the default cache-coherent traffic. Traces contain essential information such as the clock cycle when a packet needs to be injected, sender-receiver router IDs, and virtual network IDs. This information allows us to model the inputs and outputs of CNN layers as packets, with payload sizes corresponding to the layer outputs and inputs. Section III-A provides detailed descriptions of specific network configurations and topology selections.

TABLE I
CNN WORKLOADS AND CORRESPONDING MESH TOPOLOGY

Workload	Compute Layers	Network Topology
VGG19	17	6x3
ResNet18	21	6x4
PreActResNet18	18	6x3
Mobilenet	28	6x5
Densenet121	120	12x10

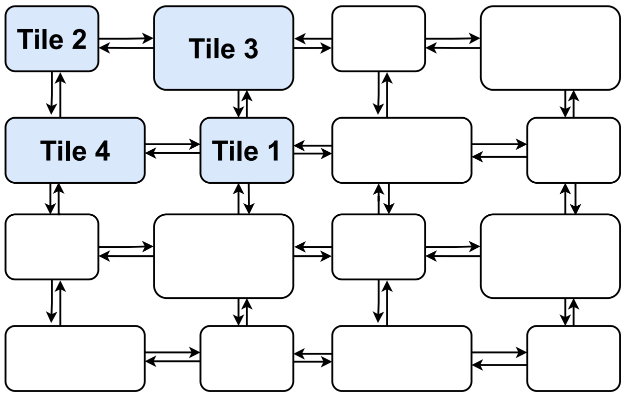

Fig. 2. The 4 mapped tiles that are highlighted correspond to the ones in Figure 1. The different relative sizes highlight the heterogeneity of the compute tiles.

C. Trace Generation: Layer to Tile Mapping

Generating a trace involves mapping various layers and data operands to appropriate tiles and packet payloads. We employ a *layer-tile* mapping method that aims to balance the latency of processing a batch of data for each tile. Figure 2 illustrates the layer-tile mapping for a simple CNN depicted in Figure 1. Where different CNN layers are mapped to different tiles, ensuring that each tile contains at most one "high-computational-cost" layer. To achieve this mapping, we initially cut all possible edges in the neural network to obtain subgraphs. Subsequently, we merge two connected subgraphs if the resulting merged graph contains fewer than one high-computational-cost layer. This merging process is repeated until no further subgraphs can be merged. Each resulting subgraph is then assigned to a compute tile. This algorithm effectively transforms a graph of CNN layers into a graph of interconnected compute tiles. The CNN workloads and corresponding network topology are detailed in Table I. The packet payload between two tiles can then be calculated from the output activation tensor shape represented by the directed edge connecting them. Specifically, the traffic payload for a tensor of shape $x \times y \times z \times b$ is given by:

$$size_{payload} = Bits_{Floating-Precision} \times x \times y \times z \times b \quad (1)$$

Where $x - y$ are 2-D dimensions of the output activation, z is number of channels and b is the batch size.

We generated the source-destination tile pairs with the packet payload. Now, we need to determine the timing of packet injection. To accomplish this, we make the following assumptions: i) each packet comprises 8 flits, where each flit can carry 128 bits ii) one flit is injected per simulation cycle iii) packets are injected simultaneously from each tile. Based on these assumptions, a straightforward algorithm can iterate through all source-destination tile pairings and assign a simulation tick number, providing timing information for each source-destination pair. Additionally, we assume there is only one virtual network in the network topologies under study. Given the presence of skip connections in the CNNs, we employ 3 virtual channels to mitigate head-of-line blocking. We assume table-based shortest path routing algorithm balancing all link utilization.

D. Power Estimation using DSENT [12]:

One of the main benefits of this approximation algorithm is that there is no overhead involved with dropping bits while there are large savings with reduced data movement. Quantifying the power reduction will give us an estimate of the energy efficacy of this approximation. To this end, DSENT is a framework that is used to evaluate the power consumed by the links, routers, and buffers [12]. DSENT takes the output statistics from the HetroGarnet simulation, and generates the dynamic and leakage power. It reports this as a total power, which is then used along with the total simulation duration to obtain an estimate of the energy consumed in the transmission of all packets.

Each workload outlined in Table I has a specific number of compute layers. Correspondingly, the chosen topology for each of these networks is the nearest integer of the form $a \times b$ to the compute layers. The reason for this choice instead of a single topology size (E.g: 12x10 for all workloads) is that DSENT reports leakage and dynamic power for every tile. This would skew the total power estimate when considering workloads with fewer compute layers. We justify choosing different topology sizes as it is equivalent to power gating the unused tiles of a larger topology, thus giving a more realistic runtime power estimate. Energy consumption by individual components, as well as the complete interconnect quantify the savings from a precision reduction approximation scheme.

III. Evaluation

A. Experimental Setup

We use 5 CNN workloads: VGG19, ResNet18, PreActResNet18, MobileNet, and DenseNet121, to validate the performance of the approximation scheme. For these 5 workloads, the inter-layer payloads are calculated, so that the layer-tile mapping step can be carried out.

The trace is built from this data, which the modified HeteroGarnet uses for network simulation. As mentioned earlier, we consider only one virtual network with 3 virtual channels. The topology parameters are set according to Table I in the HeteroGarnet build process for each workload. DSENT is then run on a 32nm node at 1GHz frequency, using the output statistics from HeteroGarnet to estimate the power consumption in the links and routers.

B. Accuracy Analysis

The main tradeoff with the approximation scheme is model output accuracy. We assume that data payload is of 8 flits per packet, where each flit can carry 128-bits. When using precision reduction, the size of the inter-layer tensors i.e., number of floating point numbers is reduced. We compare the model outputs before and after approximation to evaluate the accuracy degradation. Figure 3 shows that for all workloads, accuracy is maintained extremely evenly with bit-reductions, until 20-bits are dropped. This can be attributed to the fact that the IEEE single-precision floating-point standard has a 23-bit mantissa. As the number of least significant bits dropped crosses 20, most of the mantissa is removed and this drastically degrades accuracy.

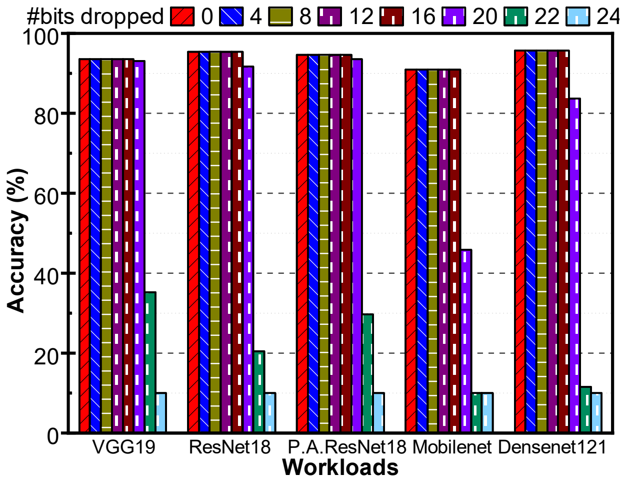

Fig. 3. Accuracy of different workloads for a specific number of least significant bits dropped.

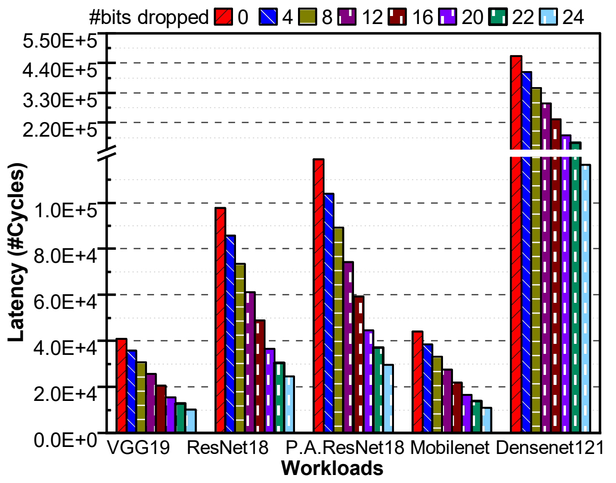

Fig. 4. End-to-end network latency of different workloads for a specific number of least significant bits dropped. Note that DenseNet121's plot is scaled down to fit within the same window.

C. Interconnect Performance

We evaluate the performance of the interconnect after the application of the approximation scheme with end-to-end network latency and energy consumption.

1) Latency: We primarily look at the end-to-end communication latency, which is the total cycles, within which all the packets specified in the trace complete their source to destination traversal. Figure 4 shows a decreasing trend across all workloads for network latency as the precision reduction increases. This is to be expected, as the bit-reduction reduces the network traffic, reducing congestion, and subsequently reducing latency across all packet flows. HeteroGarnet also reports the average flit latency, which follows an identical trend to the end-to-end communication latency.

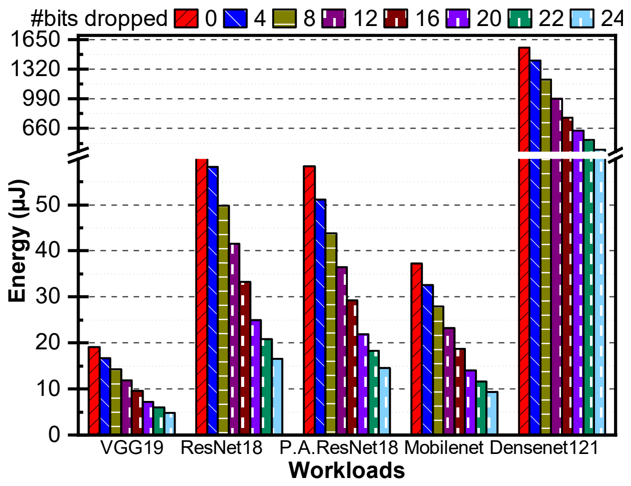

Fig. 5. Energy consumption of different workloads for a specific number of least significant bits dropped. Note that DenseNet121's plot is scaled down to fit within the same window.

2) Energy Consumption: DSENT provides dynamic, leakage, and total power consumption for buffers, routers and links. When combined with the number of total simulation cycles, we are able to compute the energy consumption from the average power numbers provided by DSENT. This energy consumption is plotted in Figure 5 across all 5 workloads. Similar to the latency plot, there is a decreasing trend with increasing bit-reduction. This trend is also expected, although with the caveat that the average leakage, dynamic, and total power remains the same before and after bit-reduction. The energy consumption reduction comes as a consequence of the reduction in active time of the network i.e., network latency, during which power is consumed.

IV. Related Works

Several approaches have been proposed to enhance the performance of NoCs, focusing on reducing traffic and optimizing data communication in compute-intensive and communication-centric applications. By leveraging high voltage for critical data and lower voltage for less critical data, AxNoC effectively manages power consumption while maintaining reliable communication [4]. Although it enhances fault tolerance, it primarily focuses on voltage-based strategies rather than data approximation techniques. The Approx-NoC framework offers a solution for reducing NoC traffic by employing data compression techniques with software-programmable accuracy [5]. It does so through lossy compression, which leads to inaccuracies in data transmission. Axba [7] is another recent work, which presents a comprehensive framework for designing approximate *bus architectures* in chip multiprocessors (CMPs). By utilizing configurable parameters to balance accuracy and energy efficiency Dapper introduces an approximation architecture tailored for GPUs [8].

Unlike the aforementioned papers, our work specifically targets the optimization of NoC communication for ML workloads. We propose a simple yet effective approximation scheme that reduces floating-point precision, thereby minimizing data size and traffic within the NoC. Importantly, unlike existing approaches, our method incurs no overhead in the critical path of data communication, ensuring efficient and accurate transmission without compromising performance. Additionally, while previous works introduce approximation with varying degrees of overhead, our approach stands out by directly addressing the unique requirements of ML workloads within the NoC domain.

V. Conclusion

In this study, we have presented a novel approach to optimize NoC communication for AI-ML workloads through precision reduction as an approximation technique. Through empirical evaluation, we observed remarkable improvements in performance metrics compared to conventional NoC architectures. Specifically, our precision reduction approximation technique led to a notable reduction in network latency and energy consumption, with observed improvements ranging from 1.1× to 2.66× without a significant drop in accuracy. Looking ahead, our work paves the way for further exploration into approximation techniques tailored for specific application domains, facilitating the development of more efficient and scalable NoC architectures for emerging computational paradigms.

References

[1] J. Vasiljevic *et al.*, "Compute substrate for software 2.0," *IEEE micro*, vol. 41, no. 2, pp. 50–55, 2021.

[2] G. Krishnan *et al.*, "Siam: Chiplet-based scalable in-memory acceleration with mesh for deep neural networks," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 20, no. 5s, pp. 1–24, 2021.

[3] T. Moreau *et al.*, "A taxonomy of general purpose approximate computing techniques," *IEEE Embedded Systems Letters*, vol. 10, no. 1, pp. 2–5, 2017.

[4] A. B. Ahmed, D. Fujiki, H. Matsutani, M. Koibuchi, and H. Amano, "Axnoc: Low-power approximate network-on-chips using critical-path isolation," in *2018 Twelfth IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*. IEEE, 2018, pp. 1–8.

[5] R. Boyapati *et al.*, "Approx-noc: A data approximation framework for network-on-chip architectures," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, 2017, pp. 666–677.

[6] D. Deb, M. Rohith, and J. Jose, "Flitzip: Effective packet compression for noc in multiprocessor system-on-chip," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 1, pp. 117–128, 2021.

[7] J. R. Stevens, A. Ranjan, and A. Raghunathan, "Axba: An approximate bus architecture framework," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–8.

[8] V. Y. Raparti and S. Pasricha, "Dapper: Data aware approximate noc for gpgpu architectures," in *2018 Twelfth IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*. IEEE, 2018, pp. 1–8.

[9] D. Kalamkar *et al.*, "A study of bfloat16 for deep learning training," *arXiv preprint arXiv:1905.12322*, 2019.

[10] J. Lowe-Power *et al.*, "The gem5 simulator: Version 20.0+," *arXiv preprint arXiv:2007.03152*, 2020.

[11] S. Bharadwaj, J. Yin, B. Beckmann, and T. Krishna, "Kite: A family of heterogeneous interposer topologies enabled via accurate interconnect modeling," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[12] C. Sun *et al.*, "Dsent-a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling," in *2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*. IEEE, 2012, pp. 201–210.