APPROX-NoC: A Data Approximation Framework for Network-On-Chip Architectures

Rahul Boyapati Jiayi Huang Pritam Majumder Ki Hwan Yum Eun Jung Kim
Texas A&M University
{rahulboyapati,jyhuang,pritam2309,yum,ejkim}@cse.tamu.edu

ABSTRACT

The trend of unsustainable power consumption and large memory bandwidth demands in massively parallel multicore systems, with the advent of the big data era, has brought upon the onset of alternate computation paradigms utilizing heterogeneity, specialization, processor-in-memory and approximation. Approximate Computing is being touted as a viable solution for high performance computation by relaxing the accuracy constraints of applications. This trend has been accentuated by emerging data intensive applications in domains like image/video processing, machine learning and big data analytics that allow inaccurate outputs within an acceptable variance. Leveraging relaxed accuracy for high throughput in Networks-on-Chip (NoCs), which have rapidly become the accepted method for connecting a large number of on-chip components, has not yet been explored. We propose APPROX-NoC, a hardware data approximation framework with an online data error control mechanism for high performance NoCs. APPROX-NoC facilitates approximate matching of data patterns, within a controllable value range, to compress them thereby reducing the volume of data movement across the chip.

Our evaluation shows that APPROX-NoC achieves on average up to 9% latency reduction and 60% throughput improvement compared with state-of-the-art NoC data compression mechanisms, while maintaining low application error. Additionally, with a data intensive graph processing application we achieve a 36.7% latency reduction compared to state-of-the-art compression mechanisms.

CCS CONCEPTS

• **Computer systems organization** → **Interconnection architectures**; **Multicore architectures**; • **Networks** → *Network performance analysis*;

KEYWORDS

Networks-On-Chip, Approximate Computing, Data Compression

ACM Reference format:
Rahul Boyapati Jiayi Huang Pritam Majumder Ki Hwan Yum Eun Jung Kim Texas A&M University . 2017. APPROX-NoC: A Data Approximation Framework for Network-On-Chip Architectures. In *Proceedings of ISCA '17, Toronto, ON, Canada, June 24-28, 2017,* 12 pages.
https://doi.org/10.1145/3079856.3080241

1 INTRODUCTION

Approximate Computing [14, 23, 30, 34] has emerged as an attractive alternate compute paradigm by trading off computation accuracy for benefits in both performance and energy efficiency. Approximate techniques rely on the ability of applications and systems to tolerate imprecision/loss of quality in the computation results. Many emerging applications in machine learning, image/video processing and pattern recognition have already employed approximation to achieve better performance [2, 11, 15, 16, 18].

Previous research has proposed several approximation techniques for emerging data-intensive applications. Software approximation mechanisms [27, 28, 32] have attempted to reduce the computation overhead by approximately executing particular sections of application code. Hardware mechanisms, that either advocate approximate computation or storage, propose to tradeoff accuracy for high performance and energy efficiency. These hardware techniques can be broadly categorized into compute-based or memory-based approximation. Compute-based approximation techniques use inexact compute units [6, 7, 14] or neural network models [13, 15, 25, 35] for code acceleration. Memory-based techniques [21, 23, 31] exploit data similarity across memory hierarchies to achieve larger capacity and energy efficiency. A significant portion of research on hardware approximation techniques has focused on either the computation units for accelerated inaccurate execution, or the storage hierarchy (cache/DRAM-based) for low overhead (area/power) memory.

However, there has been no prior research on approximate communication techniques for the interconnection fabric of multicore systems. Networks-on-Chip (NoCs) have emerged as the most competent method to connect an ever increasing number of varied on-chip components including conventional cores, accelerators, caches and memory controllers. Communication-centric applications such as image/video processing and emerging memory intensive applications in the big data era place a significant amount of stress on the NoC for high memory throughput, triggering many designs that try to solve the memory bandwidth issue [3, 4, 19, 24]. Hence designing a high-performance NoC, which can efficiently provide high throughput, has become critical to overall system performance. Therefore, the need to explore hardware approximation techniques that can leverage the modern approximate computing paradigm for high throughput NoCs is imminent.

Approximation with error control is important for guaranteed output quality [18]. Previous research has either adopted training during compilation [15, 35] or error control at runtime [18, 23] to reduce the output noise. In NoCs, since the approximation lies on the critical path of data response, it is critical to facilitate low overhead control in the inaccuracy incurred.

In this work we propose APPROX-NoC, a data approximation framework for NoCs to alleviate the impact of heavy data communication stress by leveraging the error tolerance of applications. APPROX-NoC proposes to reduce the transmission of approximately similar data in the NoC by delivering approximated versions of precise data to improve the data locality for higher compression rate. The proposed framework operates by first utilizing an approximation engine, with a lightweight error control logic, to approximate the given data block to the nearest compressible reference data pattern. Then the encoder module of an underlying NoC compression technique [12, 17] is used to compress the data block. We propose a data-type aware value approximatiion technique (VAXX), with a light weight error margin compute logic, which can be used in the manner of plug and play module for any underlying NoC data compression mechanisms. VAXX approximates the value of a given data block to the closest compressible data pattern based on the data type,with fast quantitative error margin calculation. The error threshold to control the extent of data approximation allowed can be determined by the compiler or annotated by the programmer and can be dynamically adjusted at run time.

Tightly-coupling the approximation technique with the underlying compression is more economical in terms of area and power efficiency. To this order, we present two low overhead microarchitecture implementations of value approximation for both dynamic dictionary-based compression (DI-COMP), namely DI-VAXX, and static frequent pattern compression (FP-COMP), namely FP-VAXX.

The major contributions of the work are as follows:

- We exploit approximate data similarity in communication, which translates to high data compressibility to reduce traffic load in NoCs thereby improving performance.
- We design an approximation engine with a data-type aware value approximate technique (VAXX) and lightweight error control logic to cater to a wide range of applications.
- Low overhead microarchitectural implementations to materialize the value approximation technique for static and dynamic compression mechanisms are presented.
- Our evaluation results show that APPROX-NoC provides promising opportunities in big data application domain. With an data intensive graph processing benchmark, we achieve latency reduction of 36.7% compared to state-of-the-art compression mechanisms.

The rest of the paper is organized as follows. In Section 2 we motivate our work by presenting the motivation and challenges of approximation in NoCs. In Section 3 we present the architectural overview of APPROX-NoC and the VAXX technique. Section 4 explains the microarchitectural implementation and functional principles of the VAXX techniques for dictionary-based and frequent pattern based compression mechanisms. Section 5 presents our experimental setup and evaluations. Section 6 details the related work and we conclude our work in Section 7.

2 MOTIVATION AND CHALLENGES

In this section we first detail our motivation leading to the use of data approximation in NoCs and then present the challenges of implementing the proposed techniques.

2.1 Motivation

Data movement is becoming the critical component in multicore systems. The rapid explosion of computational units in comparison with memory bandwidth, and increasing data-movement-to-compute ratio of emerging data-intensive big data workloads has resulted in heavy NoC communication loads. In such scenarios, state-of-the-art NoC designs can rapidly become the communication bottleneck and struggle to deliver the traffic in an energy-efficient manner. Multiple potential solutions like near data processing [4], moving processing to memory plane to reduce the amount of data movement, are being proposed. But even these mechanisms still require significant data movement between processsing and memory planes or within the memory plane, between the different memory slices. Therefore mechanisms that can reduce the communication traffic load in state-of-the-art NoCs become critical to cater to emerging data-intensive applications.

Frequently repeated patterns appear in applications. Previous research [12, 17, 37] has proposed using data compression techniques in NoCs to facilitate low latencies even at saturation level of injection loads. It can be trivially deduced that if enough data repetition is present in applications, then data compression mechanisms will be an appropriate antidote to the communication bottleneck issue explained above.

Data accuracy is not required. Additionally, applications that allow for approximate outputs do not require exact data to be transmitted across the network for accurate computations. Previous research [14, 30] has proposed an EnerJ framework which can be used by programmers to annotate sections of the data in applications that can be stored approximately. Doppelganger [23] proposes an approximate cache architecture that leverages the similarity between different cache blocks to eliminate redundant data storage. These mechanisms prove that in addition to repetition of specific data patterns, sufficient amount of value similarities, with small variance, between data patterns exists in many applications. But the techniques mentioned above still incur the cost of bringing the data accurately to the cache before determining whether storage is required or not. Therefore, the data movement in the NoC can be further reduced by eliminating transmission of approximately similar cache blocks across the network by using network data approximation techniques.

Defining approximate data similarity is necessary. Data similarity is defined according to a predefined error threshold. For example, when 0% error is allowed then the two patterns must be an exact match to be considered similar, however with an error of $e\%$ allowed two patterns are considered similar if the difference between them is less than $e\%$. The value difference is defined as the variance in the value between the two patterns. For example, the 8 bit patterns 10101011 and 10100000 have a value difference of 11.

2.2 Challenges

Value approximation and compression are not cheap. Value approximation and data compression mechanisms are on the critical path of the data packet injection. The underlying compression techniques have considerable area and latency overheads. Dynamic compression requires storage for pattern tracking and data lookup while static compression incurs significant encoding and decoding logic. The approximation operation adds further latency overhead on to the compression mechanism's latency. Value range and error computation using complex multiplication is expensive and hence can eat up the benefits from flit reduction achieved through approximation and compression. Thus, low latency approximation and error compute logic design are required. Although the approximation engine can be treated as a plugin module, it is economical to have tightly-coupled approximation and compression implementation alternatives for lower overheads in terms of area, latency and energy.

Quality control is important. Approximable applications still require some Quality of Service (QoS) guarantees in terms of the outputs produced or data being supplied. As mentioned in Rumba [18], it is also critical to differentiate overall quality control versus controlling errors in individual elements. Hence the proposed mechanism should be capable of controlling the data error rate individually in each cache block similar to Doppelganger [23] and also across the whole program execution. We assume that the programmer can determine the QoS needed and the compiler can translate this into error threshold allowed in different simultaneously available hardware techniques, i.e. if multiple hardware approximation techniques are concurrently available in the system the compiler/firmware can determine the error threshold each technique can incur. It should be noted that, this way, our mechanism can work in synergy with CPU/cache/storage approximation mechanisms to determine the error budget allowed in each scheme, respectively.

3 APPROX-NOC FRAMEWORK ARCHITECTURAL OVERVIEW

In this section, we first describe the baseline multicore system architecture and then detail the APPROX-NoC framework. The baseline system includes a collection of heterogeneous tiles connected via an NoC. Each tile may consist of core/accelerator units, FPGA/ASICs, private caches, a slice of the last level cache and/or an on-chip memory controller (MC) unit. The tiles are connected to routers of the NoC, in either a one-to-one or many-to-one (concentrated) fashion depending on the NoC design. Each router connects to the different components of a tile via Network Interface (NI) ports. The packetization/de-packetization of injected communication and the flit fragmentation/assembly for flow control are performed in the NI. The NoC traffic consists of control packets for message passing/shared memory and data request/reply packets. The size of the packet varies depending on whether it is an address/control packet or a data packet.

3.1 APPROX-NoC Framework

Figure 1 shows the high level architectural depiction of the APPROX-NoC framework. Traditionally, when data to be transmitted enters the NI from the tile, it is packetized and fragmented into flits in preparation for transmission. The packet is then injected into the router via the NI port in a flit-by-flit fashion. When the packet reaches its destination, the flits are assembled to restore the packet. The APPROX-NoC framework consists of a value approximate module, namely VAXX, and an encoder/decoder pair for data compression in the NI. The encoder, of the underlying compression technique, tries to compress each word in the cache block to be transmitted and sends a small encoded index with meta data instead of the whole pattern, thereby reducing the size of the packet being injected into the

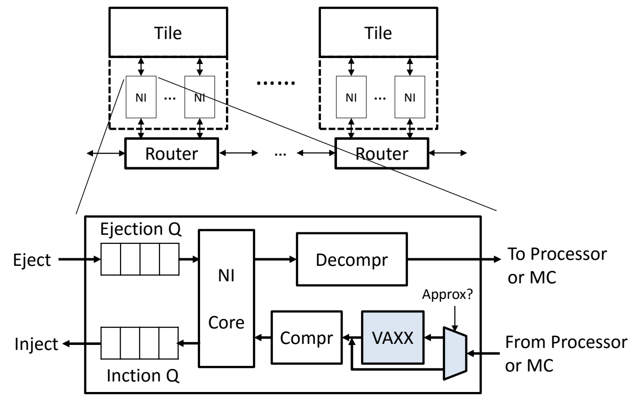


Figure 1: APPROX-NoC Architectural Overview.

network. Before compression, the VAXX module facilitates value approximation for the underlying compression scheme as detailed below, thereby improving the compression rate.

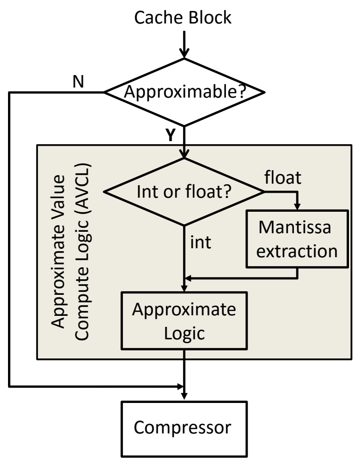


Figure 2: APPROX-NoC Operation Flowchart.

Figure 2 shows the flowchart describing the functioning of APPROX-NoC. For a cache block waiting to be injected into the network, metadata containing the approximable flag and data type are initially checked. If the cache block is not approximable, it bypasses the approximation (VAXX) engine and starts compression. In case of an approximable cache block, the data type is checked and the block is sent to the approximation logic if it is an integer. For floating-point data variables, we approximate only the mantissa fields and the approximation logic for integer values is reused to minimize the area and power overheads. The error range compute unit is also included in the approximation logic and the VAXX technique guarantees that the approximated data differs from the precise word within the preset error threshold. The approximated data blocks are then sent to encoder for compression operation.

Figure 3 shows an APPROX-NoC working example by depicting the encoding of a cache block (24B with 6 x 4B words) at the source and its decoding at the destination. The encoder in this example has two recorded reference patterns B and E, which can be

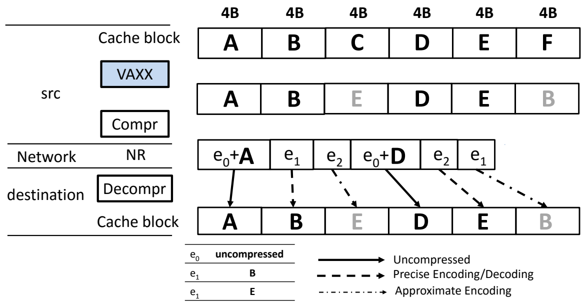


Figure 3: Compression and Decompression of a 6-Word Cache Block.

encoded, and the patterns C and F are determined to be approximately similar to E and B, respectively, using the VAXX technique. When the cache block is ready to be injected into the network, the encoder compresses the approximated block to an intermediate network representation (NR) by replacing the candidate data patterns with encoded code. The cache block, now in the NR form, is then packetized, fragmented into flits and injected into the attached router. Note that the patterns C and F are compressed approximately only if the compiler annotates the data to be safely approximable. When the packet reaches its destination, the decoder at the destination detects the reference pattern encoded code to decode the NR into the cache block which is an approximated version of the original cache block, with words C and F replaced by similar words E and B, respectively.

3.2 Approximate Value Compute Logic Design

We propose the VAXX value approximate technique to compute an approximate value for a given data block within a predetermined error threshold. In this work we focus on integer and floating-point value approximation. We approximate the cache block, to be transmitted, only when all the words in the block are approximable and this information is assumed to be carried with the access request for this block. The core of VAXX is implemented in the Approximate Value Compute Logic (AVCL), which consists of floating-point mantissa extraction, error range compute and approximate logic.

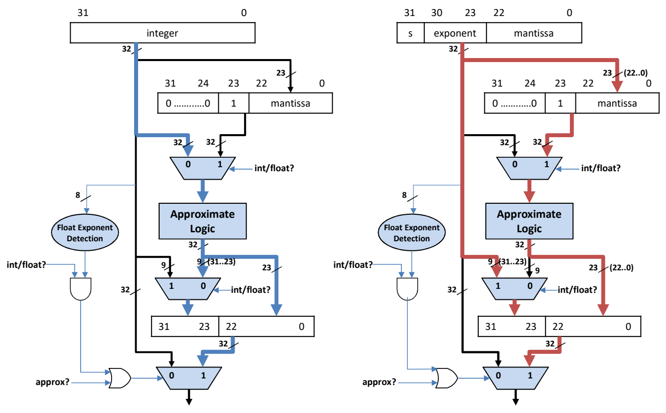


Figure 4: Approximate Value Compute Logic.

For a given value, the VAXX technique needs to compute the variance by which the approximate value can deviate from the provided precise value. For example, for a data pattern 1001(value = 9) and an error threshold of 20% the range of values 8(1000), 9(1001), 10(1010), 11(1011) can be potential matches, i.e., the data value patterns 8, 9, 10 or 11 can be approximately matched to the pattern 9(1001). We observe that in this example the 2 least significant bits are don't cares for the approximate matching, i.e., we can match the pattern "10xx" (approximate pattern) to any reference pattern to make the similarity decision. This computation can be performed using multiplication/division operations but such a design is too expensive and also unscalable.

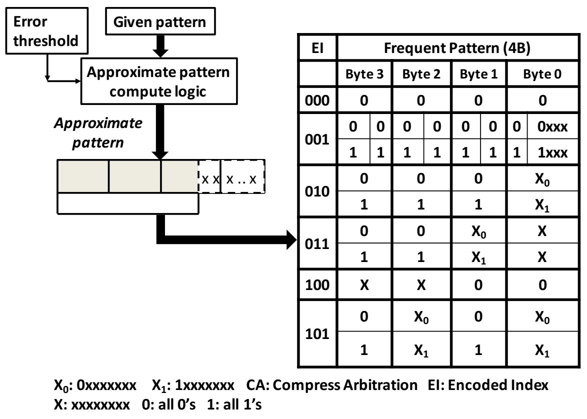
To calculate the error range, we first compute the number of bits to represent the largest error a value can tolerate given the predetermined threshold. We simplify the logic by precomputing the number of shift bits, $100/e$ where e is the error threshold (%), which are used to shift right the value to compute the error range ($error\_range = given\_value \times (e/100) \Rightarrow given\_value/(100/e)$). For example, for an error threshold of 25%, the number of shift bits is 4. Hence, when the data pattern value is 128, the *error_range* can be easily determined to be 32.

Floating-point value approximation is more complicated than integer due to the representation. A floating-point value is represented as: $(-1)^{sign} \times (1 + .mantissa) \times 2^{(exponent-bias)}$. We propose to approximate only the mantissa field of floating-point values. The mantissa part is extracted and transformed to scale to the size of an integer value, by padding the most significant bits with zeros. To transform and scale the value of a floating-point value, we extract the 23-bit mantissa part and concatenate it with a higher bit 1 to form the significant, where the exponent part is scaled out. This way both the integer and transformed floating-point variables can use the same approximate logic to maintain low overhead. Figure 4 shows the AVCL design in detail, where the datapaths taken by the integer and floating-point variables are represented separately for ease of understanding. The float exponent detection logic determines whether to bypass the approximation unit, for floating-point variables, whenever the exponent is 0 or all 1's, which represent special objects such as zero, denormalized numbers, infinity and NaN. For variables that are annotated to be non-approximable, the AVCL logic is bypassed.

The proposed APPROX-NoC framework can use the VAXX technique on top of any data compression mechanisms. But, trivially adding VAXX modules on top of NoC data compression can be expensive and unscalable due to the computation as well as latency overhead. Therefore it is critical to design microarchitectures that optimize the functionality of VAXX + compression as a whole in terms of area/latency/power. To this extent, in the next section, we showcase two microarchitectural implementation casestudies of the APPROX-NoC framework with two state-of-the-art NoC data compression mechanisms.

Index	Pattern encoded	Data Size
000	Zero run	3 bits
001	4-bit sign-extended	4 bits
010	One byte sign-extended	8 bits
011	Halfword sign-extended	16 bits
100	Halfword padded with a zero halfword	16 bits
101	Two halfwords, each a byte sign extended	16 bits
111	Uncompressed Word	32 bits

Figure 5: Frequent Pattern Compression [5].



EI	Frequent Pattern (4B)			
	Byte 3	Byte 2	Byte 1	Byte 0
000	0	0	0	0
001	0 0	0 0	0 0	0 0xxx
	1 1	1 1	1 1	1 1xxx
010	0	0	0	X_0
	1	1	1	X_1
011	0	0	X_0	X
	1	1	X_1	X
100	X	X	0	0
101	0	X_0	0	X_0
	1	X_1	1	X_1

X_0: 0xxxxxxx X_1: 1xxxxxxx CA: Compress Arbitration EI: Encoded Index
X: xxxxxxxx 0: all 0's 1: all 1's

Figure 6: FP-VAXX Microarchitecture.

4 IMPLEMENTATION OF APPROX-NOC

In this section, we first present the VAXX implementation for an underlying FP-COMP mechanism, namely FP-VAXX. Next we describe the implementation for a DI-COMP mechanism, namely DI-VAXX. Then we discuss about the latency overhead due to the approximation mechanisms.

4.1 Frequent-Pattern Mechanisms

First, we briefly describe the Frequent-Pattern Compression (FP-COMP) technique and then propose low cost microarchitectural implementation for FP-VAXX. Previous research [5] has proposed an FP-COMP mechanism for data compression and [12] has extended it for NoCs with low overhead decompression which we adopt in this work. The mechanism compresses a static set of frequent patterns as shown in Figure 5, whereas DI-COMP mechanism detects recurring patterns during run time. The FP-COMP mechanism detects a match on one of the pattern types and sends adjunct data along with the encoded index. Therefore FP-COMP incurs additional decompression complexity due to variable length compression.

4.1.1 FP-VAXX Implementation

Figure 6 depicts the microarchitectural overview of the VAXX implementation for FP-COMP. For each data word, we first compute the approximate pattern, using the AVCL. Once the don't care bits of the word are determined, the rest of the data word (shaded portion in the figure) is matched with the corresponding portion of the frequent patterns in the Pattern Matching Table (PMT) to find a match and compress on a frequent pattern hit. We propose to utilize a content addressable memory based (CAM) based structure to implement the PMT structure for fast matching. By doing this only the bits, which can be approximated according to the value error threshold, are candidates for approximation and the rest of the pattern must be a complete match to a frequent pattern for compression. For data that is not annotated to be approximable, the AVCL is bypassed to enable exact matching for given data words.

4.2 Dictionary-Based Mechanisms

Dictionary-based Compression (DI-COMP) keeps track of recurring data patterns dynamically and maintain an encoded-index consistency between senders and receivers so as to compress any occurrences of those data patterns in future communication between these senders and receivers. To track recurrent data patterns and maintain the dictionary, we propose to use a table-based mechanism similar to the one proposed in [17]. Figures 7(a) and (b) show the microarchitectural depictions of an example encoder and decoder pattern matching tables (PMTs) with size of 4 entries, respectively, in a (3x3) NoC. In the encoder PMT each entry contains a data pattern, frequency counter and a vector of encoded indices, each corresponding to one destination router (decoder), i.e. in a N node NoC each entry will have a vector of (N-1) encoded indices. For a data pattern in the encoder PMT, the vector of indices indicates whether this data pattern can be compressed for a particular destination in the network. In addition, the encoder PMT can have different encoded index values for different destinations, for the same data pattern, since each decoder performs detection in an independent fashion. The decoder PMT entries consist of the data pattern, frequency counter, encoded index and a vector of (N-1) valid bits, one for each of the N-1 encoders. The decoders detect recurrent data patterns and place them in decoder PMTs while sending an update notification to the encoder, with the new encoded index. The vector of valid bits indicates all the encoders that also have this data pattern in their PMTs and is used when replacements happen to invalidate the pattern at all encoders. In the example shown in Figures 7 (a) and (b), the encoder PMT at node 3 stores the indices for patterns 0000 and 1111 for destination 6 while the decoder PMT at node 6 has valid bits set for the respective patterns for node 3.

4.2.1 DI-VAXX Implementation

In order to optimize the microarchitectural cost of implementing VAXX matching with the DI-COMP mechanism we modify the operational flow of the approximation as described in Section 3. Instead of passing a given data block through the AVCL before reaching the compression logic we integrate tightly the AVCL with the DI-COMP scheme. We propose to compute the approximate pattern for every reference pattern, at the time of the pattern being recorded, in the DI-COMP scheme and save the approximate versions of the reference patterns. This way any given pattern can be compared to a set of approximate patterns for fast matching and hence the AVCL is removed from the critical path of the packetization.

We propose to use a Ternary Content Addressable Memory (TCAM) structure to optimize the time required to perform value-based approximation. TCAMs function similar to a CAM, and in addition to 0 or 1, a third state of "x" (don't care) is allowed, i.e., in a TCAM

Data pattern	Frequency counter	Vector of indices							
		0	1	2	4	5	6	7	8
0000	--			11			00		
0101	--				00				
1011	--					00			
1111	--						01	11	

(a) Encoder PMT at Node 3.

Data pattern	Frequency counter	Index	Vector of valid bits							
			0	1	2	3	4	5	7	8
0000	--	00	0	0	1	1	0	0	0	1
1111	--	01	0	0	0	1	1	0	0	0
1100	--	10	0	0	0	0	0	0	1	0
1110	--	11	1	0	0	0	0	0	0	0

(b) Decoder PMT at Node 6.

Figure 7: The Encoder PMT at Node 3 and the Decoder PMT at Node 6.

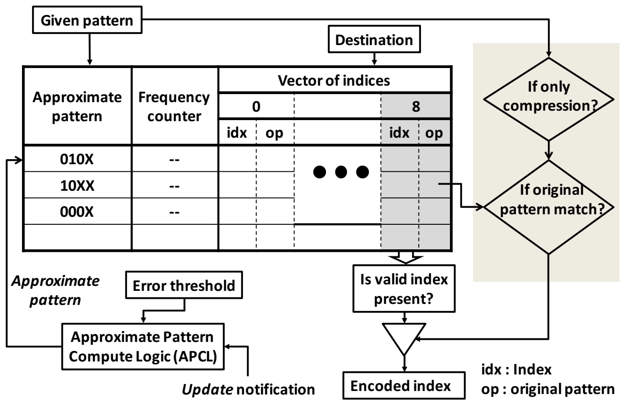


Figure 8: DI-VAXX Microarchitecture.

we can actually store 10xx for a pattern (1001) and the table entry will result in a match for the patterns 1000, 1001, 1010 and 1011. The decoders utilize a regular CAM structure to recover the original pattern (1001) based on the index. The microarchitecture of the TCAM-based encoder PMT microarchitecture is shown in Figure 8 and the operation is explained below:

- The receivers (decoders) detect frequent data patterns and send an update to the encoders to reflect in the PMT.
- When the encoder receives an update, instead of just storing the original pattern it computes the approximate pattern with don't care bits (e.g. 1001 –> 10xx) based on the error threshold, using the Approximate Pattern Compute Logic (APCL). Then the encoder records the approximate pattern in the TCAM and stores the index for the corresponding receiver. If a matching TCAM entry was already present the encoder just updates the index.
- When a data pattern arrives at the encoder, the TCAM is accessed and in case of a hit the encoded index is used for compression. This way the latency overhead on the critical path of compression is reduced.

For data packets that are not annotated for approximation this TCAM-based mechanism cannot provide compression since a TCAM match does not guarantee that the recovered pattern at the receiver is the same pattern the sender intended to transmit (e.g. 8 can match in TCAM and be recovered as 9). To facilitate exact matching along with approximate matching, we propose to add storage capability in the encoders for the original patterns in addition to the TCAM entry (approximate pattern). Figure 8 shows the encoder PMTs with the original pattern storage. Each TCAM entry can have multiple original patterns because different receivers (decoders) could have detected different patterns in the range of values. We propose to store multiple original patterns for each entry and this way when a data pattern which cannot be approximated arrives at the encoder, first the TCAM entry is matched and then an exact match on the corresponding original pattern (based on receiver) is checked before compressing it. The storage overhead can be optimized by storing only the bits of the original pattern that were made don't cares in the approximate pattern.

4.3 Latency Overhead

We assume a three cycle compression latency (two cycles matching + one cycle encoding) and two cycle decompression latency overhead for each cache block as mentioned in [12]. To ensure that the DI-VAXX and FP-VAXX matching can happen within the provisioned compression latency, based on the latency overhead evaluations we propose parallel hardware matching units. In case of DI-VAXX and FP-VAXX we have 8 parallel TCAM matching units since two matches per cycle in each unit is possible based on the model from [1] and in addition FP-VAXX requires 8 APCL units.

In addition, we propose to use two latency hiding optimizations to reduce the compression overhead. First, we propose to perform the virtual channel arbitration of the packet, using the header flit which is not compressed, in parallel with the compression. We amortize the compression overhead with the NI queueing time, i.e, if there are previous packets waiting in the queue, the compression overhead would not add to the critical path network latency of the packet.

5 EVALUATION

In this section we first explain our experimental setup and then present the evaluation of the APPROX-NoC framework.

5.1 Methodology

Experimental Setup. We evaluate our APPROX-NoC framework using a cycle accurate, in house NoC simulator and a full system simulator, gem5 [10]. We implement the DI-VAXX and FP-VAXX mechanisms in addition to the DI-COMP and FP-COMP mechanisms [12, 17] in both the simulators. For detailed network impact evaluations we use the NoC simulator where we set the default error threshold as 10% and the percentage of approximable data packets is set to 75%. We later perform sensitivity studies to show the impact of varying these parameters. To evaluate the impact of our APPROX-NoC mechanism on the overall application output error, we utilize the Pin [22] tool for instrumentation. We hand-annotate the benchmarks mentioned below, in similar fashion to Doppelganger [23], to identify the data regions which can be approximated. The VAXX mechanism uses the knowledge of the data type (floating point or

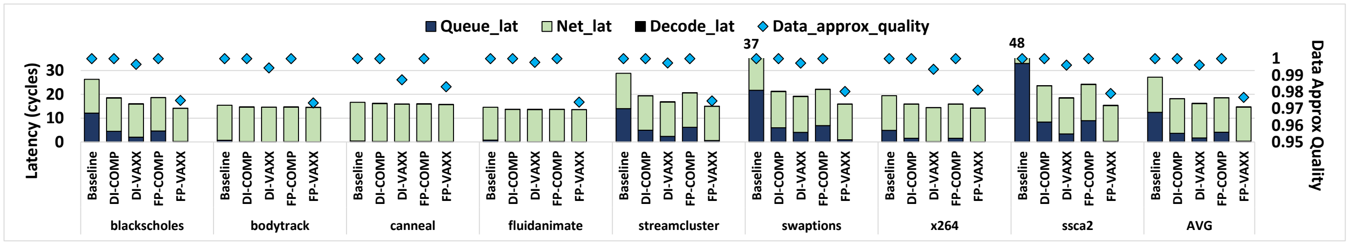
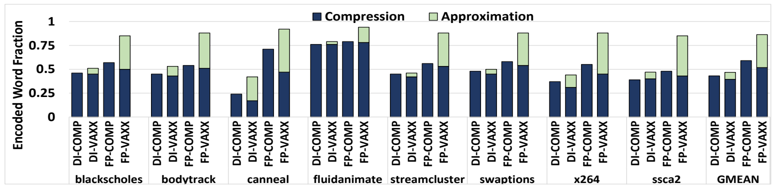
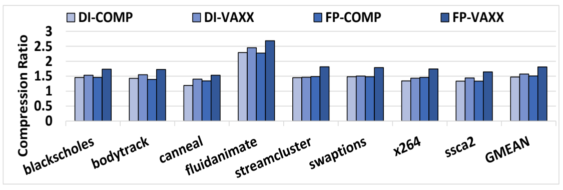


Figure 9: Average Packet Latency Breakdown and Overall Approximation Quality.



(a) Fraction of encoded words



(b) Compression ratio

Figure 10: Fraction of Encoded words Breakdown to Exact Compression and Approximation (a) and Compression Ratio Improvement of VAXX (b).

Table 1: APPROX-NoC Simulation Configuration.

System parameters	32 Out-of-Order Cores at 2GHz 32KB L1I$and 64KB L1D$, 2-way 2MB L2$ and 16 directories Cache Coherence: MOESI_hammer
NoC parameters	4×4 2D concentrated-mesh 2GHz three stage router 4 Virtual channels(4-flit buffer) 64-bit flit size wormhole switching, XY routing
Error threshold	5%, 10%(default), 20%
Approximable data packet ratio	25%, 50% 75%(default)
Dictionary-based mechanisms	8 entry PMT

integer) of variables in each benchmark to determine the approximation operation. An important consideration while hand-annotating approximable data regions of benchmarks is the data type of the variables being determined to be approximable. We assume that the data type of the cache block being compressed is known to the APPROX-NoC framework and we conservatively only compress cache blocks in which all the words have the same data type. This is because knowledge of the data type of each word would require significant metadata overhead. We use gem5 to evaluate the impact of our approximation mechanism on the overall system. The APPROX-NoC configuration and the NoC parameters used for our evaluation are listed in Table 1.

Workloads. We utilize benchmarks from the PARSEC [9], with simlarge, which have been previously utilized for evaluating approximation mechanisms [24]. In addition, we explore the approximation opportunities in big data analytics by modifying *SSCA2* [8], a data intensive graph benchmark, to evaluate betweenness centrality (BC) in real-world graphs [20]. BC is a popular graph analysis technique to identify important entities in large-scale networks. We approximate the floating-point pair-wise dependencies that is used for centrality calculation. Such applications in big data analytics can leverage approximation in data segments (e.g. weights in graphs) within a tolerable error margin since most algorithms approximate the result by only evaluating on a subset of the data with sampling. We run the benchmarks using gem5 [10] to evaluate the impact of our mechanisms on the system performance and to collect the communication traces for the region of interest, which are then fed into our NoC simulation environment and simulated for 100 million cycles for detailed NoC evaluations. To evaluate the throughput impact we utilize synthetic workloads. We collect the data injected at each node, from the gem5 benchmark traces and utilize the data traces to create data packets in the synthetic workloads. This way, the synthetic workloads can be used to vary the traffic pattern/injection rate but the data being communicated can be kept constant and correlated with data locality in the benchmarks.

5.2 Performance Analysis

In this section we present the NoC level performance evaluation of the APPROX-NoC framework using benchmarks from different application suites and synthetic workloads. We first, analyze the performance impact of APPROX-NoC on the average packet latency, compression ratio, then use synthetic workloads to evaluate the impact on network throughput.

5.2.1 Performance Analysis

Average Packet Latency. The average packet latency comparison, in a 4x4 2D concentrated mesh NoC, for the two implementation of APPROX-NoC is shown in Figure 9. Across the benchmarks DI-VAXX reduces the average packet latency by 11% with respect to DI-COMP and 40.7% compared to Baseline. FP-VAXX achieves up to 21.4% and 46.5% latency reduction compared to FP-COMP and Baseline, respectively. This is mainly due to the fact that approximation allows for more reduction in the number of injected flits leading to performance benefits, especially when the network is congested during the bursty phases. The large packet latency reduction in *SSCA2* graph benchmark is owing to the data intensive nature of the application. With a large data set, the limited cache size cannot hold the whole working set of the benchmark, and hence its irregular data accesses incur large volume of data movement. We expect that data intensive applications, in big data era, that have a high ratio of data movement to computation traffic will benefit from APPROX-NoC.

Note that the queuing latency decreases significantly by introducing approximation since the single-flit control packets face lesser blocking delays caused by the long data packets. The decoding latency portion of the average packet latency is negligible because it is amortized over the large number of control packets, and also compensated by the reduced queueing latency. In addition, it is interesting that the VAXX techniques have larger impact on packet latency with the FP-VAXX mechanism compared to the DI-VAXX. This is because the DI-VAXX mechanism needs to learn the data locality at the beginning of each new communication phase by tracking and updating its locality tables, thereby loosing approximation opportunities. In contrast, the FP-VAXX can use the static patterns across the whole program execution. For some benchmarks (*bodytrack, canneal, fluidanimate*), VAXX only achieves moderate improvement. This is because packets in these benchmarks have low queueing and network latency and the flit reduction translating to lower serialization latency is offset by the approximation/compression/decompression overheads. In addition, the percentage of data packets injected is very minimal compared to control packets, and hence the reduction in data flits does not show a significant impact on overall packet latency. The low queuing latency also supports the argument of low data to control packet ratio.

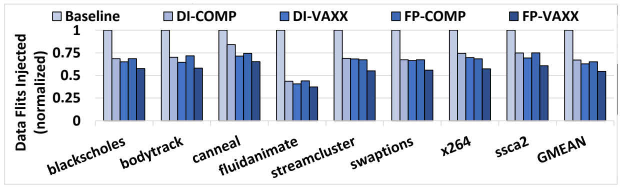


Figure 11: Reduction in Number of Injected Flits.

Approximation Effectiveness. The reduction in traffic load is shown by plotting the number of data flits injected under each APPROX-NoC mechanism in Figure 11. The DI-VAXX mechanism reduces the number of data flits injected by 3% and 38% compared to the DI-COMP and Baseline, respectively. Similarly, FP-VAXX reduces data flit volumes by 19% and 45% with respect to FP-COMP and Baseline, respectively. The moderate traffic reduction in *streamcluster* and *swaptions* benchmarks when juxtaposed with the large latency improvement seems to be counter intuitive. This can be explained by two reasons. Firstly, the value approximation enables injection acceleration for critical data, thereby translating to reduced queuing latency for the many short packets that are blocked. Therefore even though the reduction in injected flits is small the effective resulting latency reduction can be amplified. In addition, in dynamic compression, approximation may change the learnings of the DI-COMP mechanism, which might affect the compression chance of data that is required to be precise. Hence the overall flit reduction might be smaller due to changes in the operation of the DI-COMP learning. But overall we observe that the network traffic reduction translates to average packet latency improvement.

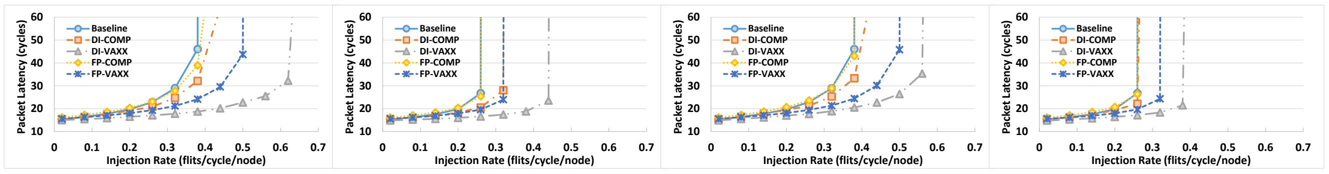
This is further supported by Figure 10 (a), which shows the breakdown of the fraction of encoded words to exact compression and approximated compression. We observe that the VAXX technique increases the encoded word fraction by up to 18% for DI-VAXX compared to DI-COMP and up to 37% for FP-VAXX over FP-COMP. Figure 10 (b) depicts the effectiveness of value approximation in improving the compression ratio. DI-VAXX and FP-VAXX enhance the compression ratio by up to 21% and 41% compared to the corresponding compression schemes, respectively. On average, the two VAXX implementation increase compression ratio by 10% and 30%. Figures 10 and 11 show that the reduction in number of injected flits does not scale proportionally to increase in compression rate due to approximation. This is because of internal fragmentation in the 8B flits where a large portion of the tail flit can be empty, since the NR might not be a multiple of 8B.

Data Value Quality. Figure 9 also depicts the data value quality for each benchmark, i.e., even though the error threshold is checked for approximating each word, the incurred error differs from word to word, so we compute the actual overall data error incurred across the benchmark execution and show the overall data value quality achieved. Across the benchmarks, though we allow for 10% error rate the effective data value quality is higher than 97%, which is due to a portion of the words being compressed without error and most of them matching with close proximity. Note that this is the quality of the integer and floating-point data values, and we analyze how this variance translate to overall application output error later.

5.2.2 Throughput Analysis

We use synthetic workloads to analyze the impact of APPROX-NoC on the network throughput. Figure 12 plots the throughput of the APPROX-NoC mechanisms compared against the Baseline, DI-COMP and FP-COMP compression schemes. We plot for data traces from *blackscholes* and *streamcluster* benchmarks, and for the Uniform Random (UR), Transpose (TR) traffic patterns. The simulations are run for 1 million cycles and we assume a 25:75 data to control packet ratio to emphasize the significance of APPROX-NoC when large amount of data is communicated.

When compared to the compression schemes, VAXX improves the throughput by up to 40% for UR and 69% for TR traffic patterns. This gain is achieved by reducing the effective injection load, due to approximating data. The huge increase in throughput compared to the latency benefits observed from benchmarks can be attributed to the larger ratio of data packets being injected. Another interesting observation is that the DI-VAXX perform better than the FP-VAXX. This is because of higher data value and temporal locality in the synthetic workloads at higher injection rates with larger data packet ratio. From our observations, the dynamic dictionary-based scheme



(a) blackscholes (UR) (b) blackscholes (TR) (c) streamcluster (UR) (d) streamcluster (TR)

Figure 12: Throughput Analysis with Different Benchmark Data Traces Under Uniform Random (UR) and Transpose (TR) Traffic Patterns.

tends to work well for applications with high data locality and intensive data movement due to its learning capability, while the static frequent pattern scheme tends to work well for applications with many frequent patterns and short communication phases without learning.

5.3 Sensitivity Studies

In this section we show the sensitivity of APPROX-NoC mechanisms to the error threshold and the percentage of approximable data packets.

5.3.1 *Error Threshold*

Figure 13 shows the average packet latency across the APPROX-NoC mechanisms for all the benchmarks by varying the error threshold. As the error threshold is increased from 5% to 10% (default) to 20% the impact of the APPROX-NoC mechanisms on packet latency amplifies due to the increased chance of approximate matching. One interesting observation is that FP-VAXX mechanism does not seem to have a significant impact on the packet latency even though a higher error threshold is allowed. The reason for that is our approximation technique can translate the approximate value into higher compression ratio even with small error threshold. It is well matched with the static frequent pattern compression. Despite the moderate latency improvement, we also observe that FP-VAXX incurs more overall error compared to DI-VAXX. This is because in the FP-VAXX mechanism, we always try to match with the highest priority frequent pattern in the PMT even though an exact match is available at lower priority. Hence some of the exact matches, when error threshold was lower, might be converted into approximate matches as the error threshold is increased. So these scenarios can lead to additional error incurred without latency benefits.

5.3.2 *Approximable Packets Ratio*

Figure 14 shows the average packet latency for the APPROX-NoC mechanisms across benchmarks as the percentage of packets approximable is varied. The packet latency benefits improve as the percentage of approximable packets increases due to the enhanced chances of approximate matching. This can be observed significantly in *SSCA2, swaptions, streamcluster* with both DI-VAXX and FP-VAXX, while the other benchmarks do not show compelling latency reduction as the percent of approximable packets is increased. The is due to the low queuing latencies in the NoC and small data-to-control packet ratio for these benchmarks leading to minimized impact of data flit reduction on the overall network latency.

5.4 Full System Impact Analysis

In this section we use Pin [22] and gem5 [10] based evaluations to analyze the impact of APPROX-NoC on the overall system. We present the overall application output errors and the overall runtime impact due to approximation on different benchmarks.

Overall Application Output Error. We analyze the impact of our mechanism on the overall application output quality in addition to the data quality using the Pin [22] instrumentation framework. We implement our approximate functionalities on top of a coherent cache simulator tool. We model a system with 16 cores and each core has a 64 KB two-way L1 private data cache of cache line size of 64 Bytes. We emulate packet response whenever a miss happens, that requires a data response from another node.

To evaluate the applications' output quality, we extend application-specific accuracy metrics based on prior approximate computing research [23, 24, 29, 32]. In addition, we exploit value approximation opportunities in big data domain by studying a graph processing benchmark *SSCA2*. *SSCA2* calculates the betweenness centrality scores of the nodes in a small world network to identify the key entities. So we evaluate the pair-wise betweenness centrality difference between the approximate output and its precise counterpart for error calculation.

Applications' output accuracy for all benchmarks are shown in Figure 16. With the predetermined 10% data noise margin, all the benchmarks are well controlled within the error bound except for *streamcluster*. This because by approximating the coordinates, the cost between points and centers might deviate from the precise one and lead to mismatch of centers between the approximate version and precise version. As mentioned in previous work, through approximate space exploration or training during compilation we can improve the accuracy while maintain the performance benefit [29, 32].

In Figures 17, we show the application output of *bodytrack*'s approximated and precise pair. The two figures are very similar and the difference is hardly captured through human vision. In this experiment, we allow for 10% error threshold in the data and observe that the overall output vectors differ by 2.4%.

Figure 16 also shows the output accuracy with different error thresholds. Even with 20% error budget, the applications' output errors are close to 5% except for *streamcluster* and *swaptions*. With the bounded data error control, APPROX-NoC can achieve high throughput and low latency by exploiting approximate communications while maintaining acceptable output quality.

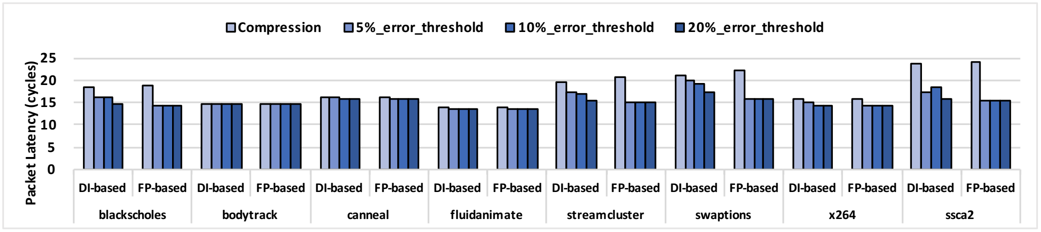


Figure 13: Error Threshold Sensitivity Analysis.

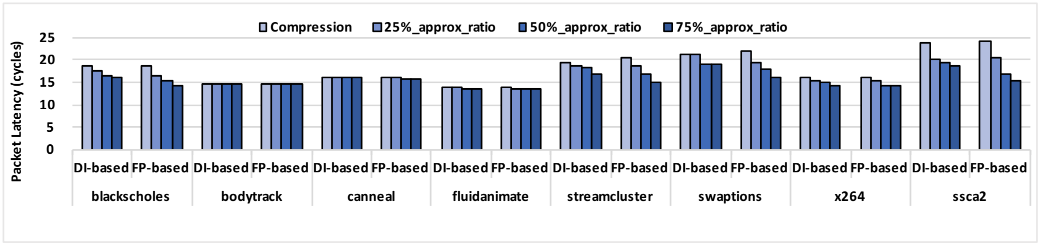


Figure 14: Approximable Packets Ratio Sensitivity Analysis.

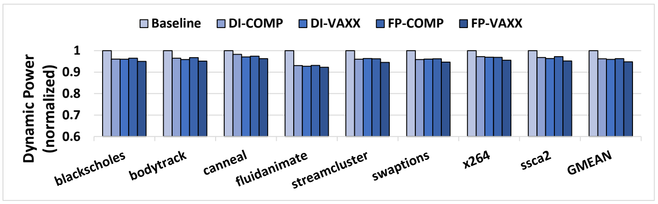


Figure 15: Dynamic Power Consumption Normalized to Baseline.

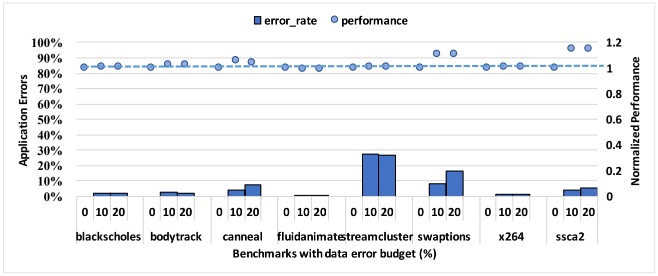


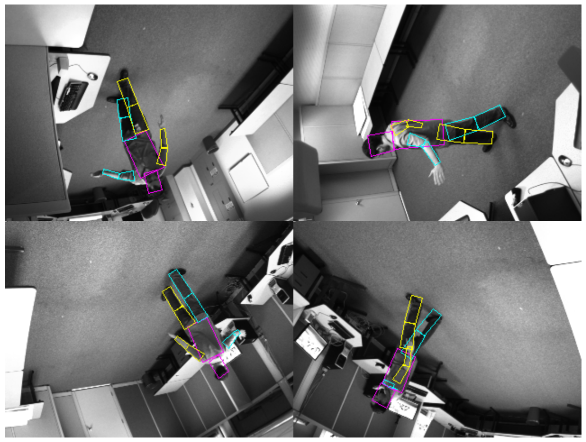
Figure 16: Application Output Accuracy and Normalized Performance.

Overall Application Performance. Next we analyze the impact of APPROX-NoC on the overall system performance. We configure a 64-core CMP connected by an 8x8 mesh network, and run the benchmarks for 100 million instructions with medium input size. Figure 16 also shows the normalized performance for different benchmarks with the proposed approximation mechanism as the error threshold allowed is varied, normalized to 0% error threshold allowed. We observe that the performance is improved by upto 10% and 14% in *swaptions* and *SSCA2*, respectively, while we see moderate improvements on the rest of the benchmarks. This is because *swaptions* and *SSCA2* have higher degree of sharing in the approximable region of interest in the application code compared to the other benchmarks. Higher degree of sharing leads to a significant amount of similar approximable data being transferred across the NoC during the execution of these benchmarks, thereby improving the efficacy of our mechanism in impacting the overall performance.

5.5 Power Consumption and Area Overhead

In this section, we evaluate the effect of APPROX-NoC on the network power consumption and area overhead, while taking into consideration the overhead of approximate matching and compression/decompression. The static power consumption does not vary across benchmarks and the static power overhead of all the APPROX-NoC mechanisms is minimal compared to the large baseline static power consumption. Hence to show the variation in power consumption between APPROX-NoC mechanisms and benchmarks, we depict dynamic power consumption in Figure 15. The best performing FP-VAXX mechanism reduces the dynamic power consumption on average by 5.4% compared to baseline and 1.3% compared to FP-COMP. Note that this can be primarily attributed to the reduction in the number of injected flits which compensates for the power overhead of VAXX techniques.

Based on the hardware requirements we evaluate the area overhead of the APPROX-NoC encoders using CACTI [26] and verilog based area analysis with 45nm technology. The DI-VAXX incurs 0.0037 mm^2 for each NI (router). Similarly, FP-VAXX require an overhead of 0.0029 mm^2. The decoder design does not change between the schemes and the overhead is as mentioned in [12].


(a) Precise Output.

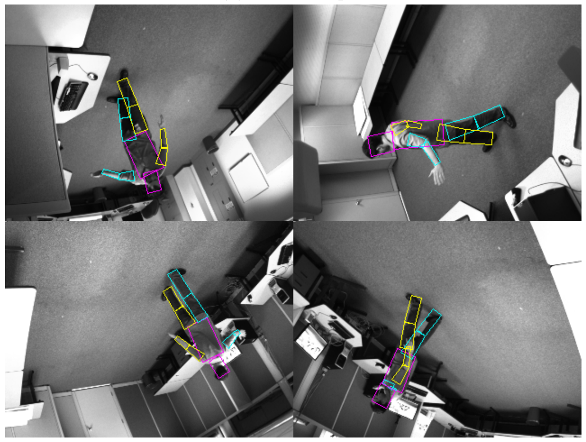

(b) Approximate Output.

Figure 17: Approximate versus Precise Output of Bodytrack.

6 RELATED WORK

In this section we discuss the related work in hardware approximation techniques and NoC data compression.

Approximation. Significant research has been done regarding approximated computation and data storage in hardware for applications that allow inaccurate outputs. Sampson et al. [29–31] proposed code annotations and compiler framework for the programmers to define the data/computations in the application that can be approximated. They also propose hardware mechanisms like voltage scaling, reducing DRAM refresh rate and SRAM supply voltage, width reduction in floating point computations for energy savings. Esmaeilzadeh et al. [14] propose dual voltage operation where precise computations use high voltage mode and approximate operations use the low voltage mode. Previous research has also proposed energy efficient accelerators based on neural networks and analog circuits [15, 25, 33, 35]. Liu et al. [21] propose to reduce the refresh rate of DRAM memories which store data, that can be inaccurate, using application level input. Miguel et al. [23] propose, Doppelganger, a cache mechanism which eliminates the storage of cache blocks with data that is similar (need not be exact match). They keep the tags for all the cache blocks, but if two cache blocks are similar then only one is stored and both the tags point to this block. Our mechanism proposes to eliminate the transmission of similar cache blocks by encoding data to a similar data pattern, that is being tracked, at the source node (memory/cache) and hence can work in synergy with approximate storage mechanisms like Doppelganger cache.

NoC data compression. Previous research has explored data compression in NoCs. Das et al. [12] explored compression in caches and the NI of the routers while proposing techniques to amortize the decompression latency with communication latency. They observe that across wide range of workloads data compression leads to significant network power savings and performance benefits. Zhou et al. [37] proposed a data compression mechanism in packet-based NoC architectures by tracking frequently repeated values in the on-chip data traffic. Zhan et al. [36] introduced a base-delta compression technique in NoCs to exploit the small intra-variance in data communication. Jin et al. [17] proposed a data compression mechanism that learns frequent data patterns using a table-based mechanism and adaptively turns the compression on/off based on the efficacy of compression on the network performance. APPROX-NoC proposes to compress the data traffic by facilitating approximate matching with an online error control mechanism.

7 CONCLUSIONS AND FUTURE WORK

In this work we propose APPROX-NoC, a hardware data approximation framework for high throughput NoCs in the memory intensive big data era. We present a value based approximate matching technique to use in a plug and play fashion with any underlying data compression mechanism. We also detail low cost microarchitectural implementations of the VAXX techbique with state-of-the-art dictionary-based and frequent pattern-based NoC data compression mechanisms. Our evaluation results show that the best APPROX-NoC mechanism reduces the average packet latency up to 21.4% over state-of-the-art NoC data compression mechanism. In addition, our evaluation results with synthetic workloads show that the best APPROX-NoC mechanism improves throughput up to 60% compared to state-of-the-art compression mechanisms. We observe that the FP-based mechanisms achieve higher approximation rate and hence performance benefits across the benchmarks, but the DI-based mechanisms outperform the FP mechanisms when there is significant data repetition. On average the application output quality is always above 99% across the benchmarks even though a 10% error threshold is allowed since a large portion of the words are within close proximity. As future work, we intend to leverage this high approximation quality by using window based instead of word based error threshold, i.e., use cumulative error threshold over a set of data words defined by a window, so as to achieve more approximate matches. This can be applicable especially in cases of video/image applications where the error rate over a frame is more appropriate than a conservative per word error threshold.

ACKNOWLEDGEMENTS

We sincerely thank the reviewers for their helpful comments and suggestions. We especially thank Joshua San Miguel for the discussion on approximate region annotation in the benchmarks. This research is supported by NSF grant 1423433.

REFERENCES

[1] Banit Agrawal and Timothy Sherwood. 2008. Ternary CAM Power and Delay Model: Extensions and Uses. *IEEE Trans. Very Large Scale Integr. Syst.* (2008), 554–564.

[2] Omar Alejandro Aguilar and Joel Carlos Huegel. 2011. Inverse Kinematics Solution for Robotic Manipulators Using a CUDA-Based Parallel Genetic Algorithm. In *Proceedings of the 10th Mexican International Conference on Advances in Artificial Intelligence - Volume Part I (MICAI 2011)*. 490–503.

[3] Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. 2015. A Scalable Processing-in-memory Accelerator for Parallel Graph Processing. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA-42)*. 105–117.

[4] Junwhan Ahn, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. 2015. PIM-enabled instructions: a low-overhead, locality-aware processing-in-memory architecture.. In *Proceedings of the 42th Annual International Symposium on Computer Architecture (ISCA-42)*. 336–348.

[5] Alaa R Alameldeen and David A Wood. 2004. Frequent Pattern Compression: A Significance-Based Compression Scheme for L2 Caches. *Dept. Comp. Scie., Univ. Wisconsin-Madison, Tech. Rep* 1500 (2004).

[6] Carlos Alvarez, Jesus Corbal, and Mateo Valero. 2005. Fuzzy Memoization for Floating-Point Multimedia Applications. *IEEE Trans. Comput.* 54, 7 (2005), 922–927.

[7] Carlos Álvarez, Jesús Corbal, and Mateo Valero. 2012. Dynamic Tolerance Region Computing for Multimedia. *IEEE Trans. Computers* 61 (2012), 650–665.

[8] David A. Bader and Kamesh Madduri. 2005. Design and Implementation of the HPCS Graph Analysis Benchmark on Symmetric Multiprocessors. In *Proceedings of the 12th International Conference on High Performance Computing (HiPC 2005)*. 465–476. https://doi.org/10.1007/11602569_48

[9] Christian Bienia. 2011. *Benchmarking Modern Multiprocessors.* Ph.D. Dissertation. Princeton University.

[10] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The Gem5 Simulator. *SIGARCH Comput. Archit. News* 39 (2011), 1–7.

[11] M. Creel and M. Zubair. 2012. High Performance Implementation of an Econometrics and Financial Application on GPUs. In *Proceedings of International Conference on High Performance Computing, Networking, Storage and Analysis (SCC 2012)*. 1147–1153. https://doi.org/10.1109/SC.Companion.2012.138

[12] Reetuparna Das, Asit K. Mishra, Chrysostomos Nicopoulos, Dongkook Park, Vijaykrishnan Narayanan, Ravishankar R. Iyer, Mazin S. Yousif, and Chita R. Das. 2008. Performance and Power Optimization Through Data Compression in Network-on-Chip Architectures. In *Proceedings of the 14th International Conference on High-Performance Computer Architecture (HPCA-14)*. 215–225.

[13] Zidong Du, Avinash Lingamneni, Yunji Chen, Krishna V. Palem, Olivier Temam, and Chengyong Wu. 2015. Leveraging the Error Resilience of Neural Networks for Designing Highly Energy Efficient Accelerators. *IEEE Trans. on CAD of Integrated Circuits and Systems* 34 (2015), 1223–1235.

[14] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. 2012. Architecture Support for Disciplined Approximate Programming. *SIGPLAN Not.* 47, 4 (2012), 301–312.

[15] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. 2012. Neural Acceleration for General-Purpose Approximate Programs. In *Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-45)*. 449–460.

[16] Alexander Guzhva, Sergey Dolenko, and Igor Persiantsev. 2009. Multifold Acceleration of Neural Network Computations Using GPU. In *Proceedings of the 19th International Conference on Artificial Neural Networks: Part I (ICANN 2009)*. 373–380.

[17] Yuho Jin, Ki Hwan Yum, and Eun Jung Kim. 2008. Adaptive Data Compression for High-performance Low-power On-chip Networks. In *Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-41)*. 354–363.

[18] Daya S. Khudia, Babak Zamirai, Mehrzad Samadi, and Scott Mahlke. 2015. Rumba: An Online Quality Management System for Approximate Computing. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA-42)*. 554–566.

[19] Snehasish Kumar, Naveen Vedula, Arrvindh Shriraman, and Vijayalakshmi Srinivasan. 2015. DASX: Hardware Accelerator for Software Data Structures. In *Proceedings of the 29th ACM on International Conference on Supercomputing (ICS 2015)*. 361–372.

[20] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data. (June 2014).

[21] Song Liu, Karthik Pattabiraman, Thomas Moscibroda, and Benjamin G. Zorn. 2011. Flikker: Saving DRAM Refresh-power Through Critical Data Partitioning. In *Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-XVI)*. 213–224.

[22] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. 2005. Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. In *Proceedings of the 26th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2005) (PLDI '05)*. 190–200.

[23] Joshua San Miguel, Jorge Albericio, Andreas Moshovos, and Natalie Enright Jerger. 2015. Doppelganger: A Cache for Approximate Computing. In *Proceedings of the 48th International Symposium on Microarchitecture (MICRO-48)*. 50–61.

[24] Joshua San Miguel, Mario Badr, and Natalie Enright Jerger. 2014. Load Value Approximation. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-47)*. 127–139.

[25] Thierry Moreau, Mark Wyse, Jacob Nelson, Adrian Sampson, Hadi Esmaeilzadeh, Luis Ceze, and Mark Oskin. 2015. SNNAP: Approximate Computing on Programmable SoCs via Neural Acceleration. In *Proceeedings of the 21st IEEE International Symposium on High Performance Computer Architecture (HPCA-21)*. 603–614.

[26] Naveen Muralimanohar, Rajeev Balasubramonian, and Norm Jouppi. 2007. Optimizing NUCA Organizations and Wiring Alternatives for Large Caches with CACTI 6.0. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-40)*. 3–14.

[27] Mehrzad Samadi, Davoud Anoushe Jamshidi, Janghaeng Lee, and Scott A. Mahlke. 2014. Paraprox: Pattern-Based Approximation for Data Parallel Applications. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-XIX)*. 35–50.

[28] Mehrzad Samadi, Janghaeng Lee, D. Anoushe Jamshidi, Amir Hormati, and Scott Mahlke. 2013. SAGE: Self-tuning Approximation for Graphics Engines. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-46)*. 13–24.

[29] Adrian Sampson, André Baixo, Benjamin Ransford, Thierry Moreau, Joshua Yip, Luis Ceze, and Mark Oskin. 2015. Accept: A Programmer-Guided Compiler Framework for Practical Approximate Computing. *University of Washington Technical Report UW-CSE-15-01* 1 (2015).

[30] Adrian Sampson, Werner Dietl, Emily Fortuna, Danushen Gnanapragasam, Luis Ceze, and Dan Grossman. 2011. EnerJ: Approximate Data Types for Safe and General Low-Power Computation. In *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2011)*. IEEE, 164–174.

[31] Adrian Sampson, Jacob Nelson, Karin Strauss, and Luis Ceze. 2013. Approximate Storage in Solid-State Memories. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-46)*. 25–36.

[32] Stelios Sidiroglou-Douskos, Sasa Misailovic, Henry Hoffmann, and Martin Rinard. 2011. Managing Performance vs. Accuracy Trade-offs with Loop Perforation. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (ESEC/FSE 2011)*. 124–134.

[33] Renée St. Amant, Amir Yazdanbakhsh, Jongse Park, Bradley Thwaites, Hadi Esmaeilzadeh, Arjang Hassibi, Luis Ceze, and Doug Burger. 2014. General-purpose Code Acceleration with Limited-precision Analog Computation. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture (ISCA-41)*. 505–516.

[34] Swagath Venkataramani, Vinay K. Chippa, Srimat T. Chakradhar, Kaushik Roy, and Anand Raghunathan. 2013. Quality Programmable Vector Processors for Approximate Computing. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-46)*. 1–12.

[35] Amir Yazdanbakhsh, Jongse Park, Hardik Sharma, Pejman Lotfi-Kamran, and Hadi Esmaeilzadeh. 2015. Neural Acceleration for GPU Throughput Processors. In *Proceedings of the 48th International Symposium on Microarchitecture (MICRO-48)*. 482–493.

[36] J. Zhan, M. Poremba, Y. Xu, and Y. Xie. 2014. Leveraging Delta Compression for End-to-End Memory Access in NoC Based Multicores. In *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*. 586–591. https://doi.org/10.1109/ASPDAC.2014.6742954

[37] Ping Zhou, Bo Zhao, Yu Du, Yi Xu, Youtao Zhang, Jun Yang, and Li Zhao. 2009. Frequent Value Compression in Packet-based NoC Architectures. In *Proceedings of the 2009 Asia and South Pacific Design Automation Conference (ASP-DAC 2009)*. 13–18.