BIT-BASED APPROXIMATION FOR APPROX-NOC : A DATA APPROXIMATION FRAMEWORK FOR NETWORK-ON-CHIP ARCHITECTURES

A Thesis

by

DIVYA RAVI SIRAVARA

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Gwan Choi
Co-Chair of Committee,	Eun Jung Kim
Committee Members,	Jiang Hu
Head of Department,	Miroslav M. Begovic

May 2019

Major Subject: Computer Engineering

ABSTRACT

The dawn of the big data era has led to the inception of new and creative compute paradigms that utilize heterogeneity, specialization, processor-in-memory and approximation due to the high demand for memory bandwidth and power. Relaxing the constraints of applications has led to approximate computing being put forth as a feasible solution for high performance computation. The latest fad such as machine learning, video/image processing, data analytics, neural networks and other data intensive applications have heightened the possibility of using approximate computing as a feasible solution as these applications allow imprecise output within a specific error range.

This work presents Bit Based Approx-NoC, a hardware data approximation framework with a lightweight bit-based approximation technique for high performance NoCs. Bit-Based Approx-NoC facilitates approximate matching of data patterns, within a controllable error range, to compress them thereby reducing the data movement across the chip. The proposed work exploits the entropy between data words in order to increase their inherent compressibility. Evaluations in this work show on average 5% latency reduction and 14% throughput improvement compared to the state of the art NoC compression mechanisms.

DEDICATION

To my parents, brother and Prithvi for helping me realize my dream and for their unconditional love and support.

ACKNOWLEDGEMENTS

I would like to express gratitude to my advisor, Dr. E J Kim, and my committee members Dr. Gwan Choi and Dr. Jiang Hu for their guidance and support throughout the course of this research. I would also like to express my sincere thanks to the guidance extended by all the members of the HPC Lab with the Computer Science Department at Texas A&M University, particularly to Dr. Kim and Jiayi Huang. Thanks also go to my department faculty and staff for making my time at Texas A&M University a great experience. Lastly, but with utmost importance, I am grateful to the support extended by my family and friends throughout the course of my Master's degree.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supervised by my thesis committee consisting of Dr. Eun Jung Kim of the Computer Science Department and Dr. Gwan Choi and Dr. Jiang Hu from Electrical and Computer Engineering Department at Texas A&M University. Boyapati et al. have proposed Approx-NoC, a data approximation framework for Network-on-Chip architectures. It was presented at International Symposium on Computer Architecture(ISCA) in 2017. Current work is an extension to the research carried out on Approx-NoC by Dr. Rahul Boyapati and Jiayi Huang, from the HPC Lab with the Computer Science Department at Texas A&M University under the guidance of Dr. E J Kim.

All other work conducted for the thesis was completed independently by the student.

Funding Sources

There are no outside funding contributions to acknowledge related to the research and compilation of this document.

NOMENCLATURE

Approx	Approximate
AVCL	Approximate Value Compute Logic
BAXX	Bit-Based Approximation
CAM	Content Addressable Memory
CMP	Chip Multi-Processor
DBX	DeltaBitPlane-XOR
DI-COMP	Dictionary-based Compression
fp	Floating Point
FP-COMP	Frequent Pattern Compression
int	Integer
LSB	Least Significant Bit
MSB	Most Significant Bit
NI	Network Interface
NoC	Network on Chip
PMT	Pattern Matching Table
TCAM	Ternary Content Addressable Memory
VAXX	Value-Based Approximation
VC	Virtual Channel
Z-RLE	Zero-Run Length Encoder

TABLE OF CONTENTS

Page

LIST OF FIGURES

LIST OF TABLES

Page

1. INTRODUCTION

Approximate Computing [1, 2, 3, 4] has emerged as an attractive alternate compute paradigm by trading off computation accuracy for benefits in both performance and energy efficiency. Approximate techniques exploit the underlying ability of applications to withstand some amount of imprecision/loss in the quality of the application results. Many evolving applications in machine learning, image/video processing and pattern recognition have already employed approximation techniques to achieve better performance [5, 6, 7, 8, 9].

Prior research work has proposed various approximation techniques for evolving data intensive applications. Several software approximation mechanisms [10, 11, 12] have attempted to reduce the computation overhead by approximately executing certain sections of the application code. Hardware mechanisms are of two categories, which are those that utilize approximate computation or approximate storage, propose to trade-off accuracy for high performance and energy efficiency. These hardware techniques can be broadly categorized into compute-based or memory-based approximation.

A significant portion of research on hardware approximation techniques has focused on either the compute-based approximation [13, 14, 15] for faster but inaccurate execution, or the storage-based [16, 17, 18] (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 until the work on Approx-NoC, A data approximation framework [19].

A large amount of strain has been placed on the NoCs for high memory throughput due to the upcoming memory intensive applications in the big data period and communication-centric applications such as image/video processing. This has led to designers trying to solve the memory bandwidth issue [20, 21, 22, 23]. 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 impending.

Previously, work has been done on an approximation engine with Value based approximation technique [19]. The Approx-NoC 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 [19]. Then the encoder module of an underlying NoC compression technique [24, 25] is used to compress the data block.

This work proposes an extension to the Approx-NoC framework by adding a bit-based approximation technique (BAXX), 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. The main idea of BAXX boils down to transforming each memory data block such that the lower order bits of all the words are placed together, to improve the inherent compressibility of data while avoiding high approximation value error. The transformed data has higher inherent compressibility and increased value locality. The error threshold to control the extent of data approximation is determined by the programmer and can be dynamically adjusted at run time.

To this order, this work presents two low overhead microarchitecture implementations of bit-based approximation for both dynamic dictionary-based compression (DI-COMP) and static frequent pattern compression (FP-COMP).

The major contributions of this work are as follows:

- Design of an approximation engine with a data-type aware Bit Based Approximation Technique (BAXX) where the data block is transformed following which computations are performed on the data block before approximating the data block.
- Low overhead micro-architectural implementations of BAXX for both the FP-COMP (static) technique as well as for the DI-COMP (dynamic) technique with lightweight error control logic.
- The evaluation results show that Bit-Based Approx-NoC provides promising opportunities in the big data application domain. Overall we have 57% improvement in the compression ratio as compared to the baseline system and 5% improvement compared to the state of the art compression techniques. The work has 4% latency improvement and 14% throughput improvement over the state-of-the-art compression techniques.

2. BACKGROUND

NoC is a network-based communication subsystem on an integrated circuit. NoCs have emerged as the most competent method to connect an ever-increasing number of varied on-chip components. It facilitates data transfer among multiple components such as processor cores, memories, and specialized IP blocks present in the Chip Multi-Processor (CMP) system. This chapter briefly talks about the general NoC architecture and the various building blocks involved in this work.

2.1 NoC Architecture

The NoC architecture is made of three main blocks Network Interface (NI), Routers and Links.

2.1.1. Network Interface

The NI facilitates communication between the IP cores and the on-chip network. It acts as a bridge between the network and the cores. NI injects packets from the core into the network and ejects data packets from the network to the cores. NI also facilitates homogeneity in the on-chip network as it packs and un-packs the data as per the various communication protocols followed by the different components of the CMP.

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 manner. When the packet reaches its destination, the flits are assembled in order to restore the packet. This work consists of a BAXX module and an encoder/decoder pair for data compression in the NI.

2.1.2. **Routers**

The flits are injected into the routers one by one following which the flits are transported through the network until they reach their destination. There are four constraints that govern the design of the network, namely topology, routing, flow control and router microarchitecture. Each of these parameters is discussed below.

- *Topology:* Topology defines the physical layout of the network, that is, it defines the physical connections between nodes and links in the network. Topology determines the number of hops required to transfer a message from the source to its destination. Hop count has a direct impact on the power consumed by the network. This work employs a mesh topology. Mesh is a *k*-ary *n*-cube design, where *k* is the number of nodes along each dimension and *n* is the number of dimensions. This work uses an 8x8 mesh network.
- *Routing Algorithm:* For a given network, the routing algorithm decides the route to be taken by the packets in order to reach their destination. The routing algorithms ability to balance the traffic flow across the network directly impacts the latency and throughput of the network. This work employs the XY routing algorithm.
- *Flow Control:* Flow control manages the allocation of resources to packets as they progress along their route to their destination. The key resources are buffers and channels. In this work the NoC is configured to use wormhole switching, which manages resources at the granularity of flits. With wormhole switching

mechanism, a flit is allowed to move to the next router as soon as sufficient buffer space is available in the downstream router to hold this flit.

- *Router Microarchitecture:* Router microarchitecture defines the build of the routers used in the NoC. The main building blocks of a router are input buffers, VC allocator, routing computation logic, switch allocator and crossbar. This work uses 4 Virtual channels.

2.1.3. **Links**

A communication link is composed of a set of wires and connects two routers in the network. Links may consist of one or more logical or physical channels and each channel is composed of a set of wires. Generally, a flit is the size of a *phit*, physical unit and this is the smallest unit that can be transmitted through a link at any given point of time. Flit length is usually the same as the channel width.

2.2. Compression in NoCs

The NoC can have two types of compression, *cache compression* where the cache architecture needs to be modified in order to compress data and store data in this compressed manner and *NI compression/decompression* modules where the compression/decompression modules are present in the NI and can be used in a plug and play manner, packets are compressed before being injected into the network. This work employs the latter approach and uses encoder/decoder pair as plug and play modules for data compression in the NI. Once compression is done, the data is transmitted to the routers in a flit-by-flit fashion.

2.3. Approximate Computing

Approximate computing is an attractive new compute paradigm that has come about for data-intensive applications. Approximate computing leverages the ability of applications and systems to have a certain level of tolerance for imprecision of the computation results. Applications employ approximation in order to gain better performance and energy efficiency at the cost of imprecision. This work leverages this fact and employs a bit-based approximation technique in order to approximately compute data by changing certain bits of a word in a block of the cache in order to further increase the compression ratio of the data packets in the network and hence reduce the volume of data movement across the chip.

3. RELATED WORK

Approximate computing techniques rely on the ability of applications and systems to tolerate imprecision/loss of quality in the computation results. Various upcoming applications in machine learning, image/video processing and pattern recognition have utilized approximation methods in order to achieve better performance [5, 6, 7, 8, 9].

Prior research has proposed various approximation techniques for the upcoming data-intensive applications. Software approximation techniques [10, 11, 12] have attempted to reduce the computation overhead by executing only certain sections of the application code. Several hardware approximation techniques either approximate the computation or the storage; hence propose to tradeoff computational accuracy for higher efficiency. Hardware techniques can be classified into compute-based approximation or memory-based approximation. The compute-based approximation techniques use imprecise computation units [13, 14, 15] or neural network models [26, 7, 27, 28] while the memory-based approximation techniques exploit the data similarity across the memory hierarchies. Compute-based approximation units are used for code acceleration while the memory-based approximation units are used in order to achieve larger capacity and more energy efficiency. However, no research prior to Approx-NoC [19] has used approximation techniques for the high throughput NoCs.

3.1. Bit-Plane Compression

J Kim et al. [29] proposed Bit Plane Compression (BPC) for transforming data for better compression in many core architectures. BPC introduces the concept of a Bit-Plane transformation to compression. Prior to encoding BPC algorithm transforms each memory data block using a novel transformation technique DeltaBitPlane-XOR (DBX), in order to improve the inherent compressibility of the data.

The DBX transformation is composed of three steps; first Delta, that subtracts the adjacent values in the memory; followed by Bit-Plane, that rotates the input values such that each new value contains one bit of each of the original input values all at the same bit position; the final step is the XOR the adjacent values. The DBX transformation increases the value locality and hence reduces entropy, thus increasing the inherent compressibility.

Following the DBX transformation, long zero runs of bit-planes are encoded by Zero Run Length Encoding (Z-RLE), whereas the non-zero bit-planes are encoded using frequent pattern encoding. The DBX transformation greatly increases the effectiveness of the compression algorithm.

3.2. APPROX-NoC: A Data Approximation Framework for NoC Architectures

The work done previously was on approximation which was memory-based or compute-based. However, there has been no prior work on approximate computing techniques for the interconnection fabric of multicore systems until the work done by R. Boyapati et al. [19].

3.2.1. Architectural Overview of the Approx-NoC Framework

3.2.1.1. APPROX-NoC Framework

The baseline system consists of various tiles. Each tile may consist of core/accelerator units, FPGA/ASICS, caches or on-chip memory controller units. As shown in Figure 3-1 each component of the tile is connected to a router via NI ports.

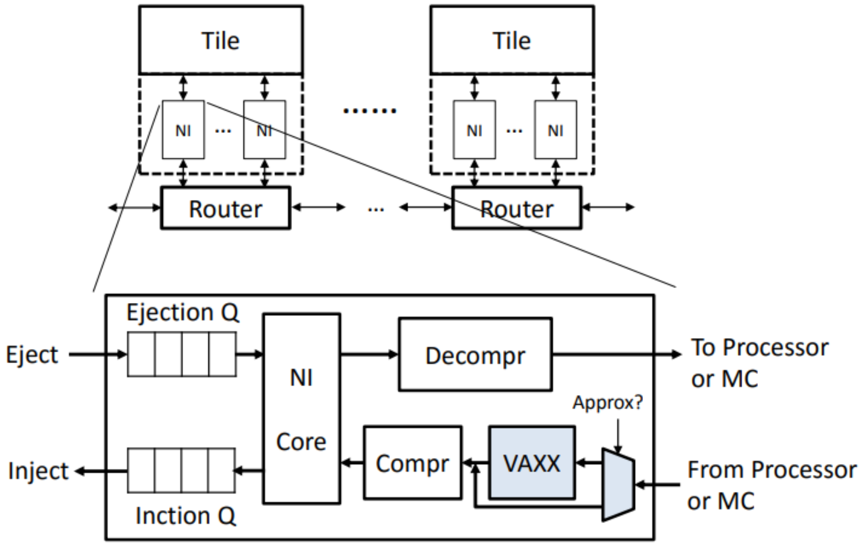


Figure 3-1: APPROX-NoC Architectural Overview

The Approx-NoC framework has a VAXX module that is present in the NI, along with an encode/decode unit for data compression. Once the message is sent to the NI, the data is first approximated based on the approximation logic, following which the data is compressed and then divided into flits and injected to the router in a flit-by-flit fashion. The packetization and de-packetization of the message for flow control is performed in the NI.

Figure 3-2 describes the working of the Approx-NoC framework. For a particular cache block that needs to be injected into the network, the metadata present in the cache block is first checked to see if the block of cache is approximable or not and the data type of the cache block is checked as well, integer (int) or floating-point (fp). If the cache block is not approximable it is directly sent to the compression unit, otherwise the data type of the cache block is checked, if it is an int the entire cache block is approximated word by word and if it is an fp, only the mantissa of all the words are approximated one after the other. Following the VAXX unit, all the approximated words are sent to the compression unit.

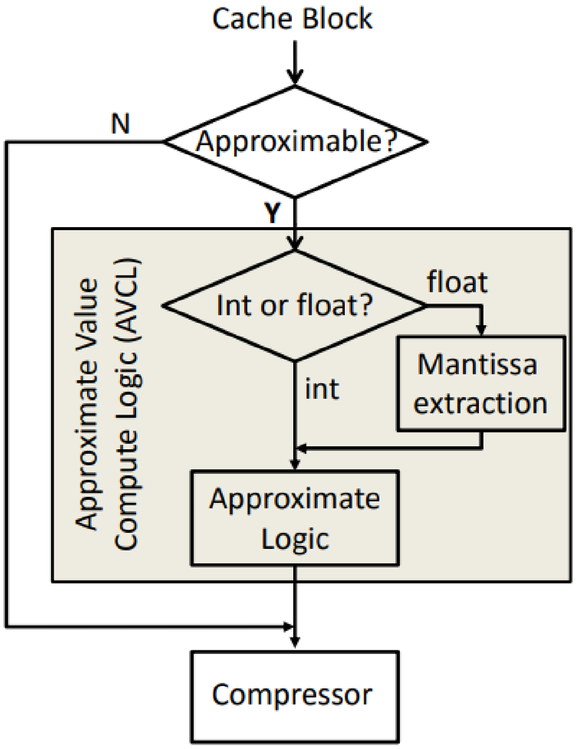


Figure 3-2 Approx NoC Flowchart

After compression the data is then sent into the router from the NI in a flit-by-flit manner. The flits are then sent to the destination through the network where it is decompressed, and the approximate block is then recovered.

3.2.1.2. VAXX model Design

In order to compute an approximated value for a particular data block the VAXX model is used. Data blocks are approximated only when all the words of that data block are approximable, this information is part of the metadata of the cache block. The data blocks are approximated within a specified error threshold. The main design of the VAXX model lies in the Approximate Value Compute Logic (AVCL) design. AVCL consists of the floating-point mantissa extraction, error range computation and the approximation logic.

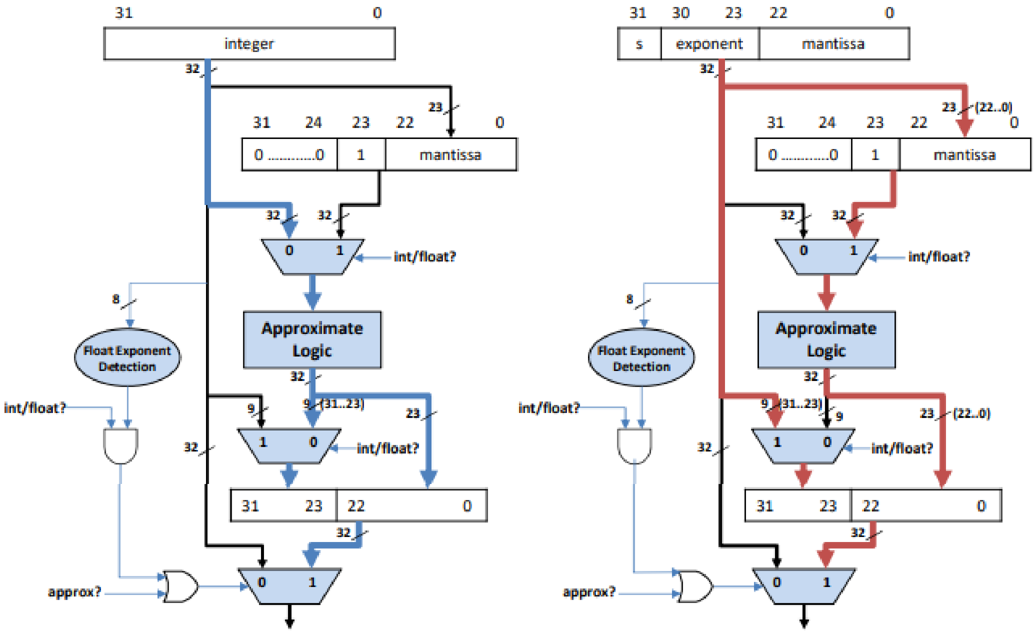


Figure 3-3 Approximate Value Compute Logic

For a particular value, the VAXX model needs to calculate how much deviation from the actual value is allowed, this is calculated by using the specified error threshold. For example, if the error threshold is 20% and the value is 1001 (9), then the allowed values would be 1000 (8), 1010 (10) and 1011 (11). This basically means that the last two bits, from the least significant bit (LSB) side are considered as don’t cares, that is 10xx.

In order to calculate the error range from the error threshold %(e). 100/e will give the number of shift bits, which are used to right shift the given value by. Error_range = given_value * (e/100). This means that for e = 25%, the number of shift bits is 4. So, if the data value is 128 then the error_range can be estimated to be 32.

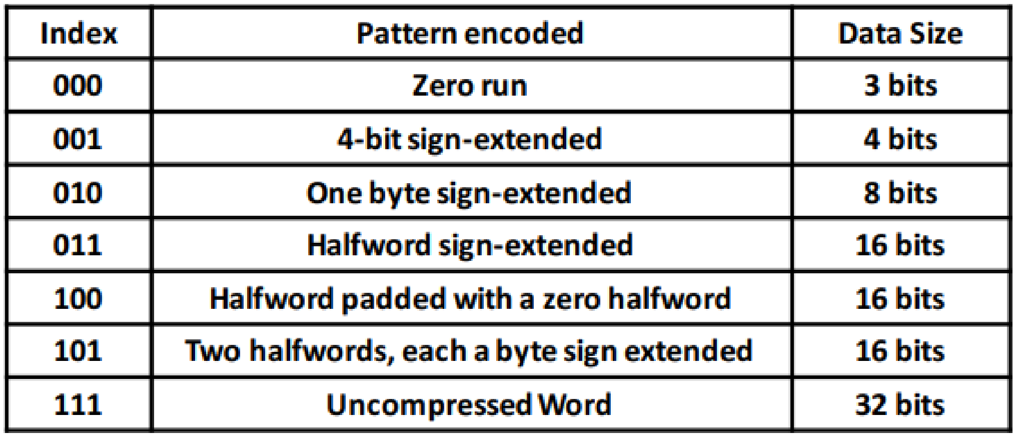
The procedure mentioned above can be followed for int approximation, however for fp approximation a few slightly complex steps need to be followed as only the mantissa of the fp values can be approximated. First the mantissa field of the fp number is extracted and 0’s are padded to the most significant bits in order to make the mantissa bit the same size as the int values. Figure 3-3 shows the AVCL model design for both int and fp data types. For cache blocks that are deemed unapproximable the AVCL unit is bypassed.

3.2.2. **Implementation of Approx-NoC**

The VAXX implementation is done on top of FP-COMP namely, FP-VAXX and DICT-COMP, namely DICT-VAXX.

3.2.2.1. Frequent Pattern Mechanisms

Prior work on compression proposed a FP-COMP [30] technique for data compression and [24] has extended it for NoCs with low overhead decompression which has been used in this work. This mechanism is static and hence compares a static set of frequent patterns as shown in Figure 3-4.

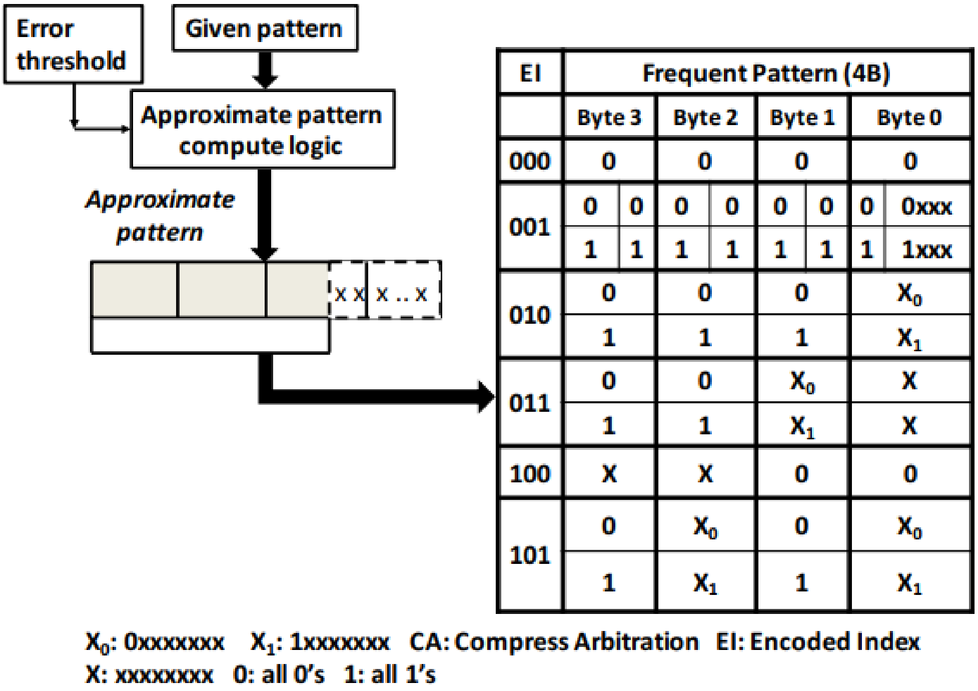


Index	Pattern encoded	Data Size
000	Zero run	3 bits
001	4-bit sign-extended	4 bits
010	One byte sign-extended	8 bits
011	Halfword sign-extended	16 bits
100	Halfword padded with a zero halfword	16 bits
101	Two halfwords, each a byte sign extended	16 bits
111	Uncompressed Word	32 bits

Figure 3-4 Frequent Pattern Compression [30]

3.2.2.1.1. FP-VAXX Implementation

The microarchitecture of the FP-VAXX is shown in Figure 3-5. First the approximate pattern is computed for each data word using the AVCL unit. Upon finding the don’t care bits of the data word, the rest of the word, which is the shaded portion as shown in Figure 3-5, is matched to the appropriate frequent pattern which is present in the pattern matching table (PMT). Upon a pattern match, the data word is compressed. The PMT structure has been implemented using a content addressable memory (CAM) based structure. For data words that are approximable based on their value only the shaded portion needs to be a match, however for data that is not approximable the AVCL is bypassed and the entire data word needs to be a match in order to undergo compression.



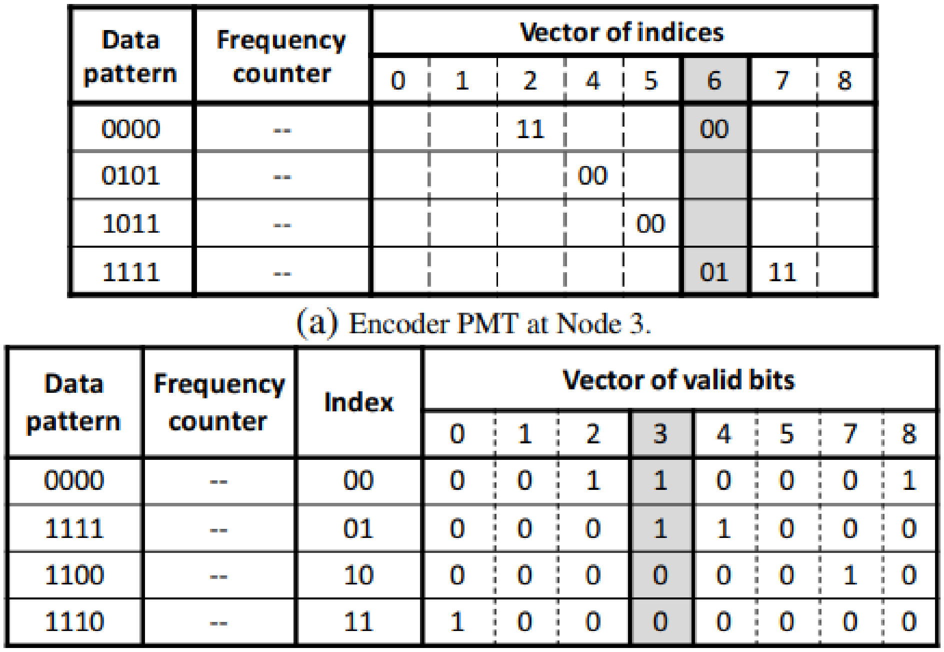
EI	Frequent Pattern (4B)			
	Byte 3	Byte 2	Byte 1	Byte 0
000	0	0	0	0
001	0 0	0 0	0 0	0 0xxx
	1 1	1 1	1 1	1 1xxx
010	0	0	0	X_0
	1	1	1	X_1
011	0	0	X_0	X
	1	1	X_1	X
100	X	X	0	0
101	0	X_0	0	X_0
	1	X_1	1	X_1

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

Figure 3-5 FP-VAXX Microarchitecture

3.2.2.2. Dictionary-Based Mechanisms

DI-COMP keeps track of repeating data patterns in a dynamic manner and maintains an encoded index consistency between the senders and receivers so that future data patterns that match these data patterns can be compressed. In order to keep track of the recurring data patters a table-based mechanism similar to [25] is proposed. Figures 3-6 (a) and (b) depict the encoder at decoder PMTs with size entries of 4 in a (3x3) NoC at node 3 and node 6 respectively.



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 3-6 Encoder and Decoder PMTs

In the encoder PMT for an N node NoC, each entry will have a vector of (N-1) encoded indices; each of these indices indicates whether a particular data pattern can be compressed for a specific destination in the network. As the decoder performs detection in an independent manner, the encoder can have different index values for different destinations but for the same data pattern.

The decoder PMT consists 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 the repeating data patterns and place them in the decoder PMT, while sending a new updated encoder index to the encoder PMT. This has been depicted in Figures 3-6 (a) and (b).

3.2.2.2.1. DI-VAXX Implementation

Unlike the FP-VAXX scheme where a data block is first passed through the AVCL before being compressed, the DI-VAXX scheme tightly couples the AVCL unit with the DI-COMP unit. At the time that the pattern is recorded the approximate pattern is computed for every given pattern in the DI-COMP and the approximate versions are saved. This enables fast matching and the AVCL unit is removed from the path of packetization.

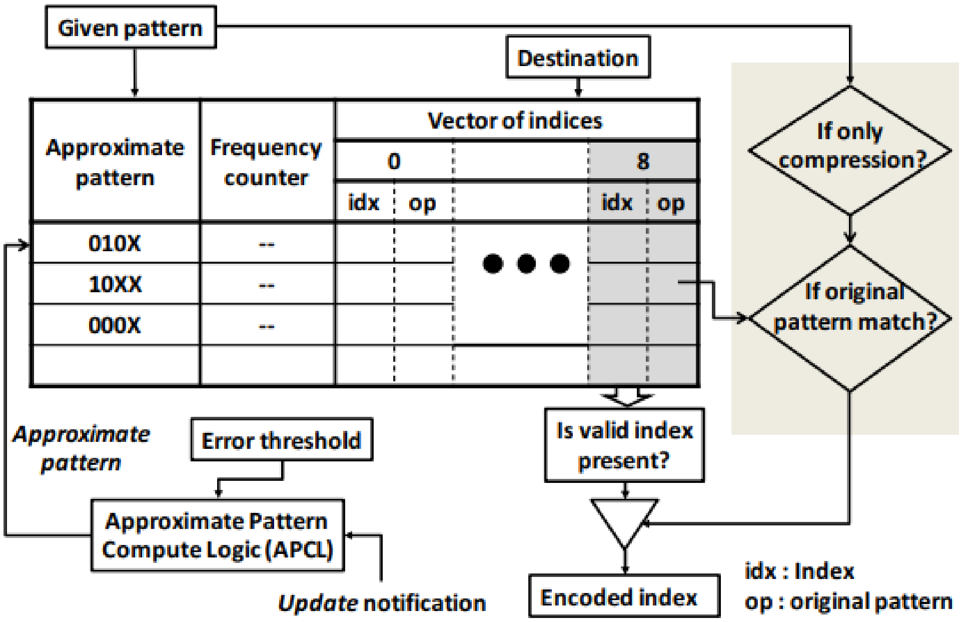


Figure 3-7 DI-VAXX Microarchitecture

The work proposes Ternary Content Addressable Memory (TCAM) structure to make the value-based approximation faster. TCAMs are similar to CAMs in their functioning and along with the 0 or 1 state they also have the x (don’t care) state. This means that in TCAM a state 10xx can be stored in the place of 1001 and patterns 1000, 1001, 1010 and 1011 will be a match to this entry. The decoders on the other hand have a CAM structure and recover the original 1001 pattern from the table. The microarchitecture of the TCAM based DI-VAXX is shown in Figure 3-7.

The operation of the DI-VAXX is as explained; the receivers/decoders detect the frequent patterns and send the update to the encoder PMTs. The encoder calculates the approximate patterns with the don't care bits and stores the new value in the PMT at the specified index, if the entry already exists the index is just updated to reflect the new index. When a data pattern arrives at the encoder, the TCAM is accessed and in case there is a TCAM hit the encoder index is sent for compression, this tight coupling reduces the latency overhead.

For data patterns that are not approximable the TCAM-based mechanism does not facilitate compression. Hence this work proposes to have storage capability in the encoders for the original patterns in addition to the approximate patterns enabling compression for data patterns that are not-approximable.

4. BIT BASED APPROX-NOC

4.1. Motivation

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. VAXX technique works well for both the FPC and DICT mechanisms in this regard for a predefined error threshold value. For both the mechanisms there is significant improvement in the compression ratio and the number of flits that are injected as compared to FPC-COMP and DICT-COMP mechanisms. However, there is more room for improvement.

The fact that needs to be considered is that error in data increases as we try to approximate the higher order bits**.** Approximating a word becomes more and more expensive as we go to the higher order bits, which are the most significant bit (MSB) side of the word. In the VAXX mechanism if we only approximate the lower order bits, then the higher order bits of the various words need to match in order to come across a pattern match. If the lower order bits of all the words of a cacheline are clustered together and then approximated, we can approximate any bit of the new clustered word as they are all least significant bits (LSBs) and hence would not be expensive and this would inherently increase the compression ratio. The idea builds the motivation to explore an improved version of the existing Approx-NoC framework, *Bit-Based Approx-NoC*.

4.2. Key Idea

Bit-Based Approx-NoC is an improved version of the existing Approx-NoC framework and uses the same baseline NoC architecture mentioned in Section 3.2.1.1.

Figures 4-1 and 4-2 show the main idea behind the bit-based approximation (BAXX) approach. In Figure 4-1, it can be seen that at the sender NI when a cache block arrives, it is pre-processed. In the pre-processing step a transpose is applied to the entire cache block. The transpose is performed in order to have the bits from each word of equal weightage together, in our case as we have a 16-way cache block with each way consisting of 32 bits, we take two bits from each way to form the new transposed cache block, with the LSBs clustered together and the MSBs clustered together. In Figure 4-2, the post-processing step at the receiver NI is shown. At the receiver NI, the compressed cache block is transposed again taking two bits at a time in order to get back the approximated word. In this step the approximated bits are brought back to their original positions.

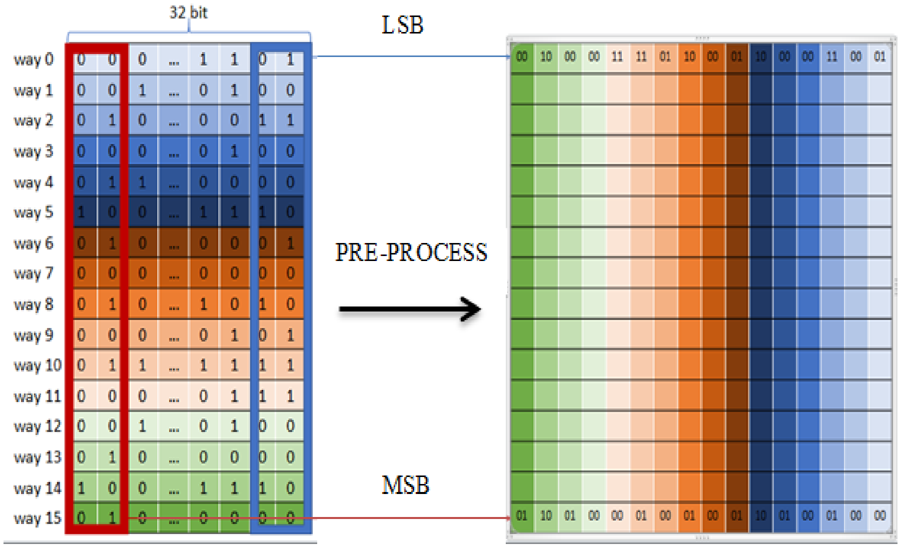


Figure 4-1 Pre-Process step at Sender NI

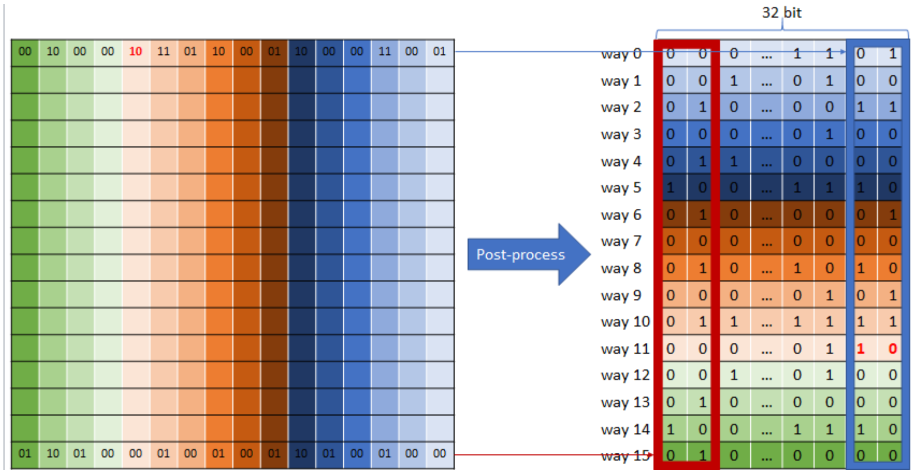


Figure 4-2 Post-Process step at Receiver NI

Figure 4-3 describes the flow of the entire algorithm involved in *Bit-Based Approx-NoC*. The cache block is received at the NI it is then checked to see if is approximable or not. If it is approximable, the cache block is then pre-processed as shown in Figure 4-1. Once it is transposed, BAXX technique is used to approximate the cache block. Upon approximation the approximated words are then sent into the compression unit to produce compressed data. The compressed data is then injected into the network and routed to the destination. At the destination the data is decompressed, following which the data is post-processed as shown in Figure 4-2. Upon post-processing the approximated words are retrieved.

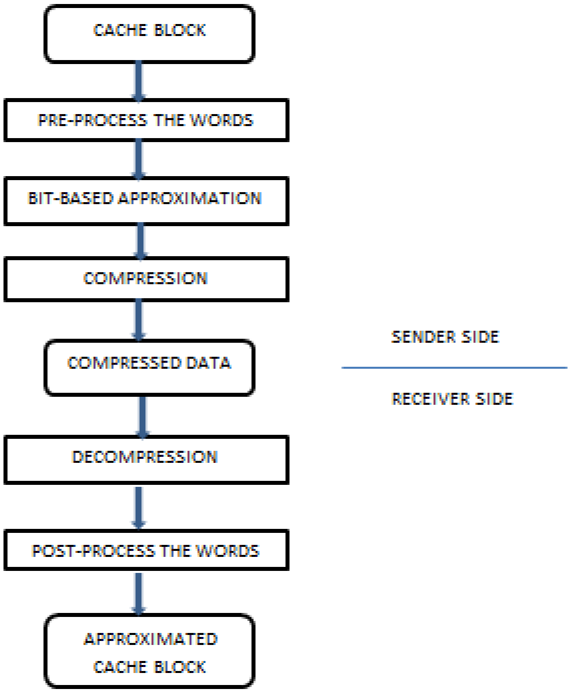


Figure 4-3 Flowchart of the BAXX Approach

4.2.1. **BAXX Technique**

For the BAXX technique first the number of bits that can be approximated per word needs to be found out. As a transpose is being applied on the entire cache block, the number of bits that can be approximated will be the minimum number of bits that can be approximated across all the words of the entire cache block. Once this is found out, we calculate the number of rows of the new transposed cache block that can be approximated; this is done as we first try to approximate only the LSBs followed by more significant bit. For example, if the number of bits that can be approximated is 2 then only the first row of the transposed cache block, which is the row containing the bits of each word of position 0 and position 1 can be approximated. As all the bits of one row hold the same weightage any of the 32 bits of the new transposed word can be approximated hence inherently enhancing the compressibility of the words. The BAXX technique has been implemented for both the FP and the DICT compression techniques.

4.2.1.1. FP-BAXX

In this technique, we use BAXX method on top of the FP-COMP technique. Upon finding the minimum number of bits that can be approximated across all the words of the cache block, the new transposed cache block is approximated row by row. In FP-BAXX we approximate allowed bits to 0 in order to increase the compressibility of the data. Once the don’t care bits of the word are determined, the rest of the word is matched to a pattern corresponding to the entries in the PMT. If there is a frequent pattern hit, then we compress that particular word.

4.2.1.2. DICT-BAXX

In this technique, we use BAXX method on top of the DICT-COMP technique. Upon finding the minimum number of bits that can be approximated across all the words of the cache block and then transposing the cache block, the new words are approximated one after another. This is done by comparing each word to the PMT and checking if there is a pattern match, if there is a pattern match then the word is approximated to that pattern. If this pattern appears frequently then the PMT is updated. In the case of DICT-BAXX approximation and compression are tightly bound together.

5. EXPERIMENTAL EVALUATION

In the field of Computer Architecture research modeling architectural hypothesis in software-based simulators is widely used. It helps in evaluating the design of a model before building an expensive hardware system. Building hardware systems can be expensive and if there is a fault or a change needs to be made to the design, then the entire hardware needs to change, and this can be a tedious and expensive task. Simulators on the other hand are low cost and can be changed as per the new requirements of the design. Simulators are easier to debug and also provide performance metrics.

A baseline configuration and suitable workloads need to be selected in order to implement and test the design. Usually the metrics such as performance that are obtained from the baseline configuration are used to compare against the new design, to see if the new design does indeed provide better performance or any other metric being tested. Selecting a simulator needs to be done with utmost care and this is due to the fact that there are various simulators that cater to different needs, and based upon the application and need a simulator must be chosen to evaluate the design.

5.1. Simulator Configuration and Methodology

For evaluating the Bit-Based Approx NoC framework we use an in-house network simulator. We implement DI-BAXX and FP-BAXX mechanisms in the in-house simulator. For detailed impact evaluations the in-house NoC simulator is used with the default error threshold set to 10% and percentage of approximable packets set to

75%. Further sensitivity studies have been carried out by varying the error threshold and the approximable packet ratio. In order to evaluate the impact of the work on the overall application error, the Pin [31] tool has been used.

Table 5-1: Approx-NoC Simulator 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	8×8 2D-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

The BAXX implementation uses the knowledge of the data type of the variables in each benchmark. It is assumed that the Approx-NoC framework is aware of the data type of the cache block that is going to be compressed. The Approx-NoC simulator Configuration and NoC parameters are listed in Table 5-1.

In order to evaluate the proposed work benchmarks from the PARSEC [32] suite have been used as they have been used in the past to evaluate work related to approximate computing. In order to evaluate our work, we perform trace-based simulation. Trace-based simulation is one in which the model’s inputs are derived from a sequence of observations made on a real system. We run the benchmarks using gem5 [33] to collect the communication traces for the region of interest which are then fed into the NoC simulation environment and are then simulated for 100 million cycles for detailed evaluations of the proposed mechanisms.

6. RESULTS AND ANALYSIS

This chapter presents performance evaluation of the *Bit-Based Approx-NoC framework* using benchmarks from different application suites and synthetic workloads. First we analyze the performance impact of the BAXX technique on the average packet latency, compression ratio and reduction in flits injected. Following this the chapter presents the use of synthetic workloads to evaluate the impact of the technique on the network throughput. We then perform sensitivity studies in order to put the technique under test under different conditions of error threshold and approximable data packet ratio. Following this the chapter dwells into the impact of the technique on the application errors.

6.1 Performance Analysis

6.1.1 Average Packet Latency

Figure 6-1 shows the average packet latency comparison, in an 8x8 mesh NoC, for the two implementations of the BAXX technique which are FPC-BAXX and DICT-BAXX. Across all benchmarks the DI-BAXX technique reduces the overall latency by 4% with respect to the baseline and 1% with respect to DI-COMP mechanism. The average packet latency of DI-VAXX is similar to that of DI-BAXX. It is observed that the latency reduction percentage is not proportional to the percentage of flits reduced; this is due to the fact that the network is not getting congested. It is not getting congested as the injection rate of the packets is quite low, so the queuing latency of each of the baseline techniques for all the benchmarks is low and so the latency reduction cannot be observed. However, if the injection rate is increased then there will be congestion in the network and so the impact of the approximation techniques can be seen. This is further studied and analyzed in section 6.2.

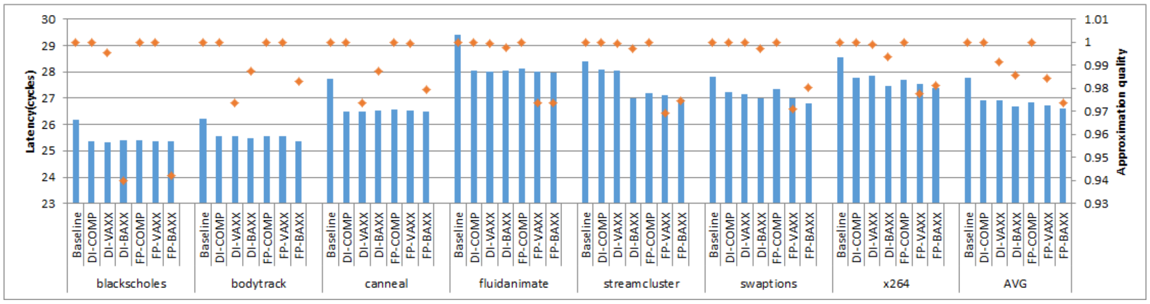


Figure 6-1 Average Packet Latency and Overall Approximation Quality

6.1.2 Approximation Effectiveness

The reduction in the traffic load is shown by plotting the number of data flits injected under each Approx-NoC mechanism in Figure 6-2. The DI-BAXX method reduces the number of data flits injected by 4% and 54% compared to DI-COMP and baseline respectively. The DI-BAXX method also reduces the number of data flits injected by 3.6% compared to the DI-VAXX method. Similarly, the FP-BAXX method reduces the number of data flits injected by 11% and 57% compared to the FP-COMP and baseline. The FP-COMP method also reduces the number of data flits injected by 6.3% as compared to the FP-VAXX method.

Figure 6-3 depicts the effectiveness of bit-based approximation in improving the compression ratio. DI-BAXX and FP-BAXX improve the compression ratio by 4.2% and 12% as compared to DI-COMP and FP-COMP respectively. The DI-BAXX and FP-BAXX mechanisms also enhance the compression ratio by 3.7% and 6.7% as compared to the respective value-based mechanisms which are DI-VAXX and FP-VAXX.

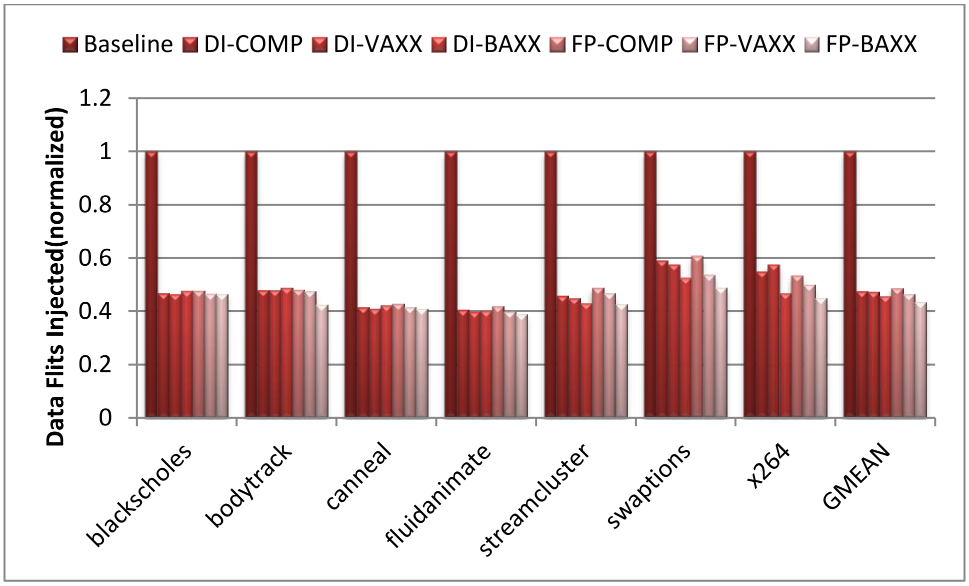


Figure 6-2 Reduction in Number of Injected Flits

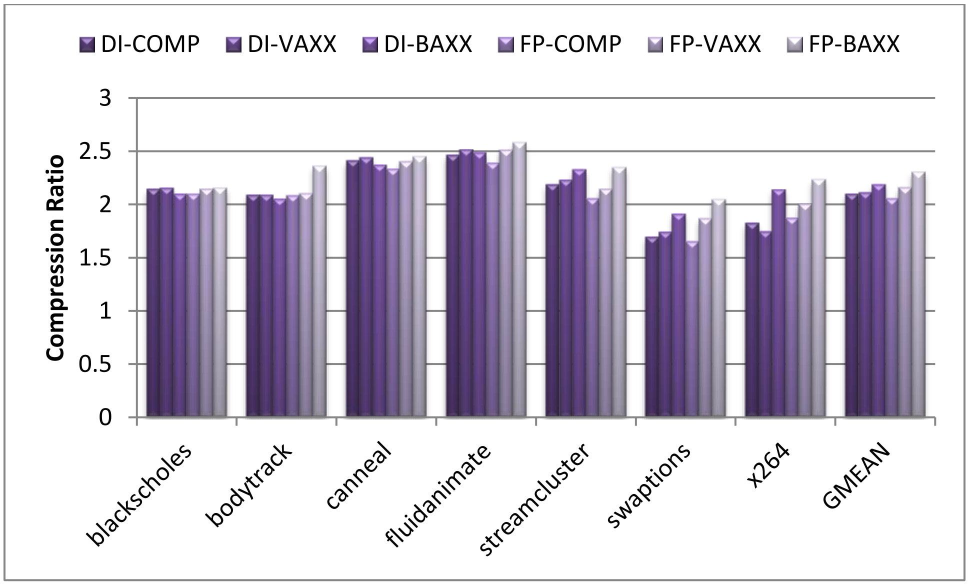


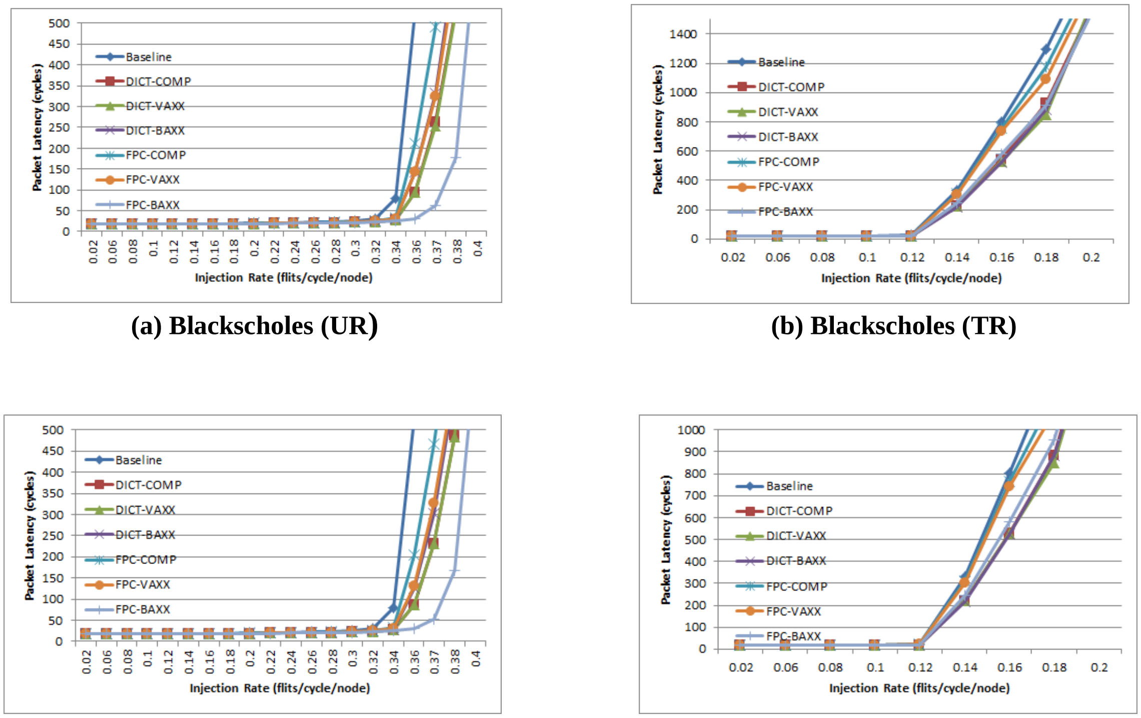
Figure 6-3 Compression Ratio

6.1.3 Data Value Quality

Figure 6-1 also shows the data value quality after approximation for all the benchmarks. Before approximating each word, the error is checked against the error threshold. However, each word is approximated to a different degree and so the data error, that is the difference in the error between the actual word and the approximated word varies from word to word. It can be observed that the data quality is close to 97% for almost all the benchmarks and this is due to the fact that only part of the data word is approximated, and the other remaining portion of the word is compressed without any error and hence the original word and the approximated word are close to each other in value proximity.

6.2 Throughput Analysis

In order to analyze the impact of the BAXX technique on the network throughput we have used synthetic workloads. Figure 6-4 depicts the throughput of the BAXX mechanism of the Approx-NoC framework compared against the Baseline, DI-COMP, FP-COMP and also the two already existing VAXX mechanisms. The plots depict the data traces for two benchmarks; blackscholes and streamcluster for two traffic patterns which are Uniform Random (UR) and Transpose (TR). When compared to the compression schemes BAXX improves the throughput by 14% for UR and 12% for TR. This is owed to the fact that the number of injected flits have reduced hence the injection load reduces due to data approximation. From the graph it can be seen that the throughput improvement is a lot more than the packet latency improvement shown in Figure 6-1. This is due to the fact that the injection rate has been increased gradually and hence congestion can be observed in the network and so the actual benefit of approximation can be seen. From our observations it can be seen that the FP-BAXX mechanism works best, especially for UR traffic pattern as can be observed in Figures 6-4 (a) and (c).



(a) Blackscholes (UR)

(b) Blackscholes (TR)

(c) Streamcluster (UR)

(d) Streamcluster (TR)

Figure 6-4: Throughput analysis with Different Benchmark Data Traces under Uniform Random (UR) and Transpose (TR) Traffic patterns

6.3 Sensitivity Studies

The sensitivity of the two BAXX mechanisms to the error threshold and percentage of approximable packets is shown in this section.

6.3.1 Error Threshold

Figure 6-5 shows the average packet latency for all the benchmarks across the Bit-Based Approx-NoC framework by varying the error threshold percentage. As the error threshold is increased from 5% to 10% (default) to 20%, the impact of the BAXX mechanism for the Approx-NoC framework on the packet latency decreases, this is owed to the fact that higher the error threshold, more bits of the data words can be approximated and hence increasing the chance of approximate matching. It can be observed that there is significant improvement in the packet latency with increase in the error threshold in certain benchmarks such as x264 and Swaptions across both the FP-based technique as well as the DICT-based technique. However, in the Canneal benchmark it can be observed that the packet latency is increasing with the increase in the error threshold and this can be owed to the fact that for the Canneal benchmark the number of flits injected indeed increases with approximation.

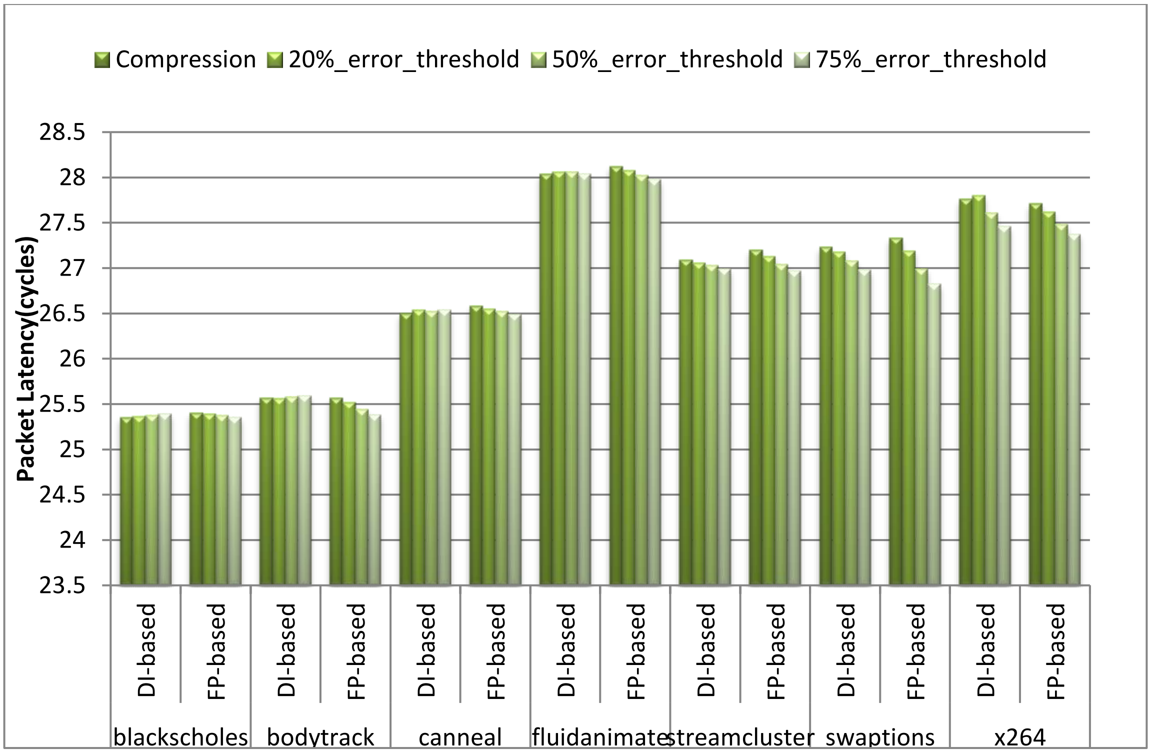


Figure 6-5 Error Threshold Sensitivity Analysis

6.3.2 Approximable Packets Ratio

Figure 6-6 shows the average packet latency for all the benchmarks across the Bit-Based Approx-NoC framework by varying the percentage of approximable packets. The packet latency reduces as the percentage of approximable packets increases from 20% to 50% to 75% (default). This is owed to the fact that increasing the approximable packets increases the chance for approximate matching of the data. We can observe significant improvement for Swaptions and x264 benchmarks with both the DI-BAXX and the FP-BAXX techniques. The rest of the benchmarks do not show a significant improvement in the latency as the percentage of approximable packets is increased. This is due to the low queuing latency in the NoC and small data-to-control packet ratio for the benchmarks, leading to a lower impact on the overall network latency.

6.4 Full System Output Accuracy Analysis

In order to analyze the impact of *Bit-Based Approx-NoC* on the entire system Pin [31] Instrumentation framework has been used. This section presents the overall application output errors across all the benchmarks. The BAXX technique has been implemented on top of a coherent cache simulator tool. In order to analyze the impact, we model a system with 16 cores and each core has a 64KB two-way L1 private data cache of cache line size 64 Bytes. The system emulates packet response whenever there is a miss that requires a data response from another node.

Application output accuracy for all benchmarks is shown in Figure 6-7. We observe that with the predetermined 10% error threshold all the benchmarks output errors fall within this bound except for blackscholes and streamcluster. Hence, we can state that there needs to be a tradeoff between the output accuracy and the performance benefits.

Figure 6-7 also shows the application output accuracy with different error thresholds. For all cases it is well within the error threshold bounds and hence it is observed that it is possible to achieve high throughput and low latency by exploiting approximate computing while maintaining an acceptable amount of error quality.

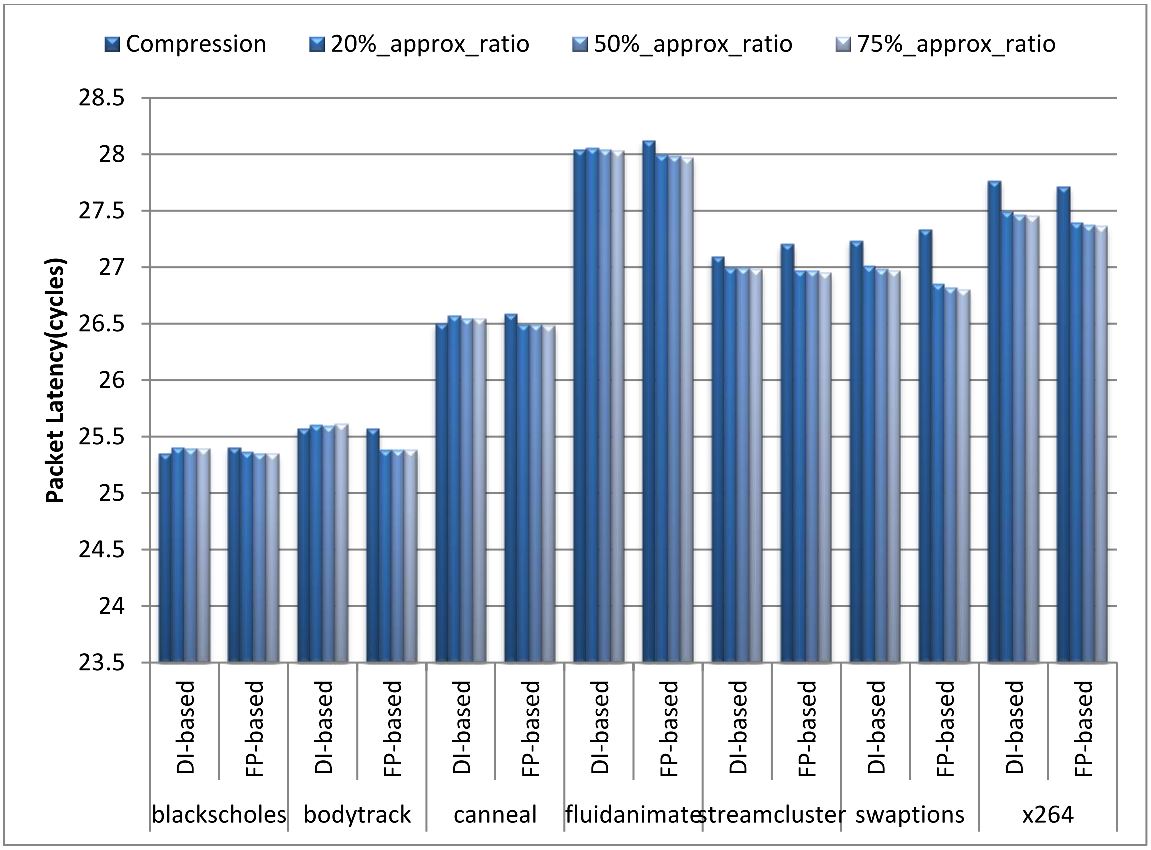


Figure 6-6 Approximable Packets Ratio Sensitivity Analysis

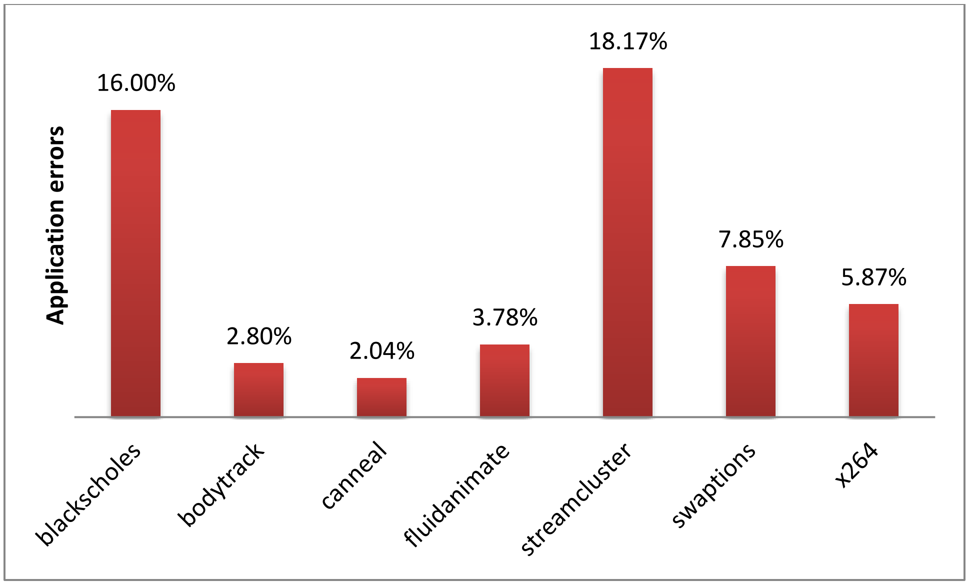


Figure 6-7 Application Output Accuracy

7. CONCLUSION

In this work I propose Bit-Based Approximation for Approx-NoC, a hardware data approximation framework for high throughput NoCs in the memory intensive big data era. The work presents a Bit- Based approximation logic that can be used as a plug and play module with an underlying compression technique. This work uses two underlying compression techniques; frequent pattern compression and dictionary-based compression. The work evaluates the low-cost micro architectural implementation of the BAXX technique.

It has been observed that the FP-mechanisms achieve higher approximation rate as compared to the DI-mechanism, however the DI-mechanism outperforms the FP-mechanism when there is some amount of data repetition. The best latency reduction achieved is 11% when compared to the existing compression techniques. The evaluation using synthetic workloads shows the best throughput improvement to be 14%.

Relaxing the constraints of applications is indeed a viable solution for high performance computing. Approximate computing in NoCs exploits this fact and helps to meet the high bandwidth requirements of NoCs in the big data era.

REFERENCES

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

[2] 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.

[3] 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.

[4] 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

[5] 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.

[6] 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

[7] 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* (MICRO45). 449–460.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

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

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

[16] 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.

[17] 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.

[18] 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.

[19] Rahul Boyapati, Jiayi Huang, Pritam Majumder, Ki Hwan Yum, Eun Jung Kim, Texas A&M University. 2017. g. In *Proceedings of the 44th Annual International*

Symposium on Computer Architecture (ISCA-44). "APPROX-NoC: A Data Approximation Framework for Network-On-Chip Architectures".

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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* (HPCA21). 603–614.

[28] 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.

[29] Jungrae Kim, Michael Sullivan, Esha Choukse, Mattan Erez. 2016. In *Proceedings of the 43rd Annual International Symposium on Computer Architecture* (ISCA-43). “Bit-Plane Compression: Transforming Data for Better Compression in Many-core Architectures”.

[30] 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).

[31] 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.

[32] Christian Bienia. 2011. “Benchmarking Modern Multiprocessors”. Ph.D. Dissertation. Princeton University.

[33] 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.