AN ABSTRACT OF THE THESIS OF

Abhishek Ramamurthy for the degree of Master of Science in Electrical and Computer Engineering presented on November 19, 2018.

Title: Characterizing On-Chip Traffic Patterns in Throughput Processors: A Deep Learning Approach.

Abstract approved:

_____

Dr. Lizhong Chen

The machine learning and deep learning models have been very lightly explored in analyzing the behavior of On-Chip network traffic. These models have proven their potential in pattern recognition, classification *etc*... In this paper we analyze the spatial pattern that each workload exhibits in its life cycle during execution. We address the problems with current studies in analyzing workload behavior and provide a refined path with less variables to tackle while analyzing the behavior. We have identified the abstraction at which analyzing the traffic behavior will result in low loss of information and could still use image recognition like approach to solve the On-Chip traffic pattern characterizing problem.

©Copyright by Abhishek Ramamurthy
November 19, 2018

Characterizing On-Chip Traffic Patterns in Throughput Processors: A Deep Learning Approach

by
Abhishek Ramamurthy

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented November 19, 2018
Commencement June 2019

Master of Science thesis of Abhishek Ramamurthy presented on November 19, 2018

APPROVED:

Dr. Lizhong Chen, representing Electrical and Computer Engineering

Head of the School of Electrical Engineering and Computer Science

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Abhishek Ramamurthy, Author

ACKNOWLEDGEMENTS

I would first like to thank my thesis advisor Dr. Lizhong Chen of the Electrical and Computer Engineering at Oregon State University. Professor Chen always encouraged me in coming up with new approaches in solving the problem and supported me with needed guidance when I had any question related to research or writing. He consistently allowed this paper to be own my work but steered me in right direction whenever he thought I needed it.

I would like to express my deep gratitude to Yunfan Li PhD candidate at Oregon State University, for his continuous support in completing this project. The Idea in consideration of this work was first proposed by him, and he always gave the guidance needed in understanding the expected result of the project. Along with giving me the project guidance, he also helped in understanding how to present the work done and gave valuable inputs in making this thesis a presentable work.

My friends have supported me, in listening to all the problems that I have during the completion of the project. They have even spent time discussing about ideas on how to solve the problems and even suggested approaches that could best fit the problem I was trying to solve. Risheek, Anurag and Durga have always given me feedback and suggested techniques to try, in searching better results.

Finally, I must express my very profound gratitude to my parents for providing me with unfailing support and continuous encouragement throughout my years of study and writhing this thesis. This accomplishment would not have been without them Thank you.

Abhishek Ramamurthy.

CONTRIBUTION OF AUTHORS

The end goal of the project is to formulate a generic intelligent model to help Network on Chip (NoC) designers in deciding the kind of architecture to employed to achieve better performance for the application under consideration. In this project, we have proposed Traffic Pattern (TP) of throughput processors as an independent entity in studying the performance of the throughput processors. The measure of similarity in TP between high performance computing applications (HPC) can be used a metric to in designing NoC architectures to achieve better performance in applications with high similarity on a common hardware. To collect the NoC traffic data needed for analysis we have used GPGPU-Sim simulator. We have successfully achieved transforming the normalizing the collected data to be visually analyzable. The visual analysis of the collected dataset has revealed that there exists similarity in traffic pattern between HPC benchmarks in consideration (RODINIA, PARBOIL and NVIDIA SDK). We have trained supervised machine learning and deep learning models on the visually analyzable data set and validated the visually obtained labels using un-supervised learning models.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1: Introduction

The advent of machine learning and deep learning models have paved a way to find solutions to the problems which exhibit characteristics that are unique and specific to the data set in consideration. Image recognition is one such application in the machine and deep learning models. These models have become so matured that even slightest of variation could be picked when trained accurately for a data set. This trending methodology has given an alternative way for solving problems which deal with identifying patterns and enabling use of computers to extract pattern information, in contrast to establishing complex mathematical relations to achieve the same. Characterizing an On-Chip traffic can be treated like the image recognition application by representing characteristic feature in a 2D space.

In recent days research on characterizing workload behavior to analyze the traffic pattern of On-Chip network is focused towards analyzing group of metrics such has dynamic instruction count, memory usage, number of CTAs and PTXs [2][3][8]. Since there is no one area to be focused, it is not feasible to represent instruction count, number of CTAs, PTXs and memory usage as 2D matrix to employ Image recognition methodology to extract unique features. In addition, each pixel in an image has relationship between its neighbors. With the above metrics it is difficult to establish a reasonable 2-Dimensional relationship like that of an image, to employ image recognition like approach to analyze patterns.

The Processing Elements and Memory controller's arrangement in On-Chip network is like that of an Image when represented as matrix, i.e. each pixel position as Processing elements in relation with Memory controllers (adjacency matrix of a graph). Instead of light intensity each position in the graph is filled with memory requests from Processing Elements during execution of a given application. Creating an adjacency like matrix between Memory Controllers and Processing elements with number of memory requests as data, it is easier to visualize the local hot spots. The adjacency matrix can be transformed to a heat map image. Transforming memory requests matrix to image opens the

opportunity in solving the pattern identification using methods like that of image recognition application.

Optimizing On-Chip network design is a critical part to obtain better performance out of the throughput processors. Critical resource distribution to cater memory requests plays a major role in reducing traffic congestion in On-Chip network. The applications executed on throughput processors have assumed to exhibit Many-to-few-to-Many memory request pattern in recent days research. This approach doesn't give any insight on the characteristics that could be extracted to optimize the On-Chip network design for application specific designs. In this paper, we study the memory request pattern behavior at different granularities to come up with a reasonable level to study the information with minimal loss. This will help us analyze the traffic pattern behavior at better extent to identify hot spots generated by application over the execution period and allocating more resources to these hot spots for faster processing of memory requests.

Chapter 2: Background and Motivation

2.1 Limited existing characterization

Recent studies toward analyzing workloads of high-performance computing treat the traffic pattern of all benchmarks as strict many-to-few-to-many (M2F2M). The memory requests of Gaussian benchmark [4] is represented in Table 1, the X-axis of the table describes different processing elements (PEs) and Y-axis denotes the memory controllers (MCs). This kind of table in the following sections are referred as memory request tables (MRTs). Each number that is populated in a cell of Table 1 is accumulated over the execution cycle time of Gaussian to quantitatively exhibit the total communication between a pair of PE-MC. The pattern that is demonstrated in Table 1 is inferred as M2F2M traffic pattern. However, if the collection of traffic pattern statistics is fine-grained, *i.e.* if traffic patterns are collected over a few hundreds of cycles during the execution of a benchmark, the situation could be more sophisticated than a naive M2F2M pattern. Table 2 shows the request traffic in the kernel-3 of Gaussian benchmark [4] during 600-700 cycles. In these 100 cycles, only PE-19 sends requests to MC-1, MC-2 and MC-3. This example indicates that in some periods, a Few-to-Many-to-Few pattern is formed in the NoC of a throughput processor. Preliminary experimental results across wide range of benchmarks from Rodinia [4], NVIDIA SDK [1] and Parboil [11] clearly demonstrate that the M2F2M pattern does not always exist especially when fine-grained collection is employed. It is thus the NoC designs of throughput processors which primarily focus on M2F2M pattern do not always serve the purpose of complete coverage to achieve optimized resource utilization, reduction in occupancy area and minimized power consumption.

A majority of the previous studies propose metrics which are driven by the factors such as number of PTXs, CTAs, dynamic instruction count, memory usage which are architecture independent as claimed by Goswami *et al.* and architecture dependent metrics proposed by Adhinarayan *et al.* for a given benchmark using unsupervised learning like principal component analysis

(PCA) [13] and hierarchical clustering method to identify similarities across benchmarks in the same suites [2, 3, 8]. . The studies till now, have not considered analyzing the traffic flow in NoC as it is classified as M2F2M pattern. In the proposed approach, NoC traffic is dissected to identify the individual components which make up the M2F2M. Another potential NoC architecture design parameter that could impact the performance is traffic pattern, current research for analyzing the workloads do not consider this parameter when characterizing.

Table 1 Cumulative Pattern of Gaussian benchmark [4]

	PE1	**....**	**PE30**	**PE31**	**....**	**PE56**
MC1	1	....	1	3	....	5
MC2	3	....	2	3	....	2
MC3	4	....	3	6	....	2
MC4	1	....	2	5	....	3
MC5	0	....	1	1	....	1
MC6	2	....	0	2	....	0
MC7	4	....	2	4	....	0
MC8	0	....	2	5	....	7

This paper emphasizes analyzing the common features of traffic pattern which exist among the benchmarks presented in Rodinia, NVIDIA SDK and Parboil. The experimental results of the paper serve as a purpose to design NoC architectures optimized for the identified common traffic patterns, to achieve improved performance in areas of power consumption, resource utilization and reduced on-chip area for NoC.

2.2 Benefits of characterization

Traffic patterns of a benchmark in throughput processors can be considered as a single metric in the system, which directly exposes the intensity of the reply packets that are flowing across the NoC. By characterizing this metric to a limited group based on the common features that are observed, it is possible to cluster the benchmarks into a single or multiple group obtained by treating traffic pattern as a single metric. This opens up the opportunity of designing a

NoC optimized to specific group to derive maximum performance out of throughput processors for the specified benchmarks.

Table 2 Kernel-3 Pattern of Gaussian benchmark collected between 600-700 Execution cycles [4]

	PE1	**....**	**PE18**	**PE19**	**PE20**	**....**
MC1	0	....	0	1	0	....
MC2	0	....	0	2	0	....
MC3	0	....	0	2	0	....
MC4	0	....	0	0	0	....
MC5	0	....	0	0	0	....
MC6	0	....	0	0	0	....
MC7	0	....	0	0	0	....
MC8	0	....	0	0	0	....

The MRTs collected from the tested benchmarks clearly illustrate some common patterns which will be presented later. On the basis of the common patterns, the MRTs are capable of classifying to a few categories. Hence, the NoC in throughput processors can be optimized based on the categories instead of based on naive M2F2M pattern. Thus, the designers are able to focus on some specific areas in the NoCs to emphasize the features of those non-M2F2M patterns, and the system performance can be improved significantly as the NoC tuning is more fine-grained.

In a throughput processor, request traffic pattern and reply traffic pattern are symmetric (i.e. request packets are same in number as reply packets). This paper analyzes and characterizes the traffic pattern of throughput processors by considering the reply traffic that is sent from each MC to each PE. The collected traffic pattern is analyzed at defined stages of granularity *i.e.* towards the completion of execution, defined execution cycle period and at CUDA kernel of a given benchmark. The best granularity is then selected for identifying a set of basic traffic patterns for varies of benchmarks. Our testing results obtained by studying reply traffic is applicable for request traffic as pattern is symmetric, thus it serves as a complete package to analyze the overall traffic pattern characteristic of throughput processors.

However, there is very limited importance given to traffic patterns in characterizing the workload in recent research. This gives the opportunity to understand the impact of traffic patterns on NoC performance. Although there are many different mathematical approaches that can be potentially used for identifying similarities *i.e.* to group the observed traffic patterns on some similar characteristic features. The particularity and complexity of traffic pattern data in throughput processors drive the selection of the mathematical method to be more challenging.

Table 3 Kernel-1Pattern of Gaussian benchmark [4]

	PE1	**PE2**	**PE3**	**PE4**	**PE5**	**...**
MC1	0	5	0	0	0	...
MC2	0	3	0	0	0	...
MC3	0	2	0	0	0	...
MC4	0	2	0	0	0	...
MC5	0	2	0	0	0	..
MC6	0	2	0	0	0	...
MC7	0	2	0	0	0	...
MC8	0	2	0	0	0	...

Table 4 Kernel-3 Pattern of Gaussian benchmark [4]

	PE1	**...**	**PE19**	**PE20**	**PE21**	**...**
MC1	0	...	4	0	0	...
MC2	0	...	3	0	0	...
MC3	0	...	2	0	0	...
MC4	0	...	2	0	0	...
MC5	0	...	2	0	0	..
MC6	0	...	2	0	0	...
MC7	0	...	2	0	0	...
MC8	0	...	2	0	0	...

2.3 Challenges

The traffic patterns of benchmarks exhibit variety of characteristics for patterns collected at different granularity. For instance, the traffic pattern collected over a range of execution cycles is not same as that of cumulative pattern collected at the completion of execution. Similarly, the traffic pattern collected at individual kernel rarely exhibit similarity to the cumulative pattern. As will be

discussed in Section III, the traffic pattern collected at kernel level retains the important characteristic features for a given benchmark and is applicable across the benchmark suites.

In this research, the MRTs collected are treated as matrices due to the number of options available for processing to extract similarities. Transforming matrices to a suitable form provides options for visual analysis and could be used as an input for deep learning models.

The problem to be addressed once having the accurate granularity for collecting the MRTs is on grouping similar data set based on the unique characteristics observed. The existing mathematical methodologies for finding similarities across matrices like correlation, identifying eigen vectors and single value decomposition (SVD) cannot be appropriately employed to classify MRTs. The correlation co-efficient of MRTs is very low, MRTs which look similar are not relative in their position of similarity occurrence thus Eigen Value changes for similar MRTs and SVD is suited in identifying similarity in a given matrix and not between two matrices. Thus, leading to usage of advanced methods which are proven in extracting and analyzing similarity information.

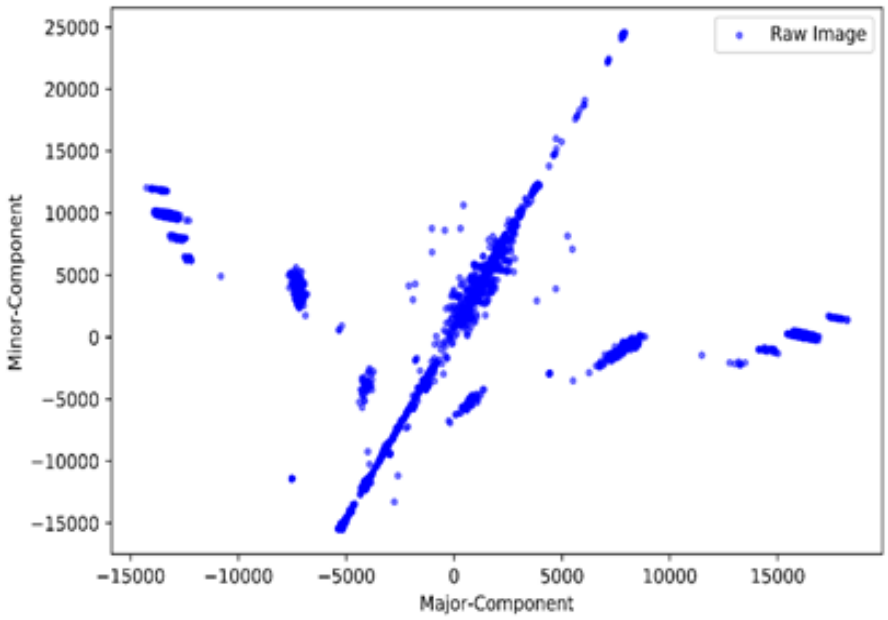

Figure 1 Plot of Prinicpal Component Analysis by transforming raw traffic pattern to a suitable form.

The correlation coefficient is used to measure the similarity between matrices. However, the MRTs collected are not correlated as there are cases where we observe the MRTs of similar type to be not exactly same due to the mis-match in the position of occurrence (i.e. the MCs and PEs involved are completely different). In Table 3 and Table 4 it is clearly evident that traffic pattern of Kernel-1 of Gaussian [4] is like that of Kernel-3. Although the kernels 1 and 3 have similarities in their traffic pattern, the PEs involved in generating the traffic pattern are not the same, thus when using correlation to find similarities between kernel-1 and kernel-3 the mathematical result would show them as non-similar traffic pattern since correlation is position dependent. The traffic patterns are not square matrices and the matrix meaning changes due to the linear and horizontal shift that could be observed for matrices of similar type

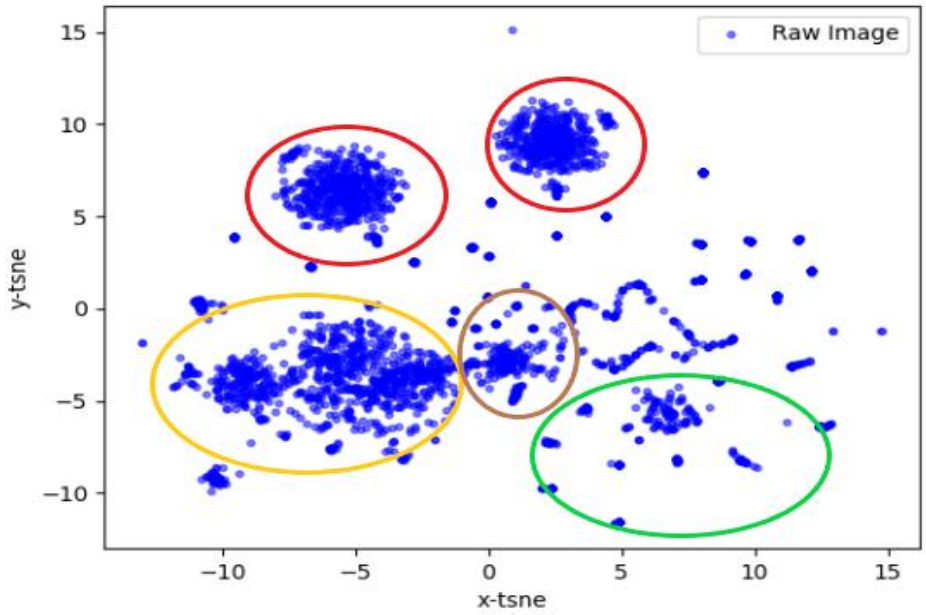

Figure 2 Plot of T-Distributed Stochastic Neighboring embedding by transforming raw traffic pattern to a suitable form.

thus employing eigen method would yield different characteristic roots for similar traffic patterns.

The existing unsupervised machine learning model such as PCA, t-distributed stochastic neighbor embedding (T-SNE) would be an alternative approach for one to identify similarities between matrices which cannot be observed with

human visual inspection and obtain groups of similar MRTs. The results obtained by using PCA and T-SNE do not yield good results as can be observed in Figure 1and Figure 2. PCA results presented in Figure 1 do not help us in identifying principle traffic pattern types as majority of the points lie on a common straight line. After visual inspection of the data points on the straight line, it appears that there are several mixed traffic patterns that exist. This means the straight line does not represent a group of similar traffic pattern. The same circumstance also happens in other small clusters appear in Figure 1. Thus, usage of PCA to identify clusters on raw MRTs is not a feasible solution. The T-SNE results referenced in Figure 2 does a better job than PCA, there are different clusters appear in the figure. However, in our visual inspection to all MRTs, the clusters which circled in red belongs to the same type, but other clusters circled in yellow, brown and green consist of various traffic patterns. Therefore, the T-SNE algorithm still cannot provide a reasonable result to the traffic pattern classification task.

To use any supervised learning methodology such as support vector machines (SVM) or convolutional neural network (CNN), the MRTs should be transformed to a suitable format which can be used as an input and must be labeled before training in SVM or CNN. SVM works on the principle of curve fitting in a high dimensional space with pre-verified and labeled dataset but does not provide any feature extraction benefits which could be used for label validation to catch missing labels or identify wrong labels as the pre-defined labels are collected manually.

CNN establishes classification relationship by capturing features using several filter masks and moving the masks across the input matrix to produce corresponding output feature matrices. A simple CNN usually consists of many convolutional layers, pooling layers and a fully connected layer (neural network layer) which uses the output feature matrices to produce multi-label classification. Each convolution layer extracts some features from an image which are representative of a part of the image, such as a unique pattern, edge

of main object *etc.* The feature extraction is done by moving a number of definitive sized filters over the image. The extracted features are fed into a

Table 5 Cumulative Pattern of Sorting Networks benchmark [1]

	PE1	PE2	PE3	PE4	PE5	...
MC1	86523	85687	86495	85261	86167	...
MC2	86246	85417	86224	84988	85897	...
MC3	86823	85975	86797	85546	86455	...
MC4	86264	85435	86242	85006	85915	...
MC5	85884	85608	85248	85950	85336	..
MC6	85168	85349	84988	85688	85077	...
MC7	86486	86194	85838	86559	85921	...
MC8	85861	85585	85225	85927	85313	...

pooling layer to down sample the feature map matrix generated from previous convolution layers, to achieve reduction in dimensionality and avoid risk of over fitting the model. After passing through several convolutional and pooling layer combinations, the final output matrices serve as input of the fully connected layer to make the final classification.

This study analyzes the benchmarks presented in RODINIA [4], NVIDIA SDK [1] and PARBOIL [11], transform the traffic pattern of each benchmark to a suitable format for deep learning, and identify the common features in these traffic patterns. We extend our study to analyze the behavior the same benchmarks and the effects observed on the traffic patterns due to architectural changes introduced and present a common subset of features which is used to

Table 6 Pattern Collected between 75k-85k execution cycles of Sorting Networks benchmark [1]

	PE1	PE2	PE3	PE4	PE5	...
MC1	0	4	10	11	6	...
MC2	0	5	9	10	6	...
MC3	0	4	9	11	5	...
MC4	0	4	9	10	6	...
MC5	0	4	8	11	5	..
MC6	0	3	10	11	8	...
MC7	0	5	8	12	8	...
MC8	0	5	11	11	8	...

represent majority of the traffic patterns observed as combination of identified basic types. This study will help the further research in designing NoC which

is optimized to the traffic pattern behavior discovered to get optimized performance

Chapter 3: Transforming Traffic data for Deep Learning

This section discusses the traffic pattern data collecting and transforming process. The traffic pattern data is collected from a cycle-accurate simulator GPGPUSim in which an ideal NoC configuration is employed to remove the impact from the architecture of interconnect network. As mentioned before, the behavior of the traffic pattern is decomposed and analyzed at cumulative stats collected at the end of execution of each benchmark (cumulative pattern), at a defined range of execution cycles (cycle-ranged pattern) and at a granularity of kernel (kernel pattern). The experimental results show that kernel pattern is the most suitable granularity to reflect runtime traffic flow in a benchmark without losing any detailed traffic information. The aggregated data also needs to be augmented to images to fulfill the input requirement of a deep convolution network.

First, an ideal NoC (any packet can travel through a network in 1 cycle without any conflict) is configured in experiments based on impact consideration. Interconnection network in throughput processors has various topologies, routing policies, scheduling or allocating schemes. All the configurations have significant performance effect on a throughput processor system. To better focus on the traffic statistical data and to lower the impact from NoC in a throughput processor, a perfect interconnection network is thus employed during the following experiments.

Before digging into the data analysis, it is imperative to have a proper granularity to collect end-to-end communication data firstly as different granularity could emerge different traffic pattern. This research is eventually for understanding runtime traffic status so that it is possible to optimize interconnection network based on this information. Therefore, an accurate but not redundant granularity is urgently needed. Previous research considers the traffic flow in a throughput processor mainly as M2F2M because the Cumulated Pattern in most of benchmarks show this feature. For example, the Table 5 is a Cumulative Pattern collected at the completion of SortingNetwork [1], which shows an exact M2F2M pattern. Other benchmarks explicitly display the same

result as SortingNetwork as well. However, these results (tables) do not comprehensively offer detailed runtime traffic status. Table 6 gives the traffic behavior from cycle 75000 to 85000 of SortingNetwork. This preliminary test presents that during the execution of SortingNetwork, the intermediate traffic flow could be different from the Cumulated Pattern. Not only SortingNetwork, but also some other benchmarks have the same characteristic. Therefore, a more fine-grained granularity should be proposed to provide accurate runtime traffic information.

The most fine-grained granularity is cycle accurate which means that the MRTs would be generated in each execution cycle. However, a benchmark could have thousands of thousands cycles and not every cycle has MRs, some cycles have only few MRs. This naive approach of analyzing every cycle would lead to a huge dataset and some of the data points are meaningless. Although considering this research is going to use CNN as a classifier so the dataset for training a CNN model should be as large as possible, the data redundancy should also be considered and kept as low as possible to avoid CNN model over-fitting problem, this can also be considered as over-fine-grained issue. The preliminary tests on several benchmarks show that continuous cycles would have same traffic performance; in other words, all these cycles can be combined and be treated as one MRT. Therefore, to efficiently reduce the dataset redundancy, it is better to group continuous cycles together so that less duplicated data points are existing in the dataset.

There are three typical methods that can be employed for grouping cycles together. The first and the most accurate method is that manually track all cycles in each benchmark, then group those continuous cycles with similar traffic pattern together. After grouped, the new traffic pattern table is an accumulated table throughout these cycles. As each individual cycle in this group has the same traffic pattern, the newly generated cumulated table has exactly same traffic pattern as any single cycle in the group. However, the major drawback of this approach is the time consuming. To accurately group continuous cycles together, a manual action should be taken to monitor every cycle in each

benchmark. Considering that there are at least 30,000 more total execution cycles in each benchmark, the hours that are spent on this manual action is not acceptable. Since the most accurate method cannot be exploited due to the unsustainable high time consuming, an efficient cycle grouping scheme is needed. The second approach of grouping scheme is group cycles by a fixed number or fixed percentage of the total cycles. This fixed grouping scheme is much faster than the first one, but it is not accurate and may cause some information loss or mixture. This is because if a fixed number of cycles is decided *e.g.* 1000 cycles, the granularity of this number is various across different benchmarks. For example, in CFD benchmark, it has in total 2263660634 cycles, which 1000 is only 4.41×10^{-5}% of benchmark. Therefore, a fixed number of cycles is not a feasible approach. An alternative method is to determine a fixed percentage of the total execution cycles. This method is to address the granularity issue that happens in the fixed number of cycles approach. With a fixed percentage, the granularity across all benchmarks is decided, but this scheme could lead to over-fine-grained issue. Taking Gaussian benchmark [4] as example, it has in total 27447 cycles, if the percentage is determined as 10%, then every 2745 cycles an MRT is generated. But the preliminary test on the Gaussian, during the 2745 cycles, only a few MRs is generated, which cannot be correctly recognized as a comprehensive pattern. The comprehensive approach for using number of cycles or percentage of total cycles is to decide number of cycles/percentages of total cycles for each benchmark. However, this is also a time-consuming approach as a proper number of cycles/percentages of total cycles should be decided by a manual action which performs on each benchmark. This manual action will take the same time as the first approach (manual group approach) since the proper number or percentage can only be determined after all cycles are evaluated individually. Therefore, these 2 approaches are either time consuming or cannot accurately reflect traffic pattern.

After carefully analyzing the preliminary results, the Kernel Pattern approach is proposed to address the previous 2 issues. Each MRT that is generated as a

cumulative traffic at each kernel execution ensures that the results collected are easy to analyze and consumes less time due to limited dataset. This approach gives the base for fixing the granularity for analysis, because usually every kernel in a benchmark has a unique task, such as loading data into the device memory, or processing a part of the computation in an algorithm (e.g. processing a part of a graph in Breath-First-Search algorithm). Therefore, within one kernel, the traffic flow is almost the same, so an MRT of a kernel has abundant information (number of request packets) to form an obvious traffic pattern. In another words, the Kernel Pattern represents the traffic flow within a program kernel, which in addition, if a NoC design is optimized to the traffic pattern of the kernel, the design is optimized to all program kernels with the same behavior. At the same time, there are more than 5000 kernels in total for all benchmarks, which indicates that even though the original data points (in terms of Cycle Accurate Pattern) are shrunk, there are still relatively large dataset that is enough for training a simple CNN model. Even though by employing the Kernel Pattern approach, a proper granularity of the traffic pattern is collected, the raw MRT cannot be directly used by any CNN model because of its specific characteristics which will be discussed in the following paragraph.

The input of a CNN model should be a matrix, the MRT serves this purpose. As introduced before, the first dimension of a MRT represents the MCs and the second dimension is the PEs, in the experiments of this paper, there are 56 PEs and 8 MCs, which implies that the dimension size of a MRT is 8×56. This dimension size is smaller than a typical input size of a CNN model, such as 224×224, and cannot be divided by a typical convolutional filter size perfectly. The smaller input size leads to obscure features in a traffic pattern which eventually considerably impacts the recognition accuracy of a CNN model. And non-divisible dimension size also affects the model suitability to this traffic pattern dataset. To address these 2 issues concurrently, a data augmentation for MRT should be proposed. Another major challenge that causes a MRT cannot be directly feed in a CNN model is data normalization. It is obvious that

different kernels in a benchmark have different density of the E2E traffic, and different benchmarks also have different number of kernels, with various of total traffic loads. For example, AlignedType benchmark [1] has about 1461367 cycles in its Kernel-275 while BFS benchmark [4] has only 9826 cycles in its Kernel-2. Since the traffic pattern is determined in a kernel-based unit, all kernels will form a dataset and be treated equally, this means that every kernel (MRT) is considered as an independent data point in the dataset. The criteria of a CNN model training dataset should have been normalized to reduce the effect of data and make the model focus on the data variation trend. The traditional data normalization scheme is based on the entire dataset, which is the scheme would firstly flatten all matrices in a dataset to a row vector, and then combine all row vectors together to be a new 2D matrix and finally normalize each column in this new matrix individually. This approach potentially treats each column in the new matrix as the same. However, in the traffic pattern analysis, every element only depends on its own vector, which is the original MRT, so the traditional normalization approach could add unnecessary information, even some misleading information after it is applied on the Kernel Pattern dataset. Therefore, the proposed augmentation scheme should not lose any information that appears in an MRT, because any information loss could potentially lead to wrong traffic pattern recognition in CNN model, and provide a proper data normalization method as well. An approach that could augment a matrix without losing detailed information is heat map coloring.

Heat map coloring scheme can achieve tinting every element in a matrix based on the matrix variation trend [5]. After applying heat map on a matrix, a new image would be generated where each color block represents an original element in the matrix as seen in Figure 10. The lighter a color block is, the higher the traffic intensity original element has. Therefore, by employing heat map scheme, all MRTs are transformed to images which contain a lot of color squares. This scheme can directly eliminate data normalization problem because all images are generated only based on the corresponding matrix, all color squares are in proportion to the original elements in the matrix. Therefore,

the generated images reflect the data variation trend in the corresponding matrices directly, but all data points now are ranged in 0-255. When an original element is transformed to a color square, the data is also augmented. The dimension size of the original matrix, as mentioned before, is 8×56, and each of the element is transferred to a color square with dimension size of 8×8. When heat map tinting is completed on a matrix, the size of the generated image becomes 69×495 which achieves data amplification without losing original information. Another benefit comes with heat map transformation is that the generated image is more convenient and intuitive to do manual visualized identification and validation.

When all MRTs are transformed to Heat Map Images (HMIs), half of the deep learning dataset is created. However, CNN model needs pre-labeled dataset to perform backpropagation training process. Therefore, a manual data labeling should be used firstly and then a scientific label validation approach is also employed to ensure the pre-label dataset is convincing.

Chapter 4: Traffic Type Identification

This chapter focuses on adding correct label to each data in the dataset. As emphasized before, any dataset that needs to be used for a CNN model has 2 major parts, one is the data itself (or matrix), the other is the corresponding label which stands for the category that this data (matrix) belongs to. It is important to have an accurate label for each data in a dataset because the training process (backpropagation training) relies on the loss between the predicted label and the ground truth label. Therefore, an accurate and correct ground truth label affects the final CNN model accuracy and reliability significantly. In this section, there are 2 steps to generate a convincing dataset with correct label for each data point. The first step is labelling all data with visualized evaluation, it is understandable since there is no previous research or work on this field. The second step is to ensure the previous visual labeled data is correct and convincing

4.1 Visual Identification

Although the color tinting scheme is introduced in the previous section, gray scale tinting scheme is referred in this research when generating a Heat Map Image. This is because the gray scale tinting scheme is more proper to the original raw data (MRT) which are single channel (2D) matrices. When a MRT is transformed to a colored (RGB-based) image, it becomes a 3 channels matrix. For example, an original MRT is 8×56×1, the generated image size will be 69×495×3, where the 3 is the Red, Green and Blue channel respectively. However, because MRT only has one channel, this augmentation does not provide more information, and on contrary, the data variance would decrease due to the more color channels. The meaning of employing Heat Map scheme is to normalize the original MRT and to augment the data variance without information loss. The gray scale tinting scheme only generates a single channel image which every pixel is from 0 to 255. This approach is suitable to the original MRT of dimension 8×56×1 and can reflect data variance within a table accurately. In a gray scale scheme, the brighter the block, the more intense the

communication between the pair (MC and PE). In another words, a white block represents intensive communication between the corresponding MC-PE pair and a black block means there is no traffic communication existing in the pair. The heat map images are evaluated independently to find out each distinguishable traffic pattern image category. Figure 3 to Figure 10 represent 8 different categories which have been classified manually, they are: 1) Column Pattern; 2) Row Pattern; 3) Row+Column Pattern (RC Pattern); 4) Group Pattern; 5) Diagonal Pattern; 6) Step Pattern; 7) Split Pattern; 8) Checkerboard Pattern. Each Image represents a Heat Map of the transformed MRT collected at kernel granularity. Not all images can be classified into the 8 categories, the rest images which cannot be visually classified is temporally put aside and will be evaluated their correct affiliation later. These manually identified traffic pattern categories would be strictly verified its correctness and generalization in the follow subsection.

To get more insight of these traffic pattern categories, especially to understand architecture level meaning of a pattern, every category of traffic pattern is analyzed in detailed about their meaning and formation reason.

Figure 3 Column Pattern observed in Gaussian benchmark

4.1.1 Column Pattern

A Column Pattern represented in Figure 3 only 1 PE sends requests and receives replies from all MCs during the execution of a kernel. This traffic pattern can be considered as a Few-to-Many-to-Few pattern and it leads to a distribution and receiving hotspot. The main reason that causes this pattern is due to the smaller number of thread blocks. Usually only one thread block is generated in the kernels which have Column Pattern, and the thread block is assigned to a PE and eventually only this specific PE has communication with MCs. A special case is more than one MCs do not have communication, this situation

happens when data addresses in the thread block are only distributed in some given memory chips.

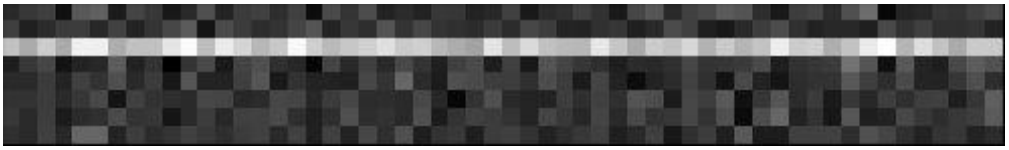

Figure 4 Row Pattern observed in BFS benchmark

4.1.2 Row Pattern

Figure 4 shows a Row Pattern, which means that only 1 MC sends replies to all PEs in a kernel. This is a typical M2F2M pattern because the very few (1 or 2) MCs have traffic communication with all PEs. The pattern is generated mainly because of imbalanced memory accessing or data address mapping scheme. For example, in BFS application, every kernel processes a layer of a graph. Because of the different number of nodes in each layer, the traffic intensity for each kernel is highly variance. And when the dataset of BFS is loaded into the main memory of a throughput processor, all the nodes of a layer is loaded into a given memory chip, which causes in a specific kernel, the corresponding memory chip that is stored the data of the processing layer, is accessed much frequently than other memory chips (around 4X in our experiments). This is a typical memory accessing imbalance which caused Row Pattern.

Figure 5 RC Pattern observed in Sorting Networks benchmark

4.1.3 RC Pattern

A RC Pattern is typically looking like a combination of Row Pattern and Column Pattern as Figure 5 represents. This pattern means that one memory chip is accessed more than any other chips and some PEs have intensive memory accessing to all MCs. According to the analysis in Row Pattern, the row pattern part in the RC Pattern is also because of imbalanced memory accessing (data address mapping). The column pattern behavior could be potentially caused by different reasons. One reason is that the thread block scheduling policy assigns more thread blocks to some special PEs due to its

own mechanism. For example, if a simple Round-Robin scheduling is employed in the experimental system, and some PEs finish the first-round assigned thread blocks earlier than other PEs, then the thread block scheduler would assign more thread blocks onto these PEs in a Round-Robin fashion, if some PEs always finish their assigned thread blocks faster than other PEs, the faster PEs would be assigned more thread blocks than other PEs, consequently these PEs would have more communication which results in lighter block in these columns. RC Pattern can be considered as a M2F2M pattern but with one or multiple hotspots in the interconnection network.

Figure 6 Group Pattern observed in Reduction benchmark

4.1.4 Group Pattern

As depicted in Figure 6, Group Pattern is a pattern with a continuous group of PEs have communication with all MCs. In general, Group Pattern is a more generalized case of Column Pattern. However, in manual labeling process, these 2 classes are not very similar, this is the reason why this 2 patterns are classified in 2 different categories, and this assumption will be further validated in the following subsection.

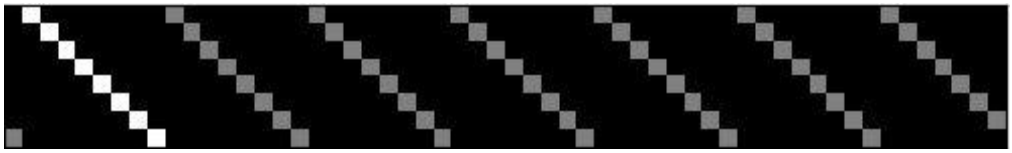

Figure 7 Diagonal Pattern observed in AlignedType bechmark

4.1.5 Diagonal Pattern

The Figure 7 describes a typical Diagonal Pattern. In this pattern, it is obvious that each MC has its own corresponding PE array. The index of start PE in each array is continuous and every array has the same interval. This distinctive pattern happens usually due to each thread block processes a part of the dataset and the whole dataset is evenly distributed in all memory chips. It is noteworthy that Figure 7 only gives a typical Diagonal Pattern, which means that in this category, some other similar cases may exists, but they could have different

interval and start PE indices. However, the interval in all arrays should be the same and the start PE indices should be an increasing sequence.

4.1.6 Stair Pattern

Figure 8 Stair Pattern observed in Histogram benchmark

In Figure 8, a general case of Stair Pattern observed in histogram benchmark. From the description of benchmark obtained in [1], demonstrates interop of rendering targets between Direct3D10 and CUDA. In the observed stair pattern, each SM interacts with 3 contiguous MCs i.e one possible MCs sequence is 0,1

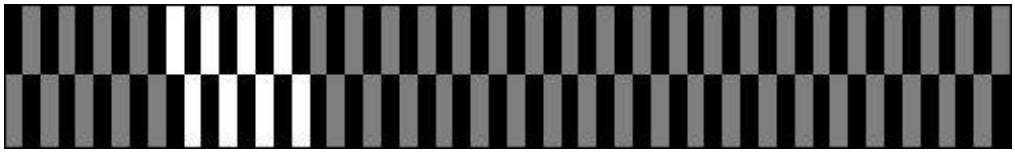

Figure 9 Split Pattern observed in AlignedType benchmark

and 2. The pattern is observed in some kernels of the histogram benchmark, since the nature of this benchmark is to deal with 3D values, the possible reason for 3 MCs serving a SM is to improve the processing speed of 3D vectors which are placed in 3 separate DRAMS connected to individual MC.

4.1.7 Split Pattern

In Figure 9 presents an Ideal case for Split pattern observed in Aligned Type Benchmark. In the considered architecture of 8 MCs and 56 SMs, the observed Split Pattern each SM either requests data packets from upper half (0,1,2,3) or lower half (4,5,6,7) MCs leaving the other half unused. The even distribution of traffic pattern intensity across upper or lower half of MCs is mainly due to the distribution of CTAs to SMs; based on the data collected with preliminary research its appears that each SM in Kernel-65 of AlignedType benchmark has 75 CTAs assigned to it. The AlignedType benchmark tests the performance of the GPGPU on aligned and mis-aligned data types, the preliminary results reveal that the data of interest for each thread block are accessed in DRAMs in sequential order. This explains the reason as to why a particular SM raises equal

number of memory requests to MCs of a given half. Interestingly the pattern observed reveals that, if a SM raises memory requests to the upper half of MCs, then its neighboring SM will keep the lower half of MCs busy.

Figure 10 Chessboard Pattern observed in CFD benchmark

4.1.8 Chessboard Pattern

The Chessboard Pattern is a special pattern which in our experiment only exists in CFD benchmark. Its characteristics are like the features of Diagonal Pattern, but in the manual classification we consider it as a new class because of its formation reason. Different from the reason of Diagonal Pattern, Chessboard Pattern forms because of the CFD has intra-PE communication through device memory. Therefore, the light block pairs indicate that for a specific PE, either odd or even index memory chips contain the needed data.

The previous 8 categories do not cover all images, some other images have no clear pattern, so they have been temporarily classified into a class called Random. The coverage rate (the total number of classified images over the total number of images in dataset) will be evaluated later in the Evaluation section. The current 8 types are not the final decision since they are not scientifically verified with a comprehensive and convincible approach. In the next subsection, such a method is proposed to validate the correctness and accuracy of manual labeling process

4.2 T-SNE Validation Schemes

This scheme adopts T-SNE algorithm on linear-divided row vectors which represent MRIs. These vectors are projected to a 2D or 3D coordinate system that is suitable for human interpretation [6]. A cluster dividing, and combination action is then performed after the projection step, which essentially is to find out all potential clusters from the projected dots and analyze whether each

cluster is a new category in addition to the previous 8 types or it belongs to any of an original type. The dividing and combination action would generate a new set of clusters which can be considered as correct labels (reasons will be discussed later).

In a CNN model, all Convolution layers, Max-Pooling/Average-Pooling layers and Dropout layers work together to compress a raw image to a series of matrices which are going to be fed into a simple neural network (NN) to get final classification. In the NN, there will be one input layer, some hidden layers and one output layer. The last hidden layer (right before the output layer) generates the important decision features of an input (MRI). For example, if a NN has 128 nodes in its last hidden layer, every input has a 1×128 row vector which exhibits the most important decision-making features of an image.

As discussed in Section II, applying T-SNE on raw images does not provide a reasonable result. This is mainly due to the T-SNE tries to linearly classify the raw image dataset which is not linearly divided. The original images has different pattern, rotation and translation which causes the dataset seems like randomly distributed in its own hyperspace. Nevertheless, the row vector that is generated by the last hidden layer in a NN is ensured linear-divided. The observation comes from the analysis of the Activation layer (the output layer of a NN). The Softmax function that is utilized in this layer is a linear classifier, which implies that the input of this layer should be linear-divided to activate the Softmax function. Based on the observation, the row vector which is the input of the Activation layer is then linear-divided.

According to the previous analysis and observation, now 2 basic conclusions are summarized:

1. The row vector that is generated by the last hidden layer is the key decision-making features of the final Activation layer, and at the same time, it is linear-divided.
2. To employ T-SNE to a dataset, the dataset should be linear-divided. Thus, the T-SNE can comprehensively reduce the data dimension to a visually identifiable coordinate system.

The first conclusion clearly indicates that, the original traffic pattern images can be replaced by the row vectors without any misleading. And the new dataset which contains all row vectors of MRIs, is ensured linear-divided. Then the new dataset can be fed into the T-SNE algorithm to perform data point projection from a hyperspace to a 2D or 3D space and get a visually identifiable coordinate system. The reason why 2D is chose in this paper instead of 3D is because there is not much differences between a 2D result and a 3D result.

There would be many clusters in the coordinate system (more than 8). After analyzing each of the clusters respectively, we find that the images that should be in the same category are distributed in different area in the coordinate system. This phenomenon happens usually because of the background noise affect. Therefore, the last step is to combine these distributed images together as pure mathematical theory (T-SNE) cannot correctly group similar MRIs together in one boundary clear area. Before combination processing, the data dots should be divided into as many clusters as possible. Then the detailed segmented clusters could be combined to form some new clusters due to their common features or similar patterns from human visualized method.

Practical application of this validation scheme is discussed in Chapter 5. A to validate our proposed original 8 patterns.

Chapter 5: Evaluation Methodology and Results

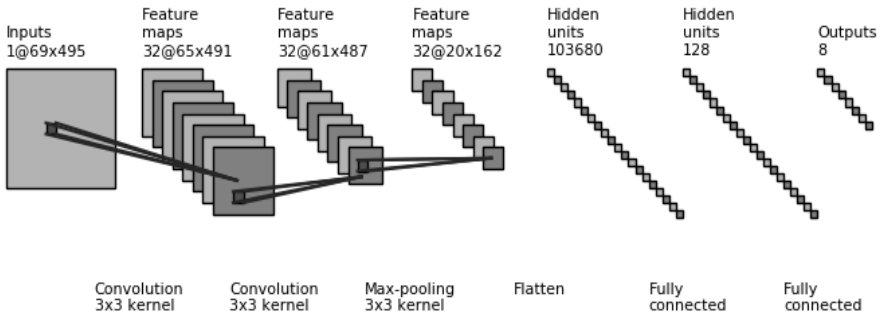

Figure 11 Convolutional Neural Network model used for feature extraction and classification

5.1 Training and Verification of CNN Model

CNN has proven to be a successful in dealing with multi-label classification of images with distinct features, i.e. well-defined edges and visually identifiable patterns [7] [10] [12]. In this paper the CNN model presented in Figure 11 is used for multi-label classification of the input traffic pattern image to the identified basic types discussed in Chapter 4. The model contains 8 layers including output layer. The model employed for traffic pattern classification uses two layers of convolution with each having 32 filters of size 5×5 and a stride length of 1, generating a feature map matrix of dimension 32×61×487. The Max-Pooling layer uses a 3×3 filter with a stride length of 1, this reduces the dimension of feature map matrix from 32×61×487 to 32×20×162 retaining enough feature information. The features learned can be visualized on a trained model by backtracking on to the input image in consideration. The mapping of features on to an original diagonal pattern image is shown in Figure 12, the top image is the original image and the bottom image is the feature extracted by the CNN model for classification. The dots in the feature map image are the diagonal edges of the original image. The down sampled feature map matrix is flattened to a vector of length 103680. This flattened vector is fed into a Neural network with an input layer, hidden layer and output layer, in Figure 11. The input and hidden layer is referred to a fully connected layer which generates the final classification.

Figure 12 Final features used for making decision on diagonal pattern

The CNN model in Figure 11 provides the probability weights of a given random input traffic pattern with respect to identified eight basic types. The weights obtained identifies the closeness of the input with the basic types described in Chapter 4. The closeness information is useful in understanding if the random input image has only one significant pattern or if it is a mix of more than one identified type. There could be cases where the traffic pattern is equiprobable and significant (i.e. consider the output probabilities obtained are [2/8,2/8,2/8,1/24,1/24,1/24,1/24,1/24,1/24] for a given input image, the equiprobable and significant out patterns are with probabilities 2/8,2/8 and 2/8).This means that the given random input image has properties of the 3 basic types which are equal in significance, and this property can be considered in designing NoC.

The training and test data set are obtained by transforming the traffic pattern images. The types obtained as discussed in Chapter 4 are very similar in nature (i.e. traffic pattern belonging to diagonal pattern almost look alike in major cases when seen them as an image). This is a serious problem during training as the data set obtained is not significantly huge to isolate the problem by dis-regarding the duplicate images. The model trained with this characteristic dataset over-fits very easily (i.e. model designed only to solve a particular

dataset, when new types are introduced behavior is unpredictable). The model is considered over-fit when the training loss is less than validation loss. The

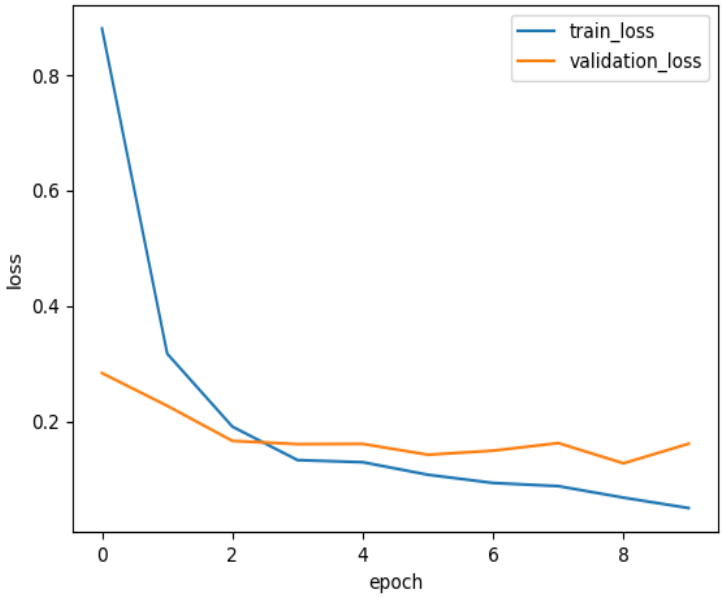

Figure 13 Training and Validation loss plot

Figure 13 is the training and validation loss plot for 10 epochs, it clear that the validation loss crosses training loss at epoch 2. The CNN model easily over-fits with minimal training epochs. KFold Cross Validation mechanism is proven to perform better with limited data set and are similar in nature [9]. Figure 14 shows the plot of training loss and validation loss of the trained CNN model with KFold Cross Validation with 10 folds. The final model uses the weights which produces best results of the 10 folds (iterations). The model achieves 96.6% training accuracy and 98.8% validation accuracy on training and validation data set. A 94.24% accuracy on test dataset. This is better than 93% precision obtained with SVM using *rbf* kernel. The main advantage of CNN model is that, it provides an opportunity to verify the visual classification which is not possible with with SVM.The results obtained are based on the classification obtained by visual classification.

To validate the visual classification and identify new types if missed any in visual classification, the method discussed in Chapter 4 to discover new cell types is used. In this method the final feature vector of the fully connected layer is fed into the unsupervised clustering algorithm T-SNE to visualize and

identify new clusters. In the paper under discussion we have used the 128-bit final decision vector for identifying new clusters. The Figure 15 is the plot of T-SNE applied on 128-bit vector feature vector used for final decision in classification. From the plot we can clearly identify seven clusters.

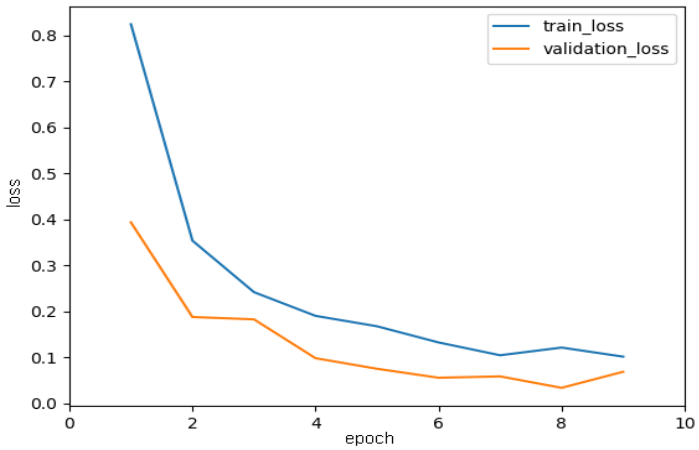

Figure 14 Training and Validation plot with K-Fold Cross Validation

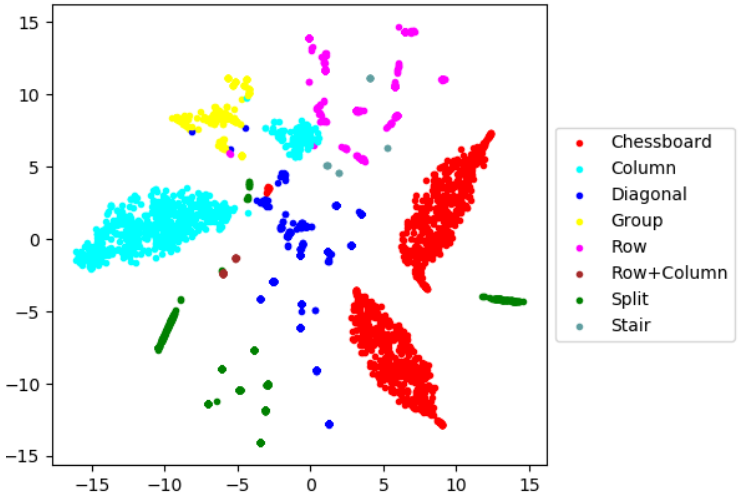

Figure 15 Clusters obtained by using T-SNE dimensional reductionality on 128-bit feature vector

Chapter 6: Conclusion

In this study, we have identified that the traffic patterns in NoC is not just a mere M2F2M but consists of many definitive patterns which are revealed on collecting the traffic data at various granularities. We have nailed down the granularity at which most of the pattern information is retrieved with minimal loss of information, collecting the traffic data at Kernel level. To have more visual intuition the collected traffic data is represented as matrices. Representing the collected data in matrices opens wide variety options for processing and classifying the collected data into groups based on some unique characteristics. The unique characteristics used could be eigenvalues, correlation coefficients, visual appearance based on transforming the matrices to image etc.., grouping based on visual characteristic is more suitable due to the properties exhibited by the collected data set.

The visually based grouping obtained by transforming collected traffic matrices can be used as an input to CNN models which have proven their ability in obtaining features and classifying the images. Usage of CNN gives the opportunity of validating the initial visual based classification/grouping used for training the CNN model, by connecting the final feature vector of the fully connected layer of CNN as an input to an unsupervised learning model such as T-SNE.

The basic traffic patterns obtained initially is based on the default configuration as show in (default configuration table). The basic traffic patterns identified could be impacted by changes in scheduling policy, network size and memory controller placement. Executing same set of benchmark application with changing simulator configuration yields interesting results, there is no change in the patterns observed it remains same as that of the default simulator configuration.

The CNN model used for classifying input traffic pattern to be identified into eight classifications yields an accuracy of 96.6% on training data set, 98.8% on validation data set and an 94.2% accuracy on the test data set. The results

obtained are promising and encourages usage of deep learning techniques in identifying the similarity in studying traffic pattern behavior.

Bibliography

[1] *CUDA 9.2 Now Available,* 2018.

[2] V. Adhinarayanan and W.-c. Feng, "An automated framework for characterizing and subsetting GPGPU workloads," in *Performance Analysis of Systems and Software (ISPASS), 2016 IEEE International Symposium on*, 2016.

[3] S. Che, J. W. Sheaffer, M. Boyer, L. G. Szafaryn, L. Wang and K. Skadron, "A characterization of the Rodinia benchmark suite with comparison to contemporary CMP workloads," in *Workload Characterization (IISWC), 2010 IEEE International Symposium on*, 2010.

[4] S. Che, B. M. Beckmann, S. K. Reinhardt and K. Skadron, "Pannotia: Understanding irregular GPGPU graph applications," in *Workload Characterization (IISWC), 2013 IEEE International Symposium on*, 2013.

[5] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, 2009.

[6] M. B. Eisen, P. T. Spellman, P. O. Brown and D. Botstein, "Cluster analysis and display of genome-wide expression patterns," *Proceedings of the National Academy of Sciences,* vol. 95, pp. 14863-14868, 1998.

[7] P. Eulenberg, N. Köhler, T. Blasi, A. Filby, A. E. Carpenter, P. Rees, F. J. Theis and F. A. Wolf, "Reconstructing cell cycle and disease progression using deep learning," *Nature communications,* vol. 8, p. 463, 2017.

[8] R. Girshick, J. Donahue, T. Darrell and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014.

[9] N. Goswami, R. Shankar, M. Joshi and T. Li, "Exploring GPGPU workloads: Characterization methodology, analysis and microarchitecture evaluation implications," in *Workload Characterization (IISWC), 2010 IEEE International Symposium on*, 2010.

[10] A. Kerr, G. Diamos and S. Yalamanchili, "A characterization and analysis of ptx kernels," in *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, 2009.

[11] R. Kohavi and others, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Ijcai*, 1995.

[12] L. v. d. Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of machine learning research,* vol. 9, pp. 2579-2605, 2008.

[13] A. Sharif Razavian, H. Azizpour, J. Sullivan and S. Carlsson, "CNN features off-the-shelf: an astounding baseline for recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2014.

[14] J. A. Stratton, C. Rodrigues, I.-J. Sung, N. Obeid, L.-W. Chang, N. Anssari, G. D. Liu and W.-m. W. Hwu, "Parboil: A revised benchmark suite for scientific and commercial throughput computing," *Center for Reliable and High-Performance Computing,* vol. 127, 2012.

[15] Y. Wei, W. Xia, M. Lin, J. Huang, B. Ni, J. Dong, Y. Zhao and S. Yan, "Hcp: A flexible cnn framework for multi-label image classification," *IEEE transactions on pattern analysis and machine intelligence,* vol. 38, pp. 1901-1907, 2016.

[16] S. Wold, K. Esbensen and P. Geladi, "Principal component analysis," *Chemometrics and intelligent laboratory systems,* vol. 2, pp. 37-52, 1987.