Characterizing On-Chip Traffic Patterns in General-Purpose GPUs: A Deep Learning Approach

Yunfan Li, Drew D. Penney, Abhishek Ramamurthy, Lizhong Chen
School of Electrical Engineering and Computer Science
Oregon State University, Corvallis, USA
{liyunf, penneyd, abhisher, chenliz}@oregonstate.edu

***Abstract*—Architectural optimizations in general-purpose graphics processing units (GPGPUs) often exploit workload characteristics to reduce power and latency while improving performance. This paper finds, however, that prevailing assumptions about GPGPU traffic pattern characterization are inaccurate. These assumptions must therefore be re-evaluated, and more appropriate new patterns must be identified. This paper proposes a methodology to classify GPGPU traffic patterns, combining a convolutional neural network (CNN) for feature extraction and a t-distributed stochastic neighbor embedding (t-SNE) algorithm to determine traffic pattern clusters. A traffic pattern dataset is generated from common GPGPU benchmarks, transformed using heat mapping, and iteratively refined to ensure appropriate and highly accurate labels. The proposed classification model achieves 98.8% validation accuracy and 94.24% test accuracy. Furthermore, traffic in 96.6% of examined kernels can be classified into the eight identified traffic pattern categories.**

I. Introduction

Architectural optimizations are generally evaluated using representative workloads that test specific architectural capabilities. Improved understanding of these workloads helps better adapt optimizations to real-world behavior. Data exchange patterns in these workloads significantly impacts performance. Specifically, communication between processing elements (PEs) and memory controllers (MCs) in general-purpose graphics processing units (GPGPUs) can affect many components, including network-on-chip (NoC) design, data prefetching policy, *etc*. This communication is increasingly impacted by dramatic increases in GPGPU PE count. It is therefore imperative to accurately characterize GPGPU traffic patterns to enable efficient hardware designs.

Communication patterns in GPGPUs differ from those in traditional multi-core CPUs since GPGPU PEs request/receive data packet independently. Limited work has considered memory access or communication patterns in NoCs [1], [2], [3], among which none exploited GPGPU traffic patterns; researchers primarily consider many-to-few-to-many (M2F2M) pattern, regardless of the application. As a result, these works attempt to optimize GPGPU NoCs by widening the "few" points [4] or by allocating NoC sub-networks for different packet types [5]. In our testing, we find that, surprisingly, only few applications exhibit this M2F2M pattern during execution on GPGPUs, especially at the kernel-level. In contrast, many patterns involve communication between specific PE-MC pairs, rather than between many PEs and a few MCs.

Our testing motivates traffic pattern analysis in other applications, which may exhibit further pattern diversity. Detailed evaluation on over 30 benchmarks verifies that the many-to-few-to-many pattern is far less common than previously believed. This crucial observation motivates reassessment of commonly accepted GPGPU traffic patterns. We therefore provide insight into these newly considered patterns, which may enable more effective future optimizations in GPGPUs.

Existing methods for dataset classification, however, cannot be directly applied to GPGPU traffic patterns without either significant loss in classification accuracy or extremely inefficient manual evaluation. Data collection granularity must be carefully chosen since it affects information quality/detail and data redundancy, both of which impact characterization accuracy. Data normalization must also be considered since traffic pattern data can vary in value from ten to millions. Mathematical techniques such as correlation, eigendecomposition, and singular value decomposition (SVD) have limited capabilities with respect to rotations and translations, which commonly occur in the traffic pattern dataset. Conventional supervised learning algorithms require accurate ground truth labels, but these do not exist since no prior research has characterized GPGPU traffic patterns. Unsupervised learning models can help extract dataset features without ground truth labels, but naive implementation is not possible since raw traffic pattern data is not linearly divisible.

Exploiting these newly considered traffic patterns necessitates novel methods to identify similarities and differences in traffic behaviors. We propose a deep learning approach to differentiate traffic patterns in GPGPU applications. This work also presents a method to generate an appropriate training set based on strictly verified traffic pattern data. To that end, we evaluate several choices for data collection granularity. This raw data, however, cannot be directly used to train our deep learning model. We thus utilize a heat map algorithm to augment and normalize the dataset, allowing convenient visual classification for dataset labeling. Finally, labels are verified using the t-SNE algorithm. This approach linearly divides, then projects data onto a low-dimensional coordinate system, allowing visual inspection of data clusters and potentially new traffic patterns. The entire verification process iterates to refine the label for each data point. The resulting deep learning model achieves 98.8% validation accuracy and 94.24% test accuracy. Furthermore, traffic in 96.6% of examined kernels can be classified into the eight identified traffic pattern categories.

This paper is organized as follows: Section II provides necessary background and highlights challenges in traffic pattern identification; Section III details our data collection and transformation techniques; Section IV elaborates our proposed scheme for traffic pattern classification; Section V then presents identified traffic patterns and model training results; Section VI discusses related work; finally, section VII concludes.

TABLE I: Cumulative communication traffic of AlignedTypes

	PE1	....	PE30	PE31	....	PE56
MC1	10683	...	276189	10683	...	10683
MC2	10681	...	276187	10681	...	10681
MC3	10680	...	10680	276186	...	10680
MC4	10680	...	10680	276186	...	10680
MC5	10680	...	1109310	10680	...	285336
MC6	10680	...	10680	1109310	...	285336
MC7	285336	...	10680	10680	...	1078800
MC8	1353456	...	10680	10680	...	10680

TABLE II: Kernel-1 communication traffic of AlignedTypes

	PE1	PE2	PE3	PE4	PE5	...
MC1	3	61033	3	0	3	...
MC2	0	0	61030	0	0	...
MC3	0	0	0	61030	0	...
MC4	0	0	0	0	61030	...
MC5	0	0	0	0	0	..
MC6	0	0	0	0	0	...
MC7	0	0	0	0	0	...
MC8	30510	0	0	0	0	...

TABLE III: Kernel-57 communication traffic of AlignedTypes

	PE1	...	PE10	PE11	PE12	PE13	...
MC1	0	...	18306	0	0	0	...
MC2	0	...	18306	0	0	0	...
MC3	0	...	0	18306	0	0	...
MC4	0	...	0	18306	0	0	...
MC5	0	...	0	0	18306	0	..
MC6	0	...	0	0	18306	0	...
MC7	9156	...	0	0	0	18306	...
MC8	9156	...	0	0	0	18306	...

II. Background and Motivation

A. Limited existing characterization

Unlike traditional many-core processors, PEs in GPGPUs request data and perform computation independently, which implies less mutual data exchange among PEs. Additionally, the number of PEs typically dominates the number of MCs, resulting in overall data exchange patterns between many PEs and a few MCs. Specifically, many PEs generate read or write request packets that are sent, through the NoC to be processed by a few MCs. Following processing, reply packets are sent back to the many PEs. This behavior is referred to as a many-to-few-to-many (M2F2M) traffic pattern. Studies toward GPGPU architecture optimization usually consider M2F2M as the primary traffic pattern, so apply novel techniques to widen the "few" points in NoCs [4] or try to exploit specific traffic features [5] to improve NoC throughput.

Prevailing understanding of the M2F2M traffic pattern is explained by results for cumulative data transfer across entire benchmarks, as shown in Table I. Here, PEs are shown on the horizontal axis, and MCs are shown on the vertical axis. Cell values therefore indicate the number of packets exchanged between a specific PE-MC pair. Data collection at a finer granularity (*e.g.*, kernel-level), however, reveals dramatically different behavior. For example, Table II presents the traffic pattern distribution for the first kernel of the AlignedTypes benchmark [6]. At this granularity, each PE primarily communicates with a single MC. This observation suggests that M2F2M may not be the primary traffic pattern in GPGPUs. We verify this hypothesis by conducting similar tests across a wide range of benchmarks from Rodinia [7], Parboil [8] and NVIDIA SDK [6]. Detailed analysis is, however, required to identify the actual traffic patterns and associated communication behavior of GPGPU workloads.

There is no prior research on characterizing traffic patterns in GPGPUs besides coarse-grained understanding of M2F2M. Some previous works have characterized PEs or memory related metrics such as memory locality, latency hiding ability, branch prediction *etc.*, but not traffic patterns or machine/deep learning models [9], [10], [3], [11]. Several works employ machine learning algorithms to optimize GPGPU components such as memory controllers, branch predictors, and data prefetchers [11], [2], [**?**], [12], [13], [14], [15]. The proposed machine learning algorithms in these works are, however, not suitable for characterizing traffic pattern data due to idiosyncrasies in traffic pattern data, which will be discussed later. Work by Giles *et al.* [14], for example, proposed a neural network (NN) approach to optimize NoC routing policies. Their model is very simple and therefore inadequate for complex matrix data classification tasks like traffic pattern characterization. Hashemi *et al.* [**?**] proposed a recurrent neural network (RNN) to predict and prefetch cache lines from memory. Their model is not compatible with traffic pattern data since individual data points do not have any spatial or temporal context. Moreover, since there is no related previous research, there is no labeled traffic pattern dataset, thus no possibility for simple supervised learning methods. Some other works focus on characterizing architecture independent metrics such as PTXs, CTAs, dynamic instruction count *etc.* by employing unsupervised learning methods such as principle component analysis (PCA) or hierarchical clustering to identify similarities across different benchmarks [16], [17], [18]. These traditional unsupervised cluster methods, however, have significant limitations and, as explained in section II-B, cannot correctly identify key traffic pattern features. Therefore, to identify traffic patterns in GPUs, a novel approach is needed that can comprehensively and accurately recognize potential patterns that exist during application execution.

B. Challenges

Tested benchmarks exhibit high variance in traffic patterns depending upon the data collection granularity. Specifically, the cumulative traffic pattern across an entire benchmark is often not the same as the pattern at either specific cycle ranges or during individual kernel execution. As will be discussed in Section III, data collected at the kernel level retains important features for specific benchmarks and is applicable across entire benchmark suites. Data representation must also be considered. In this paper, traffic pattern results are represented as matrices for flexibility in data processing options, which are needed to extract similarities and classify results accurately. Transforming matrices to a suitable form also provides options for visual analysis and could be used as an input for deep learning models.

Following considerations for granularity and representation, we can group similar matrices based on observed characteristics. Unfortunately, common mathematical techniques, such as correlation, eigendecomposition, or singular value decomposition (SVD), cannot be used to classify traffic pattern matrices. In the first method, correlation coefficients measure similarities of matrices. Traffic pattern matrices, however, are not strictly correlated as matrices with similar traffic patterns

may differ in the position of occurrence (i.e., MC-PE pairs are completely different). For example, the traffic patterns in Table II (Kernel-1) and III (Kernel-57) are highly similar, as they both present a down gradient, but with different degrees and PEs. Correlation analysis would conclude that Kernel-1 and Kernel-57 are not similar as correlation is position dependent. Eigendecomposition is also problematic since traffic patterns matrices are not square and also since results would change with linear and horizontal shifts (yet represent the same traffic pattern); these shifted matrices would yield different characteristic roots, which is an undesired behavior. SVD has a similar mathematical theory as eigendecomposition, thus also not suitable for traffic pattern characterization. In summary, more advanced methods are required to extract and analyze traffic pattern information.

Existing unsupervised machine learning models including PCA and t-distributed stochastic neighbor embedding (t-SNE) are potential alternatives to cluster similar matrices without initial human classification. Naive implementation, however, does not yield good results, as illustrated in Figures 1 and 2. PCA results in Figure 1 do not help identify principle traffic pattern types since most points lie on a single straight line. Furthermore, this line contains several classes that can be visually differentiated by inspecting the original data, indicating that PCA is not suitable for this task. T-SNE projects high-dimensional data onto 2D or 3D space to explicitly cluster correlated data points [19]. Results for t-SNE, illustrated in Figure 2, exhibit superior performance to PCA, with t-SNE successfully identifying a few clusters with similar traffic patterns (e.g., the two red circled clusters). Several other clusters (e.g., the yellow, brown, and green clusters), however, contain data points from dissimilar traffic patterns, indicating that a naive t-SNE implementation still cannot provide high accuracy traffic pattern clustering.

Supervised learning methods, such as support vector machines (SVMs), require data to be transformed into a format suitable for input and also require labeling prior to training. Using SVMs, data is classified based on curve fitting in a high dimensional space. This approach does not, however, provide feature extraction capabilities, which could be useful for label validation and help identify missing traffic pattern classes or incorrectly labeled data. Conventional supervised learning methods are therefore, in general, not ideal for this task.

Convolutional neural networks (CNNs) capture correlation in dataset features using filters to extract key input features. A simple CNN may include several convolutional layers, pooling layers, and a fully connected layer (standard neural network layer). Each convolutional layer extracts some representative feature in the input, such as edges in an image. Specifically, features are extracted by multiplying each filter cell with the corresponding cell in the input matrix and summing the results. Filters are translated across the input to extract all relevant features, where each filter can learn to extract a specific feature. These extracted features are then sent into a pooling layer for downsampling to reduce matrix dimensionality and avoid model overfitting. The final flattened output matrix serves as input to the fully-connected layer, which produces a final multi-labeled classification. Training a CNN still requires a labeled dataset which, prior to this work, does not exist. We therefore require a method to generate ground truth labels for traffic pattern data to ensure high model accuracy following training.

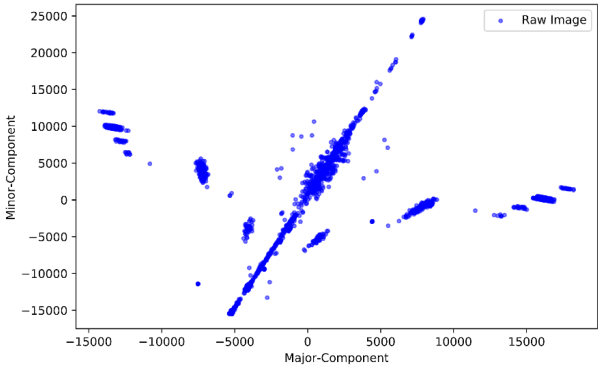

Fig. 1: PCA results for raw dataset.

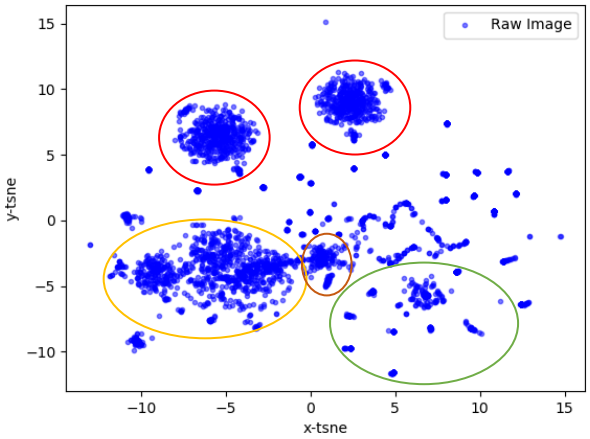

Fig. 2: T-SNE results for raw dataset.

III. Transforming Traffic Data for Deep Learning

A. Data collection granularity

Traffic pattern matrices are collected from the cycle-accurate GPUGPUSim simulator (details in Section V), using an ideal NoC configuration to eliminate any interconnection network impact. This configuration is necessary since various topologies, routing policies, allocation schemes, *etc*, can influence the communication pattern. An ideal NoC allows collected data to instead focus on the impact of the traffic pattern itself. This data is collected, decomposed, and analyzed after benchmark execution (*accumulation pattern*), at defined ranges of execution cycles (*cycle-range pattern*) and at the kernel-granularity (*kernel pattern*). Results show that kernel patterns are most appropriate in capturing traffic flow patterns during runtime while retaining detailed information, as explained below.

Data collection granularity requires similar consideration to address critical impact on data redundancy and accuracy; fine-grained data collection can exhibit dramatically different runtime traffic behavior than coarse-grain data collection on the same benchmark. We find that fine-grained data collection provides the most accurate representation so is used in this paper. There are still several options. The most fine-grained option is cycle-accurate, which generates a matrix at every cycle. This method, however, is impractical since benchmarks may run for tens of millions of cycles, leading to significant dataset redundancy in which most matrices are sparse and meaningless. Cycle-range patterns are an alternative, with two typical policies to define the range of cycles for data collection. The first, and most accurate, method involves manually tracking

TABLE IV: Example flattened dataset

	PE1-MC1	PE2-MC1	...	PE1-MC2	PE2-MC2	PE3-MC2	...	PE1-MC8	PE2-MC8	...
A_Kernel-1	15	2	...	15	2	15	...	15	2	...
A_Kernel-2	100	120	...	200	220	240	...	800	820	...
A_Kernel-3	25	3	...	25	3	25	...	25	3	...
B_Kernel-1	10000	500000	...	200	200	20	...	100000	40	...
B_Kernel-2	1000	2000	...	500	200	15	...	100	500	...

all benchmark cycles, then defining continuous cycle ranges with similar traffic patterns for data collection. This approach is impractical since it would require manual action at the cycle level. A second approach instead uses pre-defined cycles based on a fixed number or percentage of total cycles. Consequently, this approach is much faster but not necessarily accurate. Consider that the optimal cycle count granularity may differ across benchmarks. Using intervals of 10,000 cycles, CFD [7] has 2268 data points while Gaussian has only three, so information will likely be lost. This issue could potentially be addressed by defining a fixed percentage, but this may still lead to information loss. Again, considering CFD, if data collection occurs at 1% cycle count intervals, 22,636,606 cycles are included, which is likely too many to represent fine-grain traffic patterns. Alternatively, we can consider a fixed cycle count or percentage of cycles for each benchmark. This approach is still too time consuming since it requires manual evaluation for each benchmark. Overall, cycle-range patterns are not feasible since they require either significant manual evaluation (to avoid dataset redundancy) or compromise information quality.

Kernel patterns offer a solution for both issues; traffic pattern matrices generated after kernel execution retain detailed traffic information with low dataset redundancy. Specifically, individual kernel executions usually perform a similar task every time the kernel is executed (e.g., loading data or performing a specific computation). Therefore, within a single kernel, traffic flow is nearly identical so little information is lost. Kernel execution is also long enough to ensure sufficient detail (no completely sparse matrices). As an additional benefit, architecture optimization based on kernel pattern data can improve execution for all kernels with similar behavior. Lastly, there are more than 10,000 total kernels across tested benchmarks, providing an acceptable training set size.

Raw data matrices generated during benchmark execution are, however, unsuitable for deep learning models since matrix are too small and not normalized. We propose a combined data augmentation and normalization scheme to both enlarge matrix dimensions (while retaining feature accuracy) and eliminate data redundancy to speedup model convergence.

B. Heat Map Data Transforming

Effective deep learning application requires consideration for data format (both representation and actual values). Briefly, a compatible data format is not necessarily an ideal format. The raw traffic pattern dataset has two major issues: obscure features and not normalized matrices. Data is gathered on an architecture with 56 PEs and 8 MCs, so matrix dimensions are 8×56. These dimensions are much smaller than the typical input size for CNN models (224×224 for VGG-16 [20]). Consequently, traffic pattern features may be obscured or even undetectable, reducing model accuracy. This problem can be addressed by a data augmentation scheme. Simply enlarging raw matrices is not viable because it radically alters the meaning of individual matrix cells. In the raw format, cell values indicate the number of packets exchanged between a specific PE-MC pair. In a scaled format, these non-zero values would be expanded to other cells, which is not appropriate.

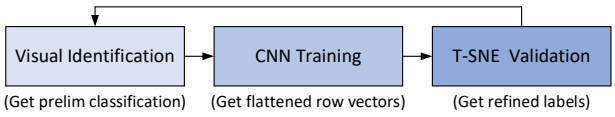

Fig. 3: Proposed methodology to identify traffic patterns

Another major challenge is data normalization. Normalization reduces the adverse impact from diverse data scales and focuses the model on significant traffic patterns. A conventional normalization scheme would flatten each matrix into a row vector, combine the resulting row vectors together into a new matrix, and finally normalize each column individually. Normalizing across row vectors (the original matrices) implicitly assumes that all features (elements in the matrix) use the same scale for data. Traffic pattern data in this paper, however, is only meaningful in the context of the original matrix. This technique would therefore distort the original meaning of the data. Table IV demonstrates this issue with a simple example dataset. Here, rows in the table correspond to the flattened matrices from individual kernel executions. Before normalization, it is easy to observe that Kernel-1 and Kernel-3, both from example benchmark A, have the same pattern in which odd index PEs have heavier traffic than even index PEs. Kernel-2 exhibits a separate pattern in which traffic load increases from the first PE to the last and also increases with the MC index. Conventional normalization techniques (such as min-max [21] or z-score [22]) would dramatically shrink several columns (such as PE1-MC1 and PE2-MC1) to accommodate the large packet counts in B_Kernel-1. Consequently, other values in those columns (such as those for A_Kernel-1) would become very small, reducing the likelihood that patterns can be accurately recognized. Various columns would also scale differently, again altering the meaning of the original values.

We propose using grayscale heat maps to both augment and normalize the traffic pattern dataset. These heat maps (such as Figure 6 shown later) code elements in a matrix based on their relative values [23], generating images in which each block (group of identical pixels) corresponds to a cell in the original matrix. Using 8×8 blocks for every original one "pixel" cell transforms the original 8×56 matrices into 64×448 images. These larger images can improve recognition accuracy in deep learning models and simplify the visual identification and validation process introduced in the next section. Heat maps also naturally eliminate normalization problems because all pixel values range from 0-255. Block intensity is proportional to the original cell values, with the highest value cell(s) white and the lowest value cell(s) black. Each image is generated with respect to a single matrix, thus limiting any negative impact due to different scales for packet count in various kernels.

Transforming raw data matrices into heat map images represents half of the dataset generation task. Labels must still be generated for each image in order to train the CNN model using backpropagation.

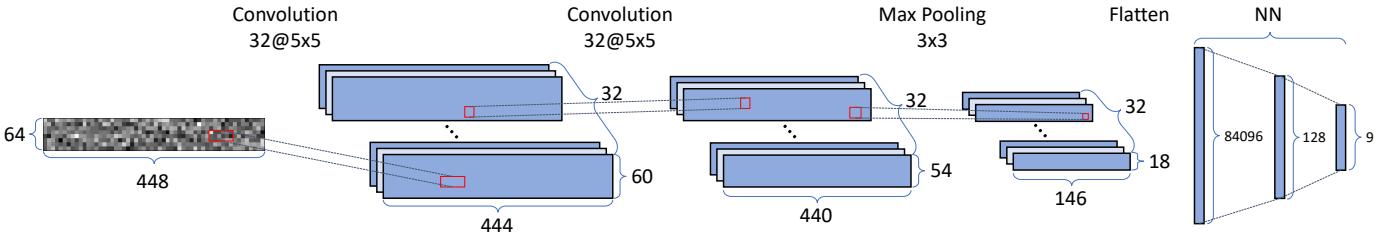

Fig. 4: Convolutional Neural Network model used for feature extraction and classification

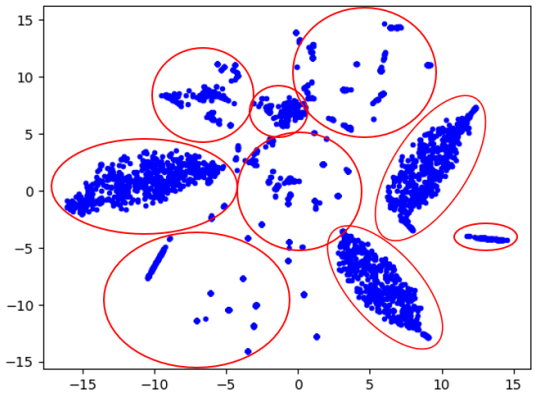

Fig. 5: Clusters obtained on CNN feature map to verify labels used for classification

IV. Identifying Traffic Patterns

A. Overview

The proposed scheme for traffic pattern labeling is depicted in Figure 3. As previously mentioned, effective neural network training requires accurate labels for data points since the training process relies on the loss (difference) between predicted labels and ground truth labels. The labeling process generates accurate labels in a three-step procedure of visual identification and label validation. In the first step, labels are generated through visual identification (i.e., looking at heat maps for dominant patterns), which is a common step in many well-known image datasets such as CIFAR10 [24] and MNIST [25]. This step is required since there is no prior research on these diverse GPGPU traffic patterns.

The resulting pre-labeled dataset is then used in label validation. In this step, the CNN is trained on the pre-labeled dataset. Following initial training, we obtain a flattened vector representation for each data point by capturing the output from "Flatten" in our CNN model (Figure 4; explained shortly). Crucially, this representation is linearly divisible due to non-linear neural network activations (explained in section IV-C). In the third step, this flattened representation is combined with labels from the pre-labeled dataset and fed into a t-SNE algorithm. This algorithm projects the high dimensional traffic data onto a 2-D coordinate system, allowing further verification for existing labels. This projection further enables visual inspection of traffic pattern distributions to determine if the clusters that have been identified so far are appropriate. Poorly defined clusters or mixed distributions may suggest the presence of incorrect labels, missing categories, *etc.*, all of which can be corrected in a subsequent iteration(s), as indicated in the back arrow in Figure 3.

B. Visual Identification

Grayscale heat maps are preferred over RGB heat maps since grayscale avoids unnecessarily augmenting the raw 2D matrices and additional noise that can impact classification accuracy. This representation is used for initial visual identification to generate the pre-labeled dataset. During the labeling process, some patterns are easily identified and labeled, such as the *column pattern* presented in Figure 6 and the *row pattern* presented in Figure 7. Other patterns, however, are difficult to differentiate, leading to potentially inaccurate initial labels. This labeling alone is therefore insufficient and must be strictly validated and adjusted to ensure the ground truth for every matrix is reasonable. Further use of visual verification after t-SNE is described in section IV-D.

C. CNN Details

Subsequent label validation using t-SNE requires a linearly divisible dataset, which can be obtained from a CNN model. A typical CNN model contains several convolution layers, pooling layers, and a standard neural network classifier. As shown in Figure 4, the input (e.g., a heat map image), passes through several convolution layers to compress and extract key features such as edges. Pooling layers then downsample extracted feature images to both shrink the final decision vector size and focus the neural network on relative feature positions rather than specific locations [26]. Following several convolution-pooling combinations, extracted feature images can be transformed into flattened vectors that are relatively small, but retain significant feature information. The final stage is a classical neural network containing an input layer, some hidden layers, and an output layer. A softmax activation function is chosen for the output layer to easily characterize the relative likelihood of various traffic pattern classes. More details of the CNN architecture used in this work are presented in Figure 4. This model contains eight total layers. The first two convolution layers each uses 32 filters of size 5×5 and a stride length of 1, generating feature map matrices of dimension $32 \times 60 \times 444$ and $32 \times 54 \times 440$, respectively. The subsequent max-pooling layer uses a 3×3 filter with a stride length of 1 to downsample feature map matrices from $32 \times 54 \times 440$ to $32 \times 18 \times 146$. Extracted images are then flattened into 84096×1 vectors and fed into the neural network (NN) layer for classification.

Note that output vectors from the "Flatten" layer are guaranteed to be linearly divisible, as the non-linear activation functions (i.e., ReLu [27], tanh, *etc.*) in previous layers provide non-linear transformation from the original hyperspace, thus "twisting" and projecting the data into a linearly divisible space.

D. T-SNE Validation Scheme

Validation using t-SNE projects the CNN-generated decision vector for each data point onto a 2-D coordinate system that is

Fig. 6: Column Pattern (e.g., Gaussian, Kernel-1)

Fig. 7: Row Pattern (e.g., BFS, Kernel-10)

Fig. 8: RC Pattern (e.g., SortingNetworks, Kernel-6)

Fig. 9: Group Pattern (e.g., Reduction, Kernel-8)

Fig. 10: Diagonal Pattern (e.g., AlignedTypes, Kernel-1)

Fig. 11: Stair Pattern (e.g., Histogram, Kernel-48)

Fig. 12: Split Pattern (e.g., ConvolutionTexture, Kernel-2)

Fig. 13: Checkerboard Pattern (e.g., CFD, Kernel-45)

suitable for human interpretation [28]. Ideally, this projection will separate traffic patterns into clearly defined (and separate) clusters; any new clusters therefore imply new traffic pattern categories. Dataset labels can then be adjusted to better match any newly identified clusters. Following label revision, this adjusted dataset can be used to train a more accurate CNN model that will then extract more appropriate features from traffic pattern heat maps. This iterative process ensures that labels will eventually be correct and the CNN model will be highly accurate.

T-SNE application in Section II-B did not provide acceptable results because t-SNE cannot linearly classify the raw dataset. Specifically, considering every cell in the original traffic matrix as an axis in a hyperspace, both rotations and translations could be interpreted as new patterns since they will be randomly distributed in the hyperspace. It is therefore critical that the dataset is made linearly divisible so that t-SNE can project the dataset onto a human interpretable coordinate system. As previously mentioned, vectors generated by the CNN "Flatten" layer are linearly divisible and each raw data point corresponds to only one such vector, indicating that the original heat map image dataset can be replaced by a new dataset of these vectors. The clusters obtained from running t-SNE on this linearly divisible dataset are presented in Figure 5. In this representation, there are nine distinct clusters (red circles). These clusters, however, still contain points from different categories (i.e., different labels). This phenomenon likely results from background noise and inherent visual similarities between traffic patterns. These points are specifically targeted for further visual inspection to ensure that replaced labels are reasonable. The overall three-step procedure iterates several times to continuously improve dataset label accuracy.

V. Results and analysis

A. Traffic Patterns

Figures 6 to 13 represent eight different categories that are identified and verified by following the aforementioned label creation process, including: 1) Column pattern; 2) Row pattern; 3) Row+Column pattern (RC pattern); 4) Group pattern; 5) Diagonal pattern; 6) Stair pattern; 7) Split pattern; 8) Checkerboard pattern. We analyze these traffic categories in detail to identify underlying causes and better understand their behavior at the architectural level.

1) Column pattern: A column pattern, presented in Figure 6, comprises traffic from one PE to all MCs, leading to a PE send/receive hotspot. This pattern results from kernel execution involving few thread blocks; all thread blocks are then assigned to one PE that communicates with all MCs. A special case occurs when more than one MC is not involved, which may occur when data addresses are not distributed across all memory chips.

2) Row pattern: Figure 7 illustrate a row pattern in which only 1 MC communicates with all PEs. This pattern is generated by imbalanced memory access or data address mapping schemes. For example, in BFS, every kernel processes one graph layer. Consequently, memory accesses are concentrated to very few MCs that contain the data for nodes in that layer.

3) RC pattern: A typical RC pattern, shown in Figure 8 combines the previous row and columns patterns. In this pattern, one MC is heavily accessed by all PEs, but some PEs still communicate heavily with all MCs. The row pattern, as previously mentioned, could be established due to imbalanced memory accessing. The column pattern here could be caused by different reasons. A main reason is that the thread block scheduling policy assigns more thread blocks to a few specific PEs based on a special mechanism. For example, if some PEs consistently finish their assigned thread block faster than other PEs, the faster PEs would tend to be assigned more thread blocks and therefore communicate more heavily with all MCs. This RC pattern can also be considered as a special case of M2F2M pattern, but with one or more hotspots.

4) Group pattern: Figure 9 depicts a group pattern in which a continuous group of PEs communicate with all MCs. The group pattern can also be viewed as an extended case of the column pattern. Labels for the two patterns, however, are kept separate since their visual appearance can differ substantially and are therefore easily separated during manual labeling. This distinction is further validated in the following subsection.

5) Diagonal pattern: In a diagonal pattern, each MC has a corresponding PE array with which it communicates, as illustrated in Figure 10. The index of the first PE in these arrays is continuous (in Figure 10, the start indices are 0, 1, 2, 3, 4, 5, 6, and 7), and every array has the same interval (in Figure 10, the interval is 8). This distinctive pattern results when each thread block processes a part of the dataset and the whole dataset is evenly distributed in all memory chips. Many

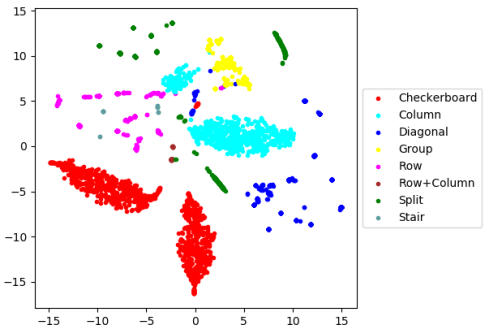

Fig. 14: Colored clusters for different traffic patterns

diagonal patterns are possible by varying the PE interval or start PE indices. Regardless, the array intervals should be identical and the start PE indices should be an increasing sequence.

6) Stair pattern: The stair pattern, shown in Figure 11, is present in the histogram benchmark for example. In this case, data is distributed relatively evenly across memory chips rather than concentrated in a single chip. Additionally, thread blocks processing the same data are assigned to a continuous and adjacent set of PEs. Variations on this pattern can translate horizontally or vertically due to differing data storage circumstances.

7) Split pattern: Figure 12 presents a case for the split pattern. This pattern arises when individual PE communication is restricted to either MCs in the upper half (index 0-3) or the lower half (index 4-7), but not both. This situation results from a combination of thread block scheduling policies and data storage positions in which thread blocks processing half the data are always scheduled to certain index (either odd or even) PEs.

8) Checkerboard pattern: The checkerboard pattern is a special pattern that, in testing, only appears in CFD. This pattern has similar features to the diagonal pattern (with an interval of 1), but is classified separately due to its unique formation circumstances. Unlike the diagonal pattern, checkerboard forms due to intra-PE communication through memory chips. Light block pairs therefore indicate that, for a specific PE, either odd or even index memory chips contain the data needed from other PEs.

Figure 14 presents the final t-SNE distribution with each color representing a specific traffic pattern. Several important conclusions can be drawn from this figure. First, we can observe that there are six primary categories that are all relatively centralized. The remaining two patterns, including stair and RC, are less conspicuous because these patterns are far less common in our dataset. The stair pattern, in particular, is distributed similarly to the row pattern, confirming the similarities recognized in visual inspection. Specifically, the stair pattern gradient may be too shallow to recognize and, consequently, may be recognized as a straight line.

B. Model accuracy and coverage

The kernel-level communication dataset is collected using GPGPUSim v3.2.2 with a recent GPGPU architecture and key parameters listed in Table V. This configuration uses an ideal network-on-chip (NoC) in which packets are transferred directly to their destination in one cycle, ensuring that traffic pattern data

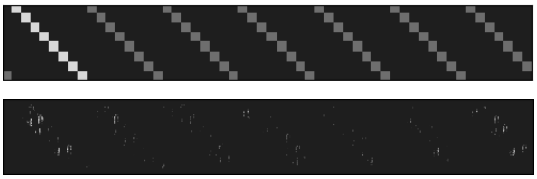

Fig. 15: Final features (bottom image) used for making decision on the Diagonal pattern

TABLE V: Key Parameters in Simulation.

Field	Value
Processing Elements	56, 1126MHz
Shared Memory / Element	48KB
L1 Cache Size / Element	16KB
L2 Cache Size / Element	2MB
# of Last Level Cache Banks	8
Memory Chips	8, FR-FCFS
Network-on-Chips	Ideal

is not influenced by the NoC architecture. Testing also confirms that the eight major traffic pattern categories are unaffected by the number of PEs or MCs, implying that conclusions are applicable to diverse GPGPU architectures. Recall that the pattern identification flow (Figure 3) is architecture agnostic, indicating that future GPGPU architectures will be compatible with our methodology as long as it is strictly followed. Finally, the specific prediction model (CNN in our work) also does not impact these categories, because classifications are obtained and validated separately; validation requires just the linearly divisible flattened vectors. This classification process can therefore be adapted to any other prediction model that can generate these vectors.

Significant features learned by the CNN model can be visualized by tracing output activations back to specific input elements. As an example, the feature map for a diagonal pattern is shown in Figure 15. The top image is the original input and the bottom image represents the features extracted by the CNN model. As expected, dots in the feature map trace the diagonal edges of the original image.

All heat map images are randomly assigned into training or test dataset with a 5:1 ratio. As previously mentioned, several categories are very similar, which might be problematic if a naively trained model is used, because the training dataset is not large enough to successfully differentiate some similar images. Consequently, the model will overfit easily, meaning that it can only classify a particular dataset and will behave unpredictably when new data is introduced. Figure 16 presents the training and validation loss across 10 epochs for such a problematic situation. Validation loss crosses training loss at epoch 2, indicating that the model begins to overfit at epoch 2. In contrast, K-fold cross validation is an alternative method that has been proven to perform better with limited data that is similar in nature [29]. Figure 17 plots training and validation loss using the k-fold cross validation with 10 folds. Our final model uses the best weights obtained using this approach and achieves 98.8% validation accuracy and 94.24% test set accuracy. Model coverage is another significant criteria that expresses how well kernels are classified into the identified traffic pattern categories. In our tests, the traffic patterns in 11,655 out of 12,064 kernels are correctly classified, giving a 96.6% coverage rate. The remaining 409 kernels are considered

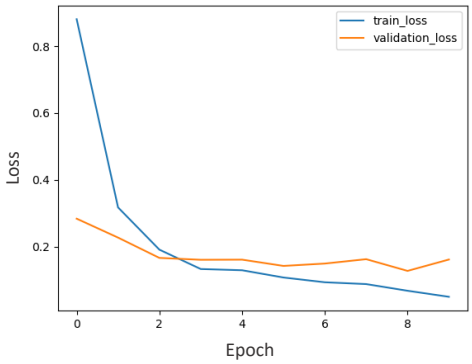

Fig. 16: Training and Validation loss plot

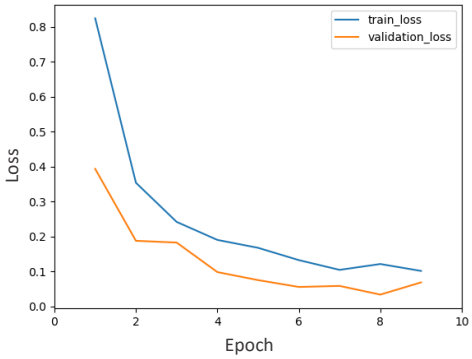

Fig. 17: Training and Validation loss plot with KFold Cross-Validation

to be typical M2F2M patterns, but most can still be recognized by our trained model. Figure 18 demonstrates Kernel-36 in SPMV [8]. According to our CNN model, this traffic is 47.83% similar to a diagonal pattern and 40.18% similar to a column pattern. Specifically, the left part of Figure 18 has diagonal edge (left two red circles) and the third circle clearly emphasizes an edge that appears in column patterns. However, those mixed pattern kernels account for only 3.4% of all the kernels.

C. Discussion

Several GPGPU architectural optimizations can benefit from improved traffic pattern characterization. Power-gating is a representative example. Long wake-up delays in traditional approaches constrain NoC throughput [30], [31]. Router wake-up could, however, be precisely scheduled given sufficient information about future traffic. Routers could also be powered down at more appropriate times based on this information, thus simplifying decision hardware and further reducing power and area. Another application is data prefetching. Prefetching policies could be simplified by collecting fewer cycles of traffic information while making higher accuracy predictions that improve performance. Again, prefetch hardware complexity (and therefore area/power) can be reduced due to improved knowledge of traffic pattern. NoC research in GPGPUs, as mentioned, has primarily considered optimizations for many-to-few-to-many traffic patterns. Given that only around 3% of kernels exhibit this pattern, there is likely significant potential for improvement by additional optimizations for the eight major patterns identified in this paper. Other improvements may also be exploited by integrating traffic pattern characterization in warp scheduling, memory load-balancing, application-specific optimizations, and many other techniques.

Fig. 18: SPMV Kernel-36 traffic pattern image

VI. Related Work

Prior work has applied machine learning (ML) to architectural optimization in both traditional CPUs and throughput processors. These works are all entirely orthogonal to our proposed work on traffic pattern characterization. Hashemi *et al.* proposed a RNN-based data prefetcher to achieve higher fetch precision [1]. Ipek *et al.* instead applied reinforcement learning to memory controllers, enabling improvements in MC efficiency. Related work has also explored machine learning application to interconnection networks. For example, Sakr *et al.* employed three different ML models to learn memory access patterns and dynamically reconfigure the interconnection network architecture [2]. Giles *et al.* used a simple neural network to generate control bits for optical multistage NoCs [14]. Branch prediction is another popular area for research in ML application since the task is easily represented in terms of actions and goals. For example, Jimenez *et al.* proposed a perceptron-based model for dynamic branch prediction to improve prediction accuracy [11]. Work by Blanton *et al.* has even used online learning in self-evolving systems that continuously optimize system performance. Work focusing on throughput processors, in particular, could benefit from improved understanding of traffic patterns in GPGPUs.

Several works have also applied ML to workload characterization. Wu *et al.* proposed a ML model to predict GPGPU power and performance across many kernels, enabling significant execution time savings compared to cycle-accurate simulation [15]. Other works have used ML to characterize architecture independent metrics such as parallel thread execution (PTX) and dynamic instruction count for various workload, thereby allowing existing workload to be categorized based on architecture independent behaviors [9], [3], [10], [17], [16], [32], [33]. These works commonly use PCA or hierarchical clustering. Recent deep learning models, however, have been proven to provide higher classification accuracy on complex image datasets. Our work therefore utilizes CNNs to perform traffic pattern classification.

VII. Conclusion

Accurate traffic pattern characterization in GPGPUs enables effective optimization in network-on-chip (NoC) design, data prefetching policy, *etc*. We show, however, that prevailing assumptions about GPGPU traffic patterns are inaccurate at fine-grain execution scales, and therefore require reassessment. The proposed scheme for traffic pattern classification addresses this problem using a three step procedure. First, traffic pattern data is transformed into heat maps. Next, a convolutional neural network is trained to generate flattened feature vectors. Finally, t-SNE projects high dimensional traffic pattern feature vectors into 2-D clusters, allowing visual verification. The resulting model achieves 94.24% accuracy in traffic pattern classification on the test dataset. Furthermore, these patterns are highly representative of real-world benchmarks, with over 96% of kernels exhibiting these patterns.

ACKNOWLEDGMENTS

We sincerely thank the reviewers for their helpful comments and suggestions. This research was supported, in part, by the National Science Foundation grants #1566637, #1619456, #1619472 and #1750047.

REFERENCES

[1] M. Hashemi, K. Swersky, J. A. Smith, G. Ayers, H. Litz, J. Chang, C. Kozyrakis, and P. Ranganathan, "Learning memory access patterns," *arXiv preprint arXiv:1803.02329*, 2018.

[2] M. Sakr, S. P. Levitan, D. M. Chiarulli, B. G. Horne, and C. L. Giles, "Predicting multiprocessor memory access patterns with learning models," in *ICML*, pp. 305–312, 1997.

[3] T. Allen and R. Ge, "Characterizing power and performance of gpu memory access," in *Proceedings of the 4th International Workshop on Energy Efficient Supercomputing*, pp. 46–53, IEEE Press, 2016.

[4] A. Bakhoda, J. Kim, and T. M. Aamodt, "Throughput-effective on-chip networks for manycore accelerators," in *Proceedings of the 43rd annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 421–432, 2010.

[5] H. Jang, J. Kim, P. Gratz, K. H. Yum, and E. J. Kim, "Bandwidth-efficient on-chip interconnect designs for gpgpus," in *Proceedings of the 52nd Annual Design Automation Conference*, p. 9, ACM, 2015.

[6] NVIDIA, "Cuda code samples." https://developer.nvidia.com/cuda-code-samples, Oct 2018.

[7] 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 *IEEE International Symposium on Workload Characterization (IISWC)*, pp. 44–54, 2009.

[8] 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.

[9] S.-Y. Lee and C.-J. Wu, "Characterizing the latency hiding ability of gpus," in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 145–146, 2014.

[10] J. Weinberg, M. O. McCracken, E. Strohmaier, and A. Snavely, "Quantifying locality in the memory access patterns of hpc applications," in *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, p. 50, IEEE Computer Society, 2005.

[11] D. A. Jiménez and C. Lin, "Dynamic branch prediction with perceptrons," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 197–206, 2001.

[12] E. Ipek, O. Mutlu, J. F. Martínez, and R. Caruana, "Self-optimizing memory controllers: A reinforcement learning approach," in *ACM SIGARCH Computer Architecture News*, vol. 36, pp. 39–50, IEEE Computer Society, 2008.

[13] R. D. Blanton, X. Li, K. Mai, D. Marculescu, R. Marculescu, J. Paramesh, J. Schneider, and D. E. Thomas, "Statistical learning in chip (slic)," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 664–669, 2015.

[14] C. L. Giles and M. W. Goudreau, "Routing in optical multistage interconnection networks: A neural network solution," *Journal of lightwave technology*, vol. 13, no. 6, pp. 1111–1115, 1995.

[15] G. Wu, J. L. Greathouse, A. Lyashevsky, N. Jayasena, and D. Chiou, "Gpgpu performance and power estimation using machine learning," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 564–576, 2015.

[16] N. Goswami, R. Shankar, M. Joshi, and T. Li, "Exploring gpgpu workloads: Characterization methodology, analysis and microarchitecture evaluation implications," in *IEEE International Symposium on Workload Characterization (IISWC)*, pp. 1–10, 2010.

[17] V. Adhinarayanan and W.-c. Feng, "An automated framework for characterizing and subsetting gpgpu workloads," in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 307–317, 2016.

[18] S. Che, B. M. Beckmann, S. K. Reinhardt, and K. Skadron, "Pannotia: Understanding irregular gpgpu graph applications," in *IEEE International Symposium on Workload Characterization (IISWC)*, pp. 185–195, 2013.

[19] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[20] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[21] D. Tax and R. Duin, "Feature scaling in support vector data descriptions," *Learning from Imbalanced Datasets*, pp. 25–30, 2000.

[22] D. Zill, W. S. Wright, and M. R. Cullen, *Advanced engineering mathematics*. Jones & Bartlett Learning, 2011.

[23] 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, no. 25, pp. 14863–14868, 1998.

[24] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," tech. rep., Citeseer, 2009.

[25] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[26] D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," in *International conference on artificial neural networks*, pp. 92–101, Springer, 2010.

[27] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[28] 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, no. 1, p. 463, 2017.

[29] R. Kohavi *et al.*, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *IJCAI*, vol. 14, pp. 1137–1145, Montreal, Canada, 1995.

[30] L. Chen and T. M. Pinkston, "Nord: Node-router decoupling for effective power-gating of on-chip routers," in *Proceedings of the 45th annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 270–281, 2012.

[31] L. Chen, L. Zhao, R. Wang, and T. M. Pinkston, "Mp3: Minimizing performance penalty for power-gating of clos network-on-chip," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 296–307, 2014.

[32] 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 *IEEE International Symposium on Workload Characterization (IISWC)*, pp. 1–11, 2010.

[33] A. Kerr, G. Diamos, and S. Yalamanchili, "A characterization and analysis of ptx kernels," in *IEEE International Symposium on Workload Characterization (IISWC)*, pp. 3–12, 2009.