HARDWARE ACCELERATION FOR SWIN TRANSFORMERS AT THE EDGE

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF
MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

By
Yunus Esergün
May 2024

Hardware Acceleration for Swin Transformers at the Edge
By Yunus Esergün
May 2024

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Uğur GÜDÜKBAY(Advisor)

Özcan ÖZTÜRK(Co-Advisor)

İbrahim KÖRPEOĞLU

Süleyman TOSUN

Approved for the Graduate School of Engineering and Science:

Orhan Arıkan
Director of the Graduate School

ABSTRACT

HARDWARE ACCELERATION FOR SWIN TRANSFORMERS AT THE EDGE

Yunus Esergün
M.S. in Computer Engineering
Advisor: Uğur GÜDÜKBAY
Co-Advisor: Özcan ÖZTÜRK
May 2024

While deep learning models have greatly enhanced visual processing abilities, their implementation in edge environments with limited resources can be challenging due to their high energy consumption and computational requirements. Swin Transformer is a prominent mechanism in computer vision that differs from traditional convolutional approaches. It adopts a hierarchical approach to interpreting images. A common strategy that improves the efficiency of deep learning algorithms during inference is clustering. Locality-Sensitive Hashing (LSH) is a mechanism that implements clustering and leverages the inherent redundancy within Transformers to identify and exploit computational similarities. This thesis introduces a hardware accelerator for Swin Transformer implementation with LSH in edge computing settings. The main goal is to reduce energy consumption while improving performance with custom hardware components. Specifically, our custom hardware accelerator design utilizes LSH clustering in Swin Transformers to decrease the amount of computation required. We tested our accelerator with two different state-of-the-art datasets, namely, Imagenet-1K and CIFAR-100. Our results demonstrate that the hardware accelerator enhances the processing speed of the Swin Transformer when compared to GPU-based implementations. More specifically, our accelerator improves performance by 1.35x while reducing the power consumption to 5-6 Watts instead of 19 Watts in the baseline GPU setting. We observe these improvements with a negligible decrease in model accuracy of less than 1%, confirming the effectiveness of our hardware accelerator design in edge computing environments with limited resources.

Keywords: Swin Transformer, LSH, Hardware Accelerator, Inference, GPU, FPGA, Edge, Power.

ÖZET

UÇTA SWIN TABANLI DÖNÜŞTÜRÜCÜLER İÇİN DONANIM HIZLANDIRMASI

Yunus Esergün
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Danışmanı: Uğur GÜDÜKBAY
İkinci Tez Danışmanı: Özcan ÖZTÜRK
Mayıs 2024

Derin öğrenme modelleri görsel işleme yeteneklerini büyük ölçüde artırmışken, sınırlı kaynaklara sahip uç ortamlarında bu modellerin uygulanması, yüksek enerji tüketimi ve hesaplama gereksinimleri nedeniyle zor olabilir. Swin Tabanlı Dönüştürücü, geleneksel evrişimsel yaklaşımlardan farklı olarak, görüntüleri yorumlamak için hiyerarşik bir yaklaşım benimseyen bilgisayarla görmede öne çıkan bir mekanizmadır. Derin öğrenme algoritmalarının çıkarım sürecindeki verimliliğini artıran yaygın bir strateji kümeleme yöntemidir. Yerel Hassas Özetleme (YHÖ), dönüştürücüler içindeki doğuştan gelen fazlalıkları belirleyip hesaplama benzerliklerinden yararlanan bir mekanizmadır. Bu tez, uç hesaplama ortamlarında YHÖ ile Swin Tabanlı Dönüştürücü uygulaması için bir donanım hızlandırıcı tanıtmaktadır. Ana hedef, özel donanım bileşenleriyle performansı artırırken enerji tüketimini azaltmaktır. Özellikle, özel donanım hızlandırıcı tasarımımız, gerekli hesaplama miktarını azaltmak için Swin Tabanlı Dönüştürücülerde YHÖ kümelemeyi kullanmaktadır. Hızlandırıcımızı Imagenet-1K ve CIFAR-100 olmak üzere iki farklı veri seti ile test ettik. Sonuçlarımız, hızlandırıcının Swin Tabanlı Dönüştürücünün işleme hızını GPU tabanlı uygulamalara kıyasla artırdığını göstermektedir. Özellikle, hızlandırıcımız performansı 1.35 kat artırırken, güç tüketimini temel GPU ayarında 19 Watt yerine 5-6 Watt'a düşürmektedir. Bu iyileştirmeleri, model doğruluğunda %1'den düşük bir azalma ile gözlemliyoruz, bu da tasarımımızın sınırlı kaynaklara sahip uç hesaplama ortamlarında etkinliğini doğrulamaktadır.

Anahtar sözcükler: Swin Tabanlı Dönüştürücü, LSH, Donanım Hızlandırıcı, Çıkarım, GPU, FPGA, Uç, Güç.

Acknowledgement

I want to thank my thesis advisor, Prof. Uğur Güdükbay, for his support throughout my master's work.

I want to thank my thesis co-advisor, Prof. Özcan Öztürk, for his support throughout my master's work.

Finally, I want to thank my mother, who has always supported me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Convolutional Neural Networks (CNNs) and Transformer architectures like Swin Transformer have become attractive solutions for computer vision applications. These models are well-known for their ability to tackle intricate visual tasks, primarily due to their improved capacity to model long-range inter-dependencies and maintain detailed visual information. The Swin Transformer model deviated from traditional CNNs to provide a more adaptable and flexible approach to processing visual data [1]. Nevertheless, these advanced models face significant obstacles despite their impressive capabilities in environments with limited resources, such as edge devices. The primary challenges lie in their substantial energy consumption and high computational requirements, making them less viable for applications where efficiency and power consumption are much more critical [2], [3], [4].

Edge computing primarily aims to process data near its source rather than in remote data centers. It offers benefits such as reduced latency, utilization of the available bandwidth, and improved privacy [5], [6]. However, deploying complex deep learning models in such environments requires adopting suitable hardware to meet their computational requirements while considering power and space limitations [7], [8]. Traditionally, GPU platforms like the NVIDIA Jetson devices have been considered for edge computing due to their compact size and relatively strong processing capabilities [9], [10]. However, these platforms often

struggle with energy efficiency and computational costs when handling advanced deep learning models like the Swin Transformer [5], [11], [12].

This thesis introduces a fresh look at edge computing by proposing custom designs instead of traditional platforms like GPU devices. FPGA-based custom hardware accelerators, renowned for their adaptable architecture, provide extensive parallelization capabilities and effectively utilize on-chip memory. These characteristics make custom hardware designs suitable for enhancing the speed of deep learning models, especially in situations that require computational efficiency and minimal energy usage [13], [14].

The main objective of this approach is to implement reuse within the context of Swin Transformers through custom hardware implementation. More specifically, we implement a Locality Sensitive Hashing (LSH) based clustering to enhance the computational efficiency of deep learning models. While LSH clustering has been used in the literature, this thesis explores its implementation for a custom accelerator for Swin Transformers. The research includes a thorough comparative analysis of the performance of the LSH-enhanced Swin Transformer model on a custom accelerator and a GPU. The empirical findings are such that the LSH-enhanced model on the GPU demonstrates a 1.29x speedup compared to the baseline. In contrast, the custom accelerator implementation exhibits 1.35x speedup, along with a reduction in energy consumption to about 6W, as opposed to about 19W GPU power consumption.

The rest of the thesis is organized as follows. Section 2 discusses the relevant literature, establishing the context and groundwork for this research. It emphasizes prior research on LSH clustering within deep learning models and highlights developments in accelerators and GPU optimizations for deep learning applications. Section 3 provides a comprehensive overview of the fundamental principles underlying Vision Transformers, specifically emphasizing Swin Transformers and LSH clustering. Section 4 motivates conducting this study, highlighting the computational obstacles associated with implementing the Swin Transformer model in limited-resource settings. It emphasizes optimizing these models to achieve efficiency without sacrificing performance. Section 5 outlines the structure and

design of the accelerator proposed in this research, describing its constituent elements, functionalities, and the reasoning behind its configuration to optimize the Swin Transformer model through LSH clustering. Section 6 gives the experimental setup, encompassing hardware configurations, datasets, and default parameters. This section presents the methodology utilized to assess the efficacy, energy efficiency, and accuracy of the Swin Transformer accelerator compared to its GPU equivalent. Section 7 concludes the thesis by providing an overview of the research findings and highlights the potential of utilizing accelerator technology to improve the implementation of deep learning models in edge computing.

Chapter 2

Related Work

The integration of LSH clustering into deep learning models, specifically within the context of the Vision Transformer, is an advancement in improving computational effectiveness [15]. LSH clustering organizes comparable characteristics in image patches, leading to simplified calculations and enhanced efficiency. Referred to as "deep reuse," this strategy benefits from decreasing computational burdens and expediting inference durations [16]. The notion of deep reuse has been explored in deep learning, specifically for CNN models [17]. Previous studies have provided evidence for the efficacy of this strategy in diverse scenarios [18]. One notable example is the utilization of deep reuse in CNN inference, where exploiting the resemblances among neuron vectors in activation maps has proven to yield substantial acceleration [17, 19]. This approach has effectively streamlined CNN inferences by identifying and reutilizing computations that exhibit similarity across distinct inputs or within a single input, resulting in significant time savings [19].

A significant advancement in this field involves the creation of adaptive methods for enhancing the training of CNNs through deep reuse techniques [19, 20]. These methods effectively identify and bypass repetitive computations during the training phase, thus exhibiting the potential to decrease the computational burden while still maintaining the effectiveness of the learning process. The ability of

these techniques to adapt to various data sets and network structures has played a pivotal role in their success [19]. Additional studies have delved into using deep reuse techniques, specifically in the context of Winograd convolution algorithms, to optimize computational efficiency [21]. Winograd algorithms are renowned for diminishing the arithmetic complexity of convolutions in neural networks, and the integration of deep reuse techniques holds promise for achieving even greater computational efficiency [21].

Substantial progress has been achieved in enhancing the hardware-accelerated general matrix-matrix multiplication (GEMM) for CNNs on FPGAs [22]. The optimization of FPGA-based accelerators for GEMM has effectively enhanced the efficiency of convolutional layers in diverse CNN architectures. These advancements have substantiated the potential of accelerators in significantly improving the performance of efficient and intricate CNNs, surpassing the capabilities of current leading implementations [22]. Furthermore, enhancing data locality in depthwise separable convolutions for CNN inference accelerators has demonstrated significant decreases in DRAM energy usage [23]. By maximizing data reuse and minimizing data movement for intermediate activations, this methodology has enhanced the speed and energy efficiency of accessing external memory [23]. The SIMCNN technique has successfully showcased substantial computational efficiency in accelerating the training of CNNs by utilizing computational similarity in hardware. This technique involves implementing a cache system called SIMCACHE, which stores LSH signatures of recent input vectors and the corresponding computed outcomes. Consequently, this allows for retrieving and reusing previously calculated results in situations where newly introduced input vectors align with existing signatures within the SIMCACHE [24].

Researchers in [25] focus on the constraints of single-board FPGA configurations in achieving optimal energy efficiency and introduce a deeply pipelined multi-FPGA architecture to address these challenges. The study proposes a new dynamic programming algorithm for efficiently mapping CNN layers across multiple FPGA boards, resulting in significant energy improvements [25]. Another research paper contributes to the advancement of FPGA-based acceleration for CNNs, focusing on the issues related to the flexibility and efficiency of hardware

accelerators [26]. This paper proposes an instruction-driven architecture that enables the use of the Winograd algorithm and cross-layer scheduling [26]. By optimizing the order of loop unrolling in cross-layer fusion, the authors effectively minimize intermediate data transfers [26].

A different research direction investigates the limited use of computational capabilities in modern FPGAs by implementing a systolic array architecture designed for high clock frequencies and efficient use of resources [27]. The authors introduce a mathematical model for predicting performance and resource usage, along with a tool that automates the exploration of different design options [27]. This tool helps convert C code into a systolic array-based implementation of CNNs to simplify the development process and improve the flexibility of FPGA accelerators for CNN inference [27].

Our methodology differs from previous approaches, as we present a specialized hardware accelerator architecture designed to improve the performance of Swin Transformers in edge computing environments. While existing solutions primarily concentrate on optimizing GPU-based systems or FPGA designs for CNN models, our research focuses on incorporating LSH clustering with custom hardware design technology to enhance the efficiency of Swin Transformers. This fusion decreases computational cost and energy consumption and maintains sufficient accuracy on edge devices.

Chapter 3

Background

3.1 Vision Transformers

The introduction of Vision Transformers (ViTs) has significantly changed computer vision. This architecture diverges from the conventional convolutional approaches traditionally used for image processing tasks, as depicted in Figure 3.1. Drawing inspiration from the effectiveness of Transformers in natural language processing, ViTs employ the self-attention mechanism to process visual data, providing a fresh approach to capturing and leveraging the abundant information contained within images [28], [29].

CNNs conventionally employ filters to analyze images, capturing local patterns through weight sharing and spatial hierarchies. However, this methodology assumes a rigid, grid-like aggregation of local features into higher-level representations. On the other hand, ViTs take a different approach by treating images as sequences of patches, similar to words in a sentence. This allows the model to learn contextual relationships between various parts of an image in a flexible and adaptable manner [28], [30].

The study that initially introduced ViTs to the field profoundly impacted the

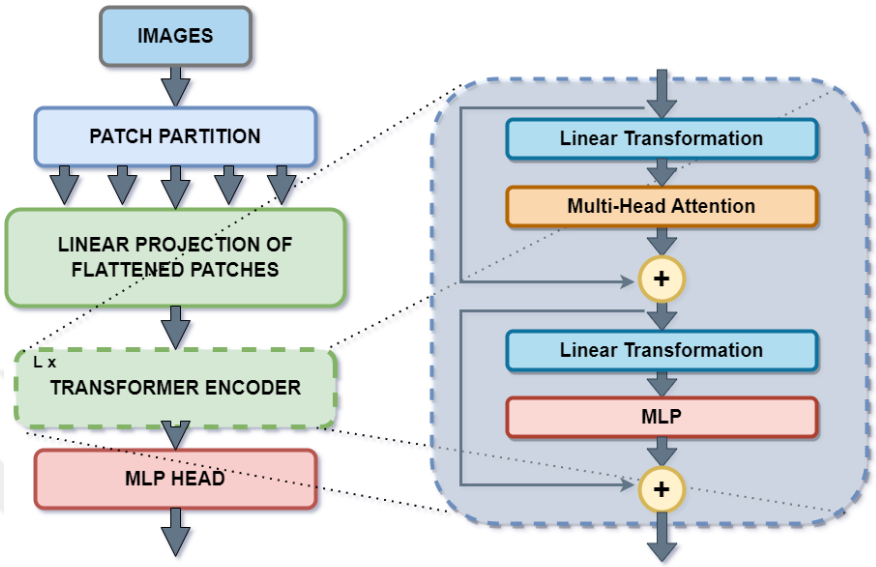

Figure 3.1: Vision Transformer architecture stages.

transformer architecture, typically employed for sequential data, by adapting it to the two-dimensional nature of images. This approach divides images into patches, each flattened and linearly embedded. Position embeddings are then added to preserve spatial information. These embedded patches undergo multiple layers of multi-head self-attention, enabling the model to assign varying importance to different parts of the image based on the content of other patches. This departure from the local receptive fields found in CNNs results in a broader global receptive field for ViTs, potentially allowing them to capture more intricate and comprehensive patterns within visual inputs [28], [31].

The primary source of effectiveness in the ViT model is its self-attention mechanism, which calculates the response at a specific position by considering the contributions from all positions in the input sequence [32]. In visual information processing, this implies that each patch within an image can attend to every other patch, irrespective of their spatial proximity. This capacity to incorporate long-range interactions among patches enables ViTs to acquire spatial hierarchies and object-centric representations that are not inherently constrained by the network's structure, as observed in CNNs [28].

3.2 Swin Transformer

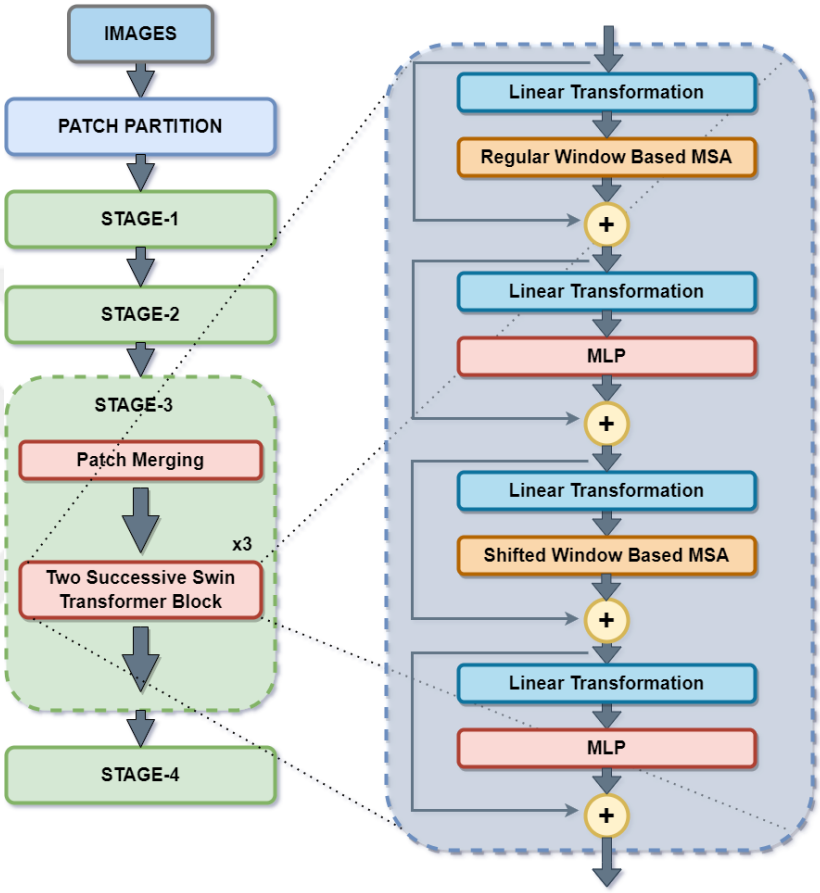

Figure 3.2: Swin Transformer base architecture properties.

The Swin Transformer is one of the new architectures in the field of computer vision that differs from traditional convolutional approaches. It adopts a hierarchical approach to interpreting images, as depicted in Figure 3.2. The process begins with dividing the image into fixed-sized patches, which are then processed through multiple stages of the network. Each stage includes patches merging and combining adjacent patches to create larger, more informative patches. Swin Transformer blocks, which involve linear transformations, self-attention mechanisms, and multilayer perceptrons (MLPs), are employed. This hierarchical structure enables the model to capture fine-grained details at lower levels and more abstract representations at higher levels, essential for tackling complex visual recognition tasks [1], [33].

The third stage of the Swin Transformer stands out due to its frequent utilization of patch merging and transformation blocks three times more than the other stages. Although highly effective, the Swin Transformer's computationally intensive nature poses difficulties when deploying it on devices with limited resources. Overcoming these challenges is crucial for progressing the utilization of advanced deep learning models in edge computing, emphasizing efficiency and minimal power consumption [1].

3.3 LSH Clustering

LSH clustering is an approach that aims to improve the efficiency of deep learning algorithms during inference. It leverages the inherent redundancy within the layers of CNNs and Transformers to identify and exploit computational similarities across different inferences or within the same inference process. This technique involves clustering similar neuron vectors and reusing their computed outcomes for subsequent operations. Implementing LSH clustering does not require modifications to the underlying deep learning architectures, allowing seamless integration into existing models. This adaptability facilitates reductions in inference times without sacrificing much accuracy [34], [35].

The effectiveness of LSH clustering is derived from its ability to dynamically recycle on-the-fly computations, resulting in a decrease in the redundant calculations usually needed by Transformers. By identifying and grouping similarities in neuron vectors using different techniques, LSH clustering minimizes the computational load, thereby accelerating the inference procedure [36], [37].

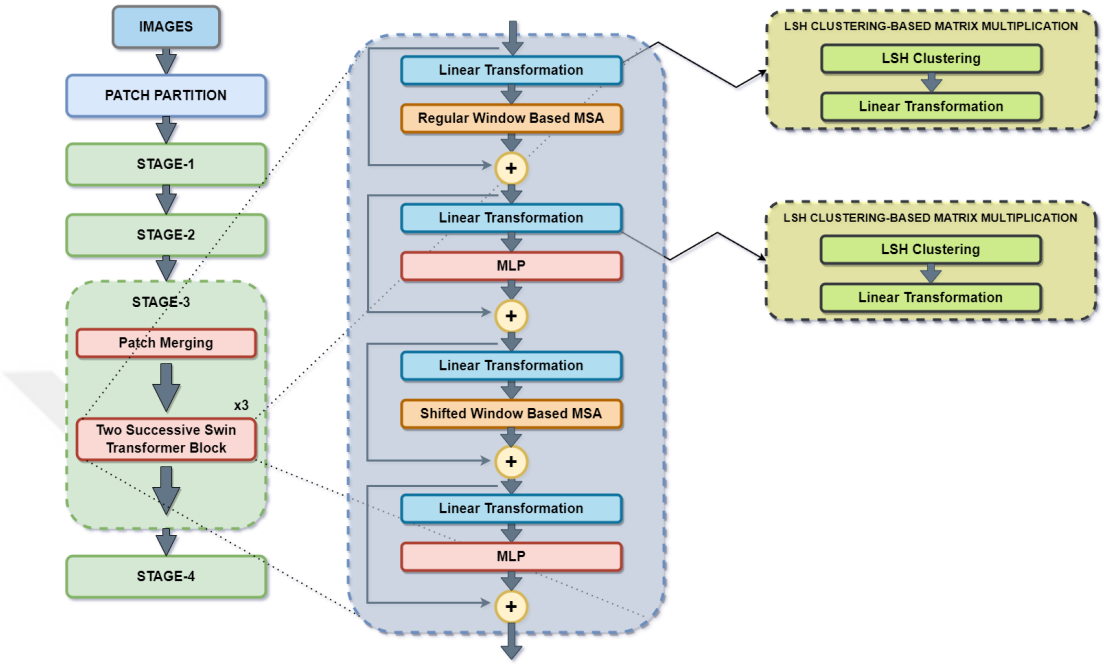

Figure 3.3: Details of LSH-based Swin Transformer architecture.

The revised Swin Transformer architecture incorporating LSH clustering, as illustrated in Figure 3.3, introduces an optimization approach designed to address the computational requirements, specifically in Stage 3. This stage is responsible for most matrix multiplication operations and, therefore, offers a great potential for optimization.

The last block on the right side of the figure shows the modifications made to the original Swin Transformer architecture. These modifications involve incorporating LSH clustering before the linear transformation steps in the third stage. The inclusion of LSH clustering allows us to simplify the workload of matrix multiplication by taking advantage of the reduced computational complexity that LSH provides.

The LSH clustering technique is utilized six times in Stage 3, preceding the initial linear transformations in the two consecutive Swin Transformer blocks. This deliberate positioning aims to optimize the computational burden by capitalizing on the significant matrix multiplication workload associated with these initial linear transformations.

The LSH clustering stages, depicted in Figure 3.4, illustrate a comprehensive workflow to improve the efficiency of matrix multiplication in neural network computations. The initial step involves extracting column groups from the QKV matrix, which is a crucial element in the self-attention mechanism of Transformer models. That matrix contains a query (Q), key (K), and value (V) matrices. These column groups serve as segmented input matrices, ready for subsequent processing.

The next step in the process is to create a hashing table for each group of columns. To do this, the high-dimensional data of the column group is projected onto random planes. The desired size of the hashing bit determines the size of these planes. The interaction between the column group and the random planes produces an integer hashing table. This hashing table effectively assigns a cluster ID to each data point in the input matrix. This step is crucial in reducing the dimension of the input data, enabling manageable and efficient computations in the following stages.

Subsequently, the procedure involves the creation of a centroid matrix through the computation of mean values for each cluster ID identified in the hashing table. The centroid matrix serves as the basis for the subsequent pivotal phase, which involves the production of the output matrix. The output matrix is obtained by multiplying the centroid matrix with a weight matrix. This multiplication operation yields a considerably smaller matrix than the initial QKV matrix while retaining the crucial characteristics and eliminating the elements that lead to computational inefficiency.

In conclusion, the workflow produces the ultimate matrix for input in subsequent neural network computations. This matrix combines the final matrices obtained from each column group to create a comprehensive representation of the input data optimized for further processing. Implementing matrix multiplication using LSH clustering presents a more efficient approach than conventional methods, as it aims to minimize computational burden and decrease the model's power consumption on edge devices without losing much accuracy in this thesis.

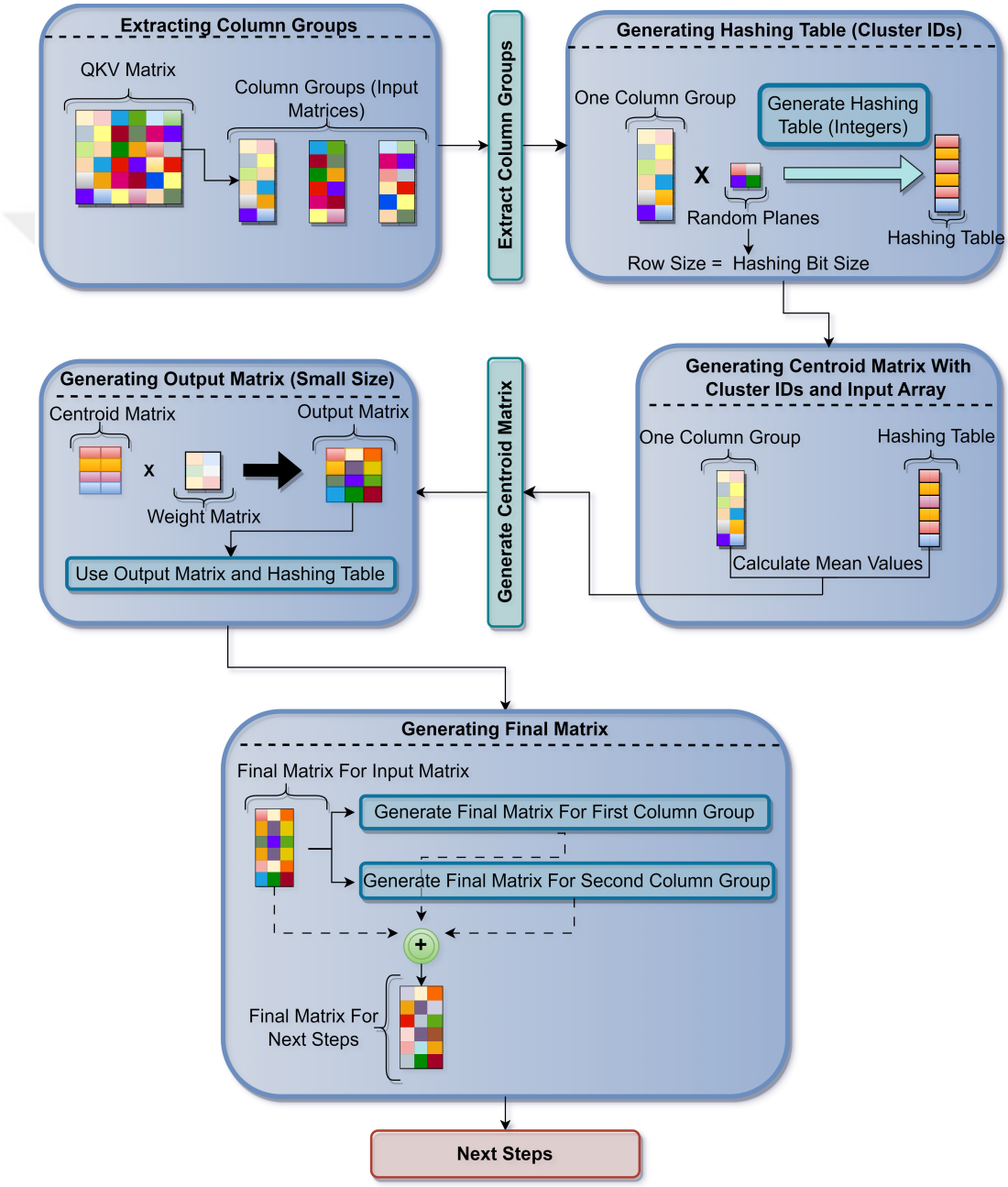

Figure 3.4: LSH Clustering stages in Swin Transformer architecture.

Chapter 4

Motivation

Examining the Swin Transformer base model through computational profiling gives performance characteristics concerning matrix multiplication operations. Timing analysis was carried out to comprehend how computational load is distributed throughout various stages of the model. The outcomes demonstrate a disproportionate focus on matrix multiplication tasks in the third stage.

Figure 4.1 demonstrates the relative computational intensity at each processing stage. Stage 3 stands out as the most demanding phase, consuming 61% of the total time, thus making it the primary area of focus for optimization.

The division of computational demand into different stages highlights a clear contrast, with Stage-1 and Stage-2 requiring relatively smaller portions of 16% and 11%, respectively, while Stage-4 accounts for 8%. Although each stage plays a role in the model's overall functionality, it is evident that Stage 3 bears a disproportionately more significant burden. This higher demand can be attributed to the intricate nature of the self-attention and feed-forward network operations within Stage 3, further intensified by the repeated use of two successive Swin Transformer blocks.

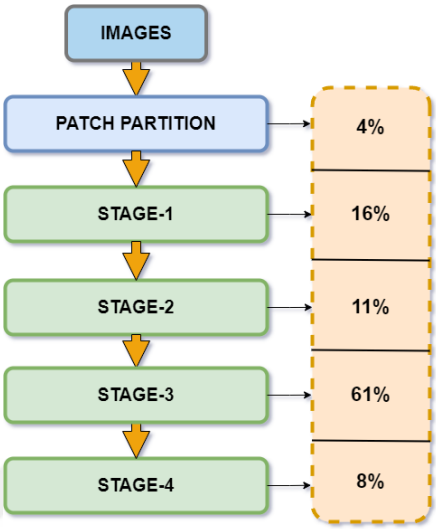

Figure 4.1: Execution time characteristics of Swin Transformer.

The Swin Transformer's design relies on linear transformations to prepare the data for attention operations, as shown in Figure 4.2. These transformations are crucial in manipulating the input data to a suitable format for subsequent attention and MLP stages. Our analysis reveals that the initial linear transformations within the self-attention workflow consume significant computational effort, comprising 64% of the processing time. Additionally, the MLPs, which come after the attention calculations, contribute to 24% of the computational effort.

The unequal distribution of time devoted to linear transformations is significant, as it highlights the specific areas where computational optimizations could benefit. In contrast, despite their intricate nature, MLPs require less time, suggesting that the primary stages of data preparation are where significant obstacles arise. The detailed analysis displayed in the results verifies that the linear transformations occurring within the linear transformations mechanism of Stage 3 are the primary areas that can be optimized.

Figure 4.3 illustrates the effectiveness of LSH clustering in optimizing an input matrix to reduce computational costs efficiently. The process commences with an input matrix that signifies the data to be processed, typically a set of feature vectors where each row represents a data sample and each column denotes a feature. In standard neural network procedures, this input matrix would be subjected to

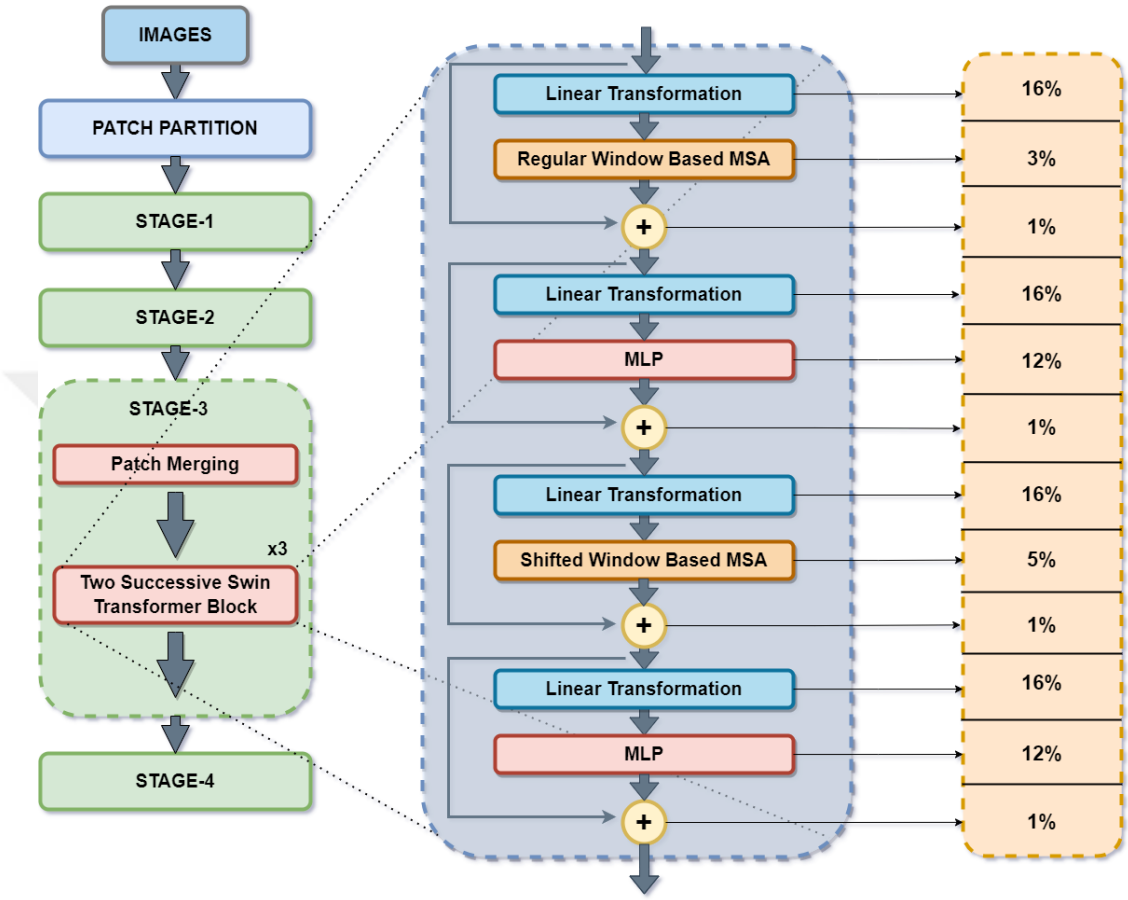

Figure 4.2: Swin Transformer timing analysis within two successive transformer blocks.

multiplication by a weight matrix to produce a final output matrix. The weight matrix encompasses learned parameters that transform the input data into a new feature space. The outcome of this multiplication is the final matrix containing the transformed data prepared for further processing. Nonetheless, this direct multiplication can be computationally demanding, especially for large matrices. To streamline this procedure, the input matrix undergoes LSH clustering. The LSH algorithm clusters similar rows (data samples) based on their feature resemblance, assigning each row to a specific cluster. This diminishes the necessity for numerous unique computations, as similar data points can share computations. Each row in the input matrix is allocated a cluster ID indicating its membership in a particular cluster. This clustering process considerably simplifies the data. A centroid matrix is computed instead of utilizing the entire input matrix for

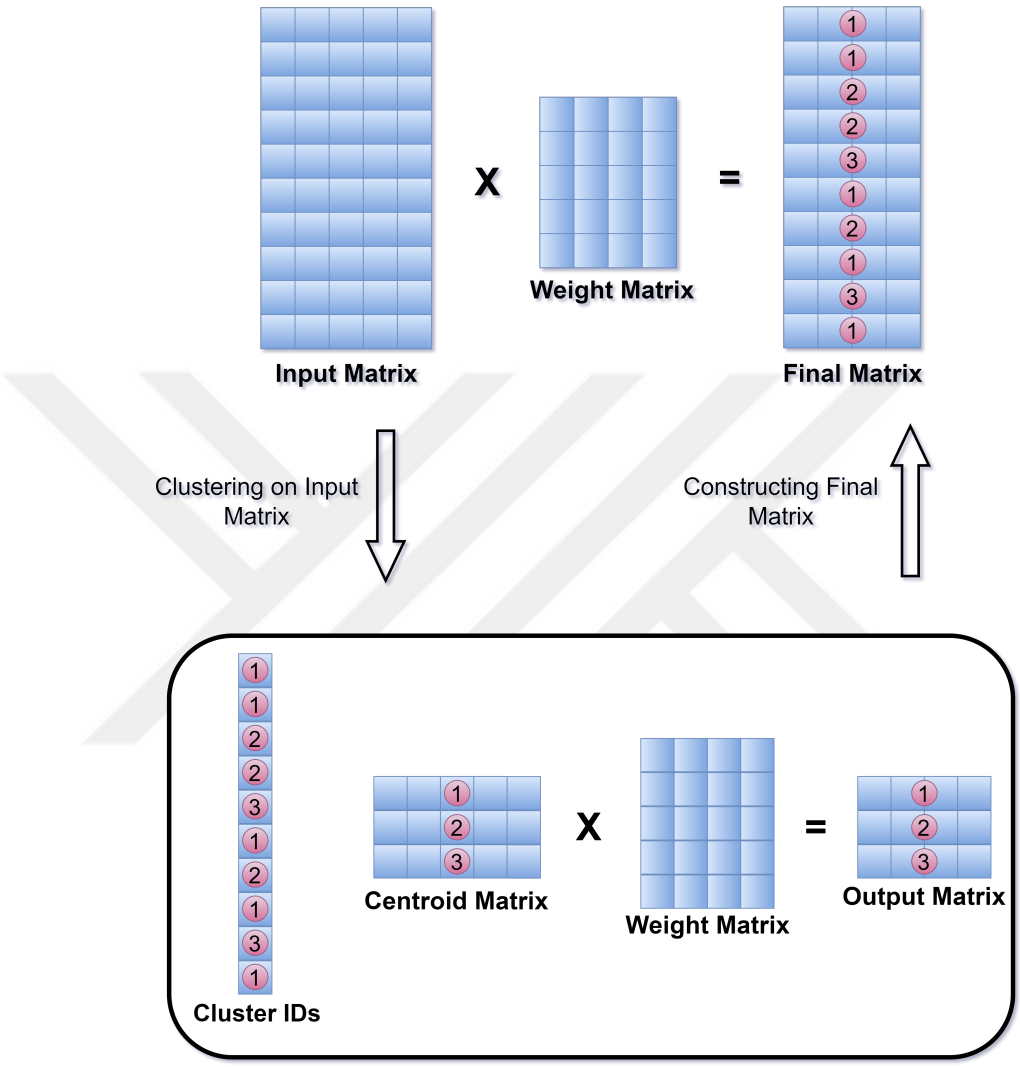

Figure 4.3: Illustration of using LSH for clustering to reduce computational cost.

multiplication. This matrix is obtained by averaging the data points within each cluster. Each row in the centroid matrix represents a cluster centroid, capturing the central tendency of the cluster members. The centroid matrix is then multiplied by the weight matrix, resulting in the output matrix with significantly fewer computations compared to the original input matrix multiplication. It operates on condensed cluster representations rather than individual data points. The final output matrix, produced from the optimized multiplication, is a concise representation that retains the fundamental characteristics of the input data while substantially reducing the computational burden.

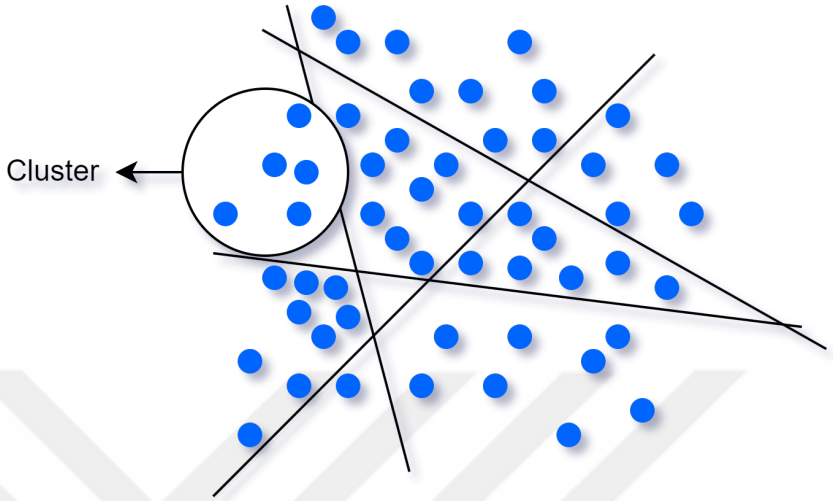

Figure 4.4: Illustration of LSH clustering within a two-dimensional plane.

Figure 4.4 visually depicts LSH clustering within a two-dimensional plane. In this scenario, data points are scattered throughout the plane, and LSH clustering is employed to categorize them according to their proximity. Each blue dot symbolizes a data point within the 2D feature space, potentially originating from a dataset containing two features. The diagram features multiple intersecting lines representing the hash functions utilized in LSH. Each hash function serves to partition the space into distinct regions or clusters. Data points falling within the same region or close, as determined by the hash functions, are grouped into clusters. A highlighted circle within the figure exemplifies one such cluster, demonstrating the aggregation of nearby data points.

LSH clustering significantly improves computational efficiency by reducing the number of unique computations needed but achieves this by approximating data points with cluster centroids. This results in a loss of resolution as individual variations within clusters are not captured in the centroids. The original data's granularity is diminished as LSH clustering averages data points within clusters, potentially leading to the loss of fine details specific to individual data points and impacting computation precision. While this reduction in data resolution can decrease accuracy by not capturing subtle variations, the extent of this accuracy loss varies based on clustering granularity and data characteristics.

Despite potential compromises in accuracy, LSH clustering offers numerous advantages compared to other optimization algorithms. These include improved computational efficiency, scalability for large datasets, flexibility for integration into deep learning models, and energy efficiency. LSH clustering optimizes the input matrix while balancing efficiency and accuracy by grouping similar data points and reducing computational costs. It is particularly effective in edge computing scenarios where resource optimization is crucial. Despite potential trade-offs in resolution and accuracy, the overall benefits of LSH clustering make it a preferable choice for various applications when we compare other clustering methods [34], [37].

Chapter 5

Hardware Accelerator Design

The general architecture of our hardware accelerator is shown in Figure 5.1. The LSH algorithm serves as the central component and is known for its ability to simplify high-dimensional data by creating hash tables. This, in turn, enables a more efficient matrix multiplication.

The overall FPGA design comprises a Processing System (PS) and Programmable Logic (PL). The PS side of the system serves as the central control unit, facilitating communication with the PL side, managing data transfers through the DMA, and storing it in the DDR memory. Its primary role is to ensure a smooth and efficient data flow into the PL side, thus enabling complex computational processes. On the other hand, PL implements the LSH-based Swin Transformer implementation.

The transformation begins with creating hash tables that group similar data points. Subsequently, centroids are calculated to capture the fundamental characteristics of these data clusters. Ultimately, this process produces an output matrix, which serves as a condensed and refined depiction of the input data. This output matrix is then prepared for further work in the Swin Transformer algorithm.

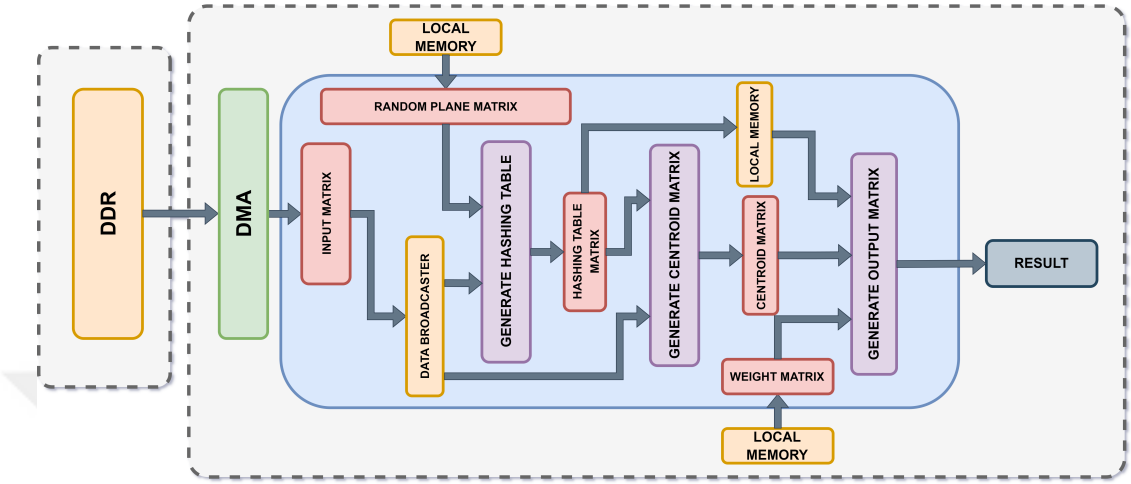

Figure 5.1: General hardware accelerator design representing the LSH-based Swin Transformer implementation.

Regarding efficiency and speed, local memories on the PL side act as fast-access memory points for the Random Plane and Weight Matrices, which are essential for the operations of the LSH algorithm. While these local memories are implemented using FPGA Block Memories (BRAMs), they could be custom-made in Application Specific Integrated Circuits (ASIC). The DMA assumes a critical role as the central link connecting the DDR memory and the PL side. Simultaneously, the systolic arrays embedded within the PL side serve as exemplary parallel processing models, effectively expanding the limits of what can be accomplished within the confines of an FPGA, which accomplishes matrix multiplication.

5.1 Hash Table

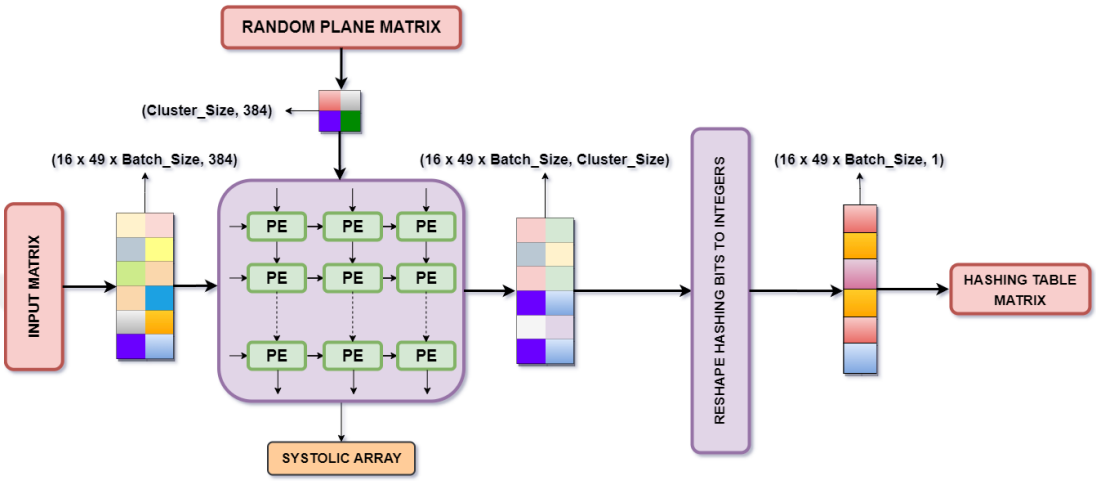

Figure 5.2: Hash Table details and implementation.

Hash Table, shown in Figure 5.2, uses a systolic array configuration. This array is organized to facilitate matrix multiplication calculations between high-dimension input data and a predetermined random plane matrix.

The input matrix contains the feature vectors and is organized as a multi-dimensional structure of (16 x 49 x BatchSize, 384) size. The variable parameter BatchSize represents the number of data instances processed simultaneously, and the dimension of 384 represents the feature size. Random plane matrix has dimensions of (ClusterSize, 384). Here, ClusterSize refers to the quantity of hash functions that are used. This matrix consists of random vectors, each functioning as a hyperplane in the LSH algorithm's high-dimensional space. By storing this matrix in the local memory, the design guarantees swift access to these essential vectors, effectively removing obstacles hindering the hashing operation's efficiency.

Each Processing Element (PE) conducts concurrent dot product operations in the systolic array. The array's structure, known for its inter-PE communication, facilitates a continuous propagation of intermediate outcomes. This propagation mechanism guarantees prompt accessibility of dot product results for subsequent operations, thereby improving the overall efficiency of the hashing process. After

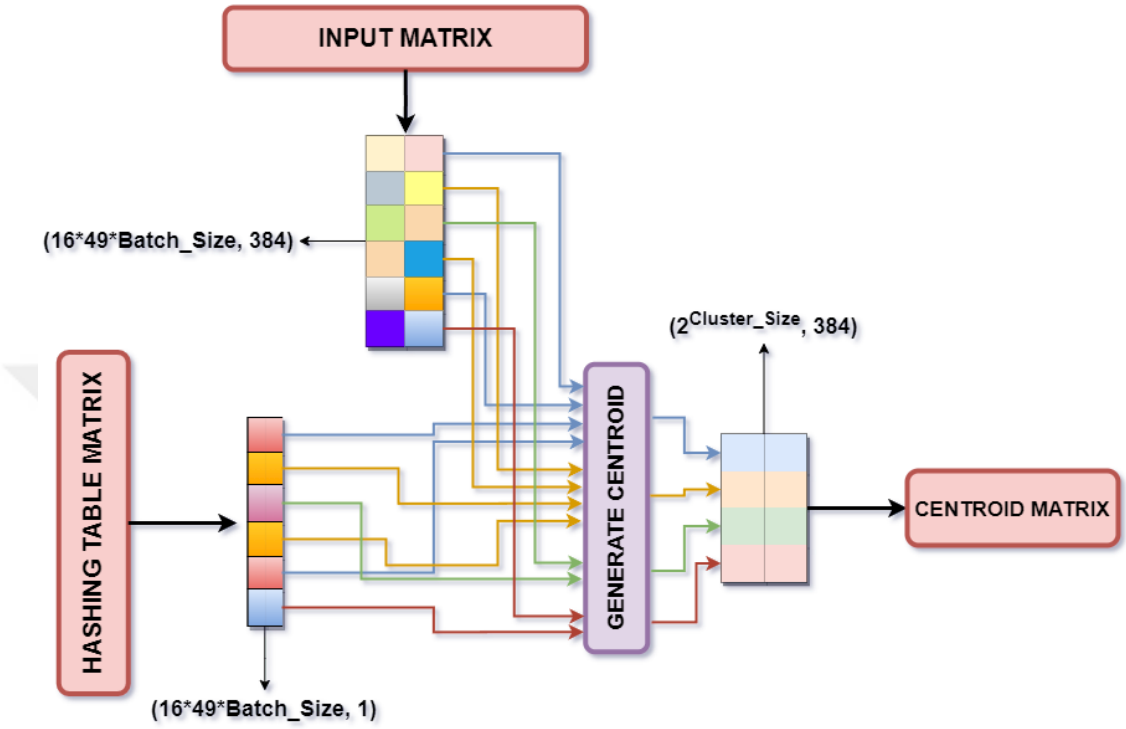

Figure 5.3: Centroid Matrix generation process.

performing the matrix multiplications, the resulting hashed values are sent to the next stage, where the bit hash space is transformed into integer units. Converting the continuous outputs of the dot products into integers is a step that assigns these values as indices in the hash table matrix, which can be described as clusters for each row in the input matrix.

5.2 Generate Centroid Matrix Block

As shown in Figure 5.3, the input matrix is structured as a matrix with (16 x 49 x BatchSize, 384) dimensions. This is the same input matrix that is utilized in the previous block. This input matrix is broadcasted at the first stage of matrix reading from DDR memory. This data, which represents the complex space commonly encountered in deep learning tasks, is systematically compared to the hashing table matrix. The hashing table matrix, with dimensions (16 x

49 x BatchSize, 1), functions as an index that associates each data point with its corresponding cluster in the centroid matrix created in the previous block.

The *Centroid Matrix* block operates through aggregation. Each item in the hashing table corresponds to a cluster in the centroid matrix, which has ($2^{ClusterSize}$, 384) dimensions. In the given process, the block gathers the vectors from the input matrix with the same hash index for each cluster. These vectors are combined to form an intermediate sum matrix. At the same time, a count of the aggregated vectors is kept. Subsequently, the block performs a division operation to compute the average vector, or centroid, for that particular cluster by dividing the sum of the vectors by the count. As a result, the centroid matrix is obtained, which provides a reduced representation of the input data, where the dimensions are now determined by the number of clusters instead of the original data points.

The dimension reduction accomplished using the centroid matrix is not simply a compression technique for data; instead, it is a transformation that maintains the structure and distribution of the original space. This transformation enables faster and more efficient computations in subsequent stages. The *Centroid Matrix* block effectively condenses the fundamental characteristics of the data by mapping the high-dimensional data points to centroids, allowing for quick processing without compromising the integrity of the information much.

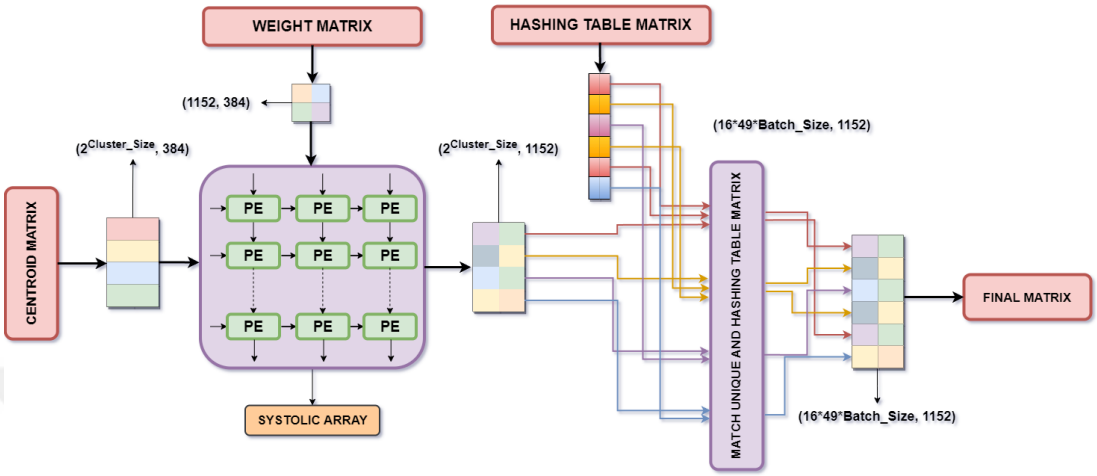

Figure 5.4: Final Matrix generation process.

5.3 Output Matrix

The *Output Matrix* module within the accelerated LSH clustering architecture is created to produce the ultimate output matrix by combining the previously generated centroid matrix and weight matrix. This module represents the final stage of the LSH acceleration process, which gives an output matrix that can be utilized in the next stage of Swin Transformer architecture. The details of this module are provided in Figure 5.4.

At the core of this block lies a systolic array, which is a well-organized set of PEs designed to effectively carry out matrix multiplication operations between the centroid matrix and the weight matrix. These PEs are similar to the ones defined before. The centroid matrix is structured as a $(2^{ClusterSize}, 384)$ matrix and contains concise information derived from the input matrix, where every centroid represents a cluster of input features. On the other hand, the weight matrix has dimensions (1152 and 384) and holds the acquired parameters that will be utilized to transform the centroids and produce the final feature space. Every processing element (PE) is assigned a centroid and weight matrix segment in the systolic array. By performing a series of calculated multiplications and accumulations in a coordinated manner, the PEs collectively produce the unique matrix. This matrix has a compressed version of the final matrix.

After systolic processing is completed, the partial products are transmitted to the *Match Module*. In this module, the hashing table matrix is merged with the compressed computed results from the systolic array. The hashing table matrix, which has dimensions of (16 x 49 x BatchSize, 1), contains the indices that associate each batch entry with its respective cluster. This association guarantees that the effects of each centroid on the output matrix are positioned accurately, thereby maintaining the inherent spatial relationships of the initial input data.

The match module is the concluding stage in combining the fragmented outcomes by utilizing the hashing table as a reference to merge the inputs from the corresponding centroids. The resulting output of this module is the final matrix, with (16 x 49 x BatchSize, 1152) dimension, which signifies the transformed feature space suitable for the following stages in Swin Transformer architecture.

Chapter 6

Experimental Evaluation

6.1 Setup

We evaluate three different hardware settings for the Swin Transformer. First, we use a default GPU-based implementation. We call experiments with that setup as *GPU*. Then, we use the GPU platform with LSH. We call experiments with that setup *LSH-GPU*. Finally, we implement a custom accelerator on an FPGA. We call experiments with that setup *ACC*.

For *GPU* and *LSH-GPU* experiments, we use NVIDIA Jetson Xavier NX 16 GB, an advanced embedded system specially designed for edge AI applications. The Jetson Xavier NX, known for its compact size and high computational abilities, is a well-suited platform for deploying and evaluating the Swin Transformer models incorporating LSH clustering [38]. The NVIDIA Jetson Xavier NX is powered by a Volta-based GPU with 384 CUDA and 48 Tensor cores. It also includes a six-core NVIDIA Carmel ARMv8.2 64-bit CPU. The system has 16 GB of 128-bit LPDDR4x memory, providing sufficient bandwidth for the demanding data operations the Swin Transformer model needs. In terms of storage, the device has a built-in 16 GB eMMC 5.1 Flash and an SD card slot for additional capacity, which is utilized for recording experimental data. Additionally, the board

offers a maximum power of 20 Watts, enabling all six cores to function [38]. The Jetson Xavier NX operates on the NVIDIA JetPack SDK, which incorporates the CUDA-11.3 accelerated computing stack and an extensive range of artificial intelligence tools and frameworks [39].

Furthermore, we use Python 3.8, Pytorch 3.11, Torchvision 0.9.0, and OpenCV 4.4.0.46 for software development [40]. Data is stored and used in float16 precision in the Swin Transformer by default. The Python time library is utilized to analyze speedup and calculate the time spent updating the Swin Transformer architecture. Moreover, Jetson devices provide Jetson Power GUI for power analysis, demonstrating power and other parameters, such as temperatures, in detail [41].

For *ACC* experiments, the ZCU102 board is used. This setup includes the Xilinx Zynq UltraScale+ MPSoC and offers a reliable infrastructure for implementing machine learning algorithms such as LSH clustering. This is due to the board's ARM processors and programmable logic [42]. The ZCU102 evaluation kit has a PS comprising a quad-core ARM Cortex-A53 central processing unit and a dual-core ARM Cortex-R5 real-time processor. On the other hand, the PL side of the kit features an FPGA fabric that allows for customization and adaptability in accelerator design. Additionally, the board is equipped with 4 GB of DDR4 memory, which both PS and PL utilize to handle data. Moreover, the FPGA consists of 600K LUTs, 32.1 Mb RAM, and 2520 DSP slices used in accelerator jobs [42].

The Xilinx Vivado Design Suite 2022.1 is employed for the software development and synthesis of FPGA logic. This integrated development environment provides tools for designing, programming, and debugging the hardware accelerator. Vivado's high-level synthesis capabilities enable converting C/C++ code into hardware descriptions that can be implemented on the FPGA, thereby expediting the design process [43]. In the FPGA setup, 8-bit integer precision is used, where data is converted from float16 to int8 when saved in DDR and local memory.

The experimental setup minimizes delays and maximizes the data transmission rate. The input data is moved from the DDR memory to the LSH IP core through the high-speed AXI bus [44]. The core performs tasks such as hashing and clustering on the data, and the results are returned to the DDR memory. The ARM CPU manages the entire process, including initiating data transfers, activating the LSH computations, and handling the results. An integrated logic analyzer (ILA) is utilized for speedup analysis, which Vivado provides [45]. With the help of this tool, time spent and latency on an LSH-based hardware accelerator are calculated. In addition, the Vivado Report Power tool is used for power analysis, which shows power estimation results in detail [46].

6.1.1 Datasets

The widely used Imagenet-1K dataset, ILSVRC 2012, assesses the effectiveness of the tested image classification algorithms [47]. It encompasses a varied assortment of images that are systematically arranged according to the WordNet hierarchy, wherein images are categorized into "synsets" or groups of synonyms based on the depicted object [47], [48].

The dataset covers 1000 distinct object classes, offering a range of challenges across various categories. It consists of 1281167 training images, utilized for training machine learning models, along with a validation set of 50000 images and a test set of 100000 images, employed for assessing the effectiveness of these models. The Imagenet-1K dataset comprises numerous images, each assigned a label representing one of the 1000 available classes [47].

The CIFAR-100 dataset, used as the second dataset in this work, is widely recognized as a prominent set of images utilized to assess machine learning algorithms for image classification [49]. Originating from the Canadian Institute for Advanced Research, the CIFAR-100 dataset represents an enhanced iteration of CIFAR-10, presenting more incredible difficulty due to its augmented range of categories [49]. CIFAR-100 is a dataset comprising 60000 color images, each 32x32 pixels in size. The dataset is divided into 100 classes, with 600 images

representing each class. These classes encompass a wide range of objects and living organisms. The classes are also categorized into 20 superclasses, allowing for a more general classification approach. The dataset is split into a training set, containing 50000 images, and a test set, containing 10000 images. This division ensures a thorough evaluation framework for supervised learning models [49].

Every image in CIFAR-100 is exhaustively labeled with a precise "fine" classification, encompassing a range of objects including but not limited to bottles, bicycles, beavers, and bees, as well as various species of animals. Additionally, a "coarse" label is assigned to indicate the overarching superclass of the image, such as aquatic mammals or household furniture, thereby facilitating an evaluation of the classification models' efficacy at both detailed and broader levels of categorization [49], [50].

6.1.2 Default Parameters

Our primary goal is to find the most effective configuration for the Swin Transformer model with LSH clustering. To this end, some experiments are conducted to systematically analyze the impact of different parameters, such as batch size, number of clusters, and the proportion of linear transformation parts where LSH clustering is applied. The aim is to determine the combination of parameters that balance minimizing accuracy loss and maximizing speedup when running inference.

Table 6.1: Default parameters used in the experiments.

Parameter	Value
Batch Size	16
Cluster Number	8
% of Linear Transformations	25%

We applied an exponential progression for the batch size selection. We tested with 4, 8, 16, 32, and 64. In addition to varying the batch size, the number of clusters utilized for Locality Sensitive Hashing (LSH) was also adjusted. The options examined included 8, 16, 32, and 64 clusters. This particular parameter directly impacts the level of detail within the hashing space and consequently affects the accuracy of the clustering operation. Moreover, it influences the quality of the resulting centroids as well as the computational intricacy of the LSH procedure.

Furthermore, the utilization of LSH was adjusted as a proportion of the overall 24 linear transformation components within the Swin Transformer algorithm. Through experimentation at intervals of 12.5% up to 62.5%, the investigation aimed to determine the optimal threshold at which the advantages of enhanced speed would surpass any potential drawbacks to the model's efficacy.

The default parameters shown in Table 6.1 are selected after extensive testing with GPU and accelerator settings. It was observed that the optimal configuration consisted of a batch size of 16, a cluster number of 8, and applying LSH clustering to 25% of the linear transformation parts. This combination yielded the most favorable balance, improving processing speed while maintaining a negligible decrease in accuracy.

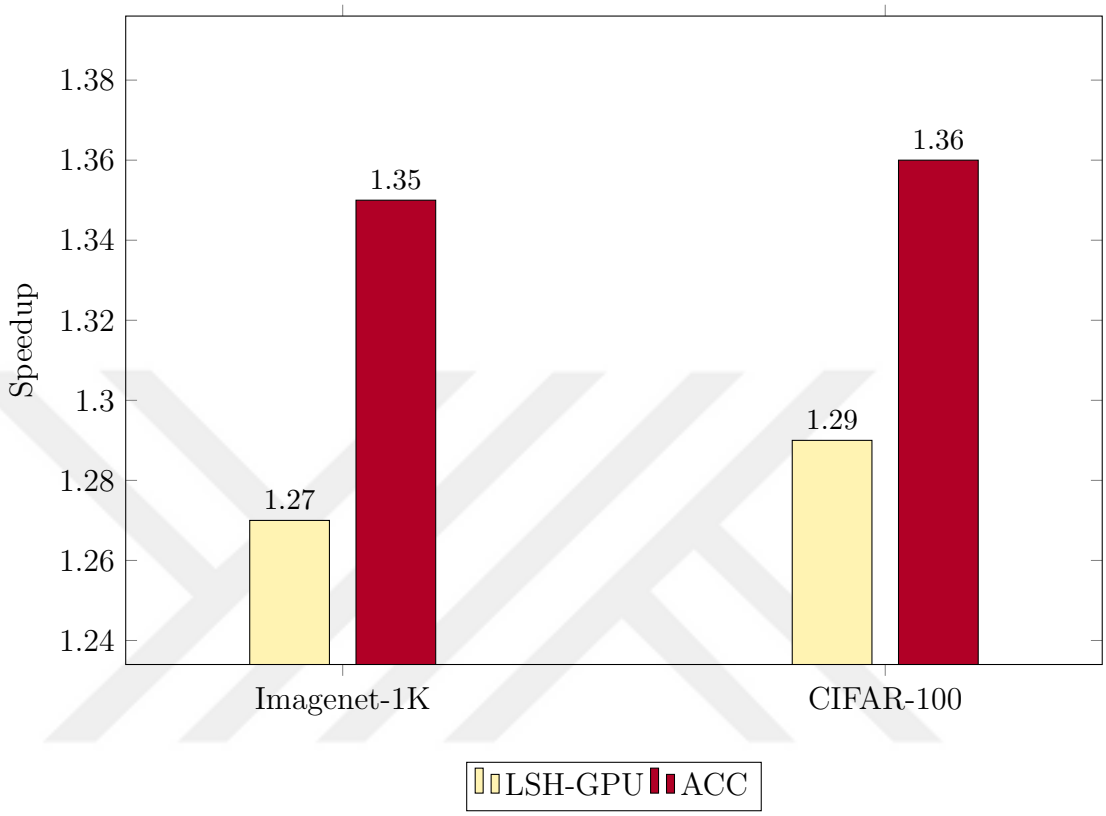

Figure 6.1: Comparison of LSH-GPU and ACC speedup with GPU.

6.2 Experimental Results

For experimental results, firstly, the speedup achieved by the Swin Transformer with LSH-GPU and ACC is assessed and compared to GPU, as shown in Figure 6.1. The results for both datasets are very close, showing the scalability of LSH clustering. The graph demonstrates a speedup of about 1,28x on the LSH-GPU, indicating a reduction in processing time compared to conventional implementations without hardware acceleration. In contrast, ACC exhibits a further improvement with a speedup of about 1,35x. This enhancement highlights the effectiveness of using an accelerator to expedite complex computational tasks. The disparity in speedup factors can be attributed to the parallel processing capabilities and customized hardware optimization inherent to our accelerator.

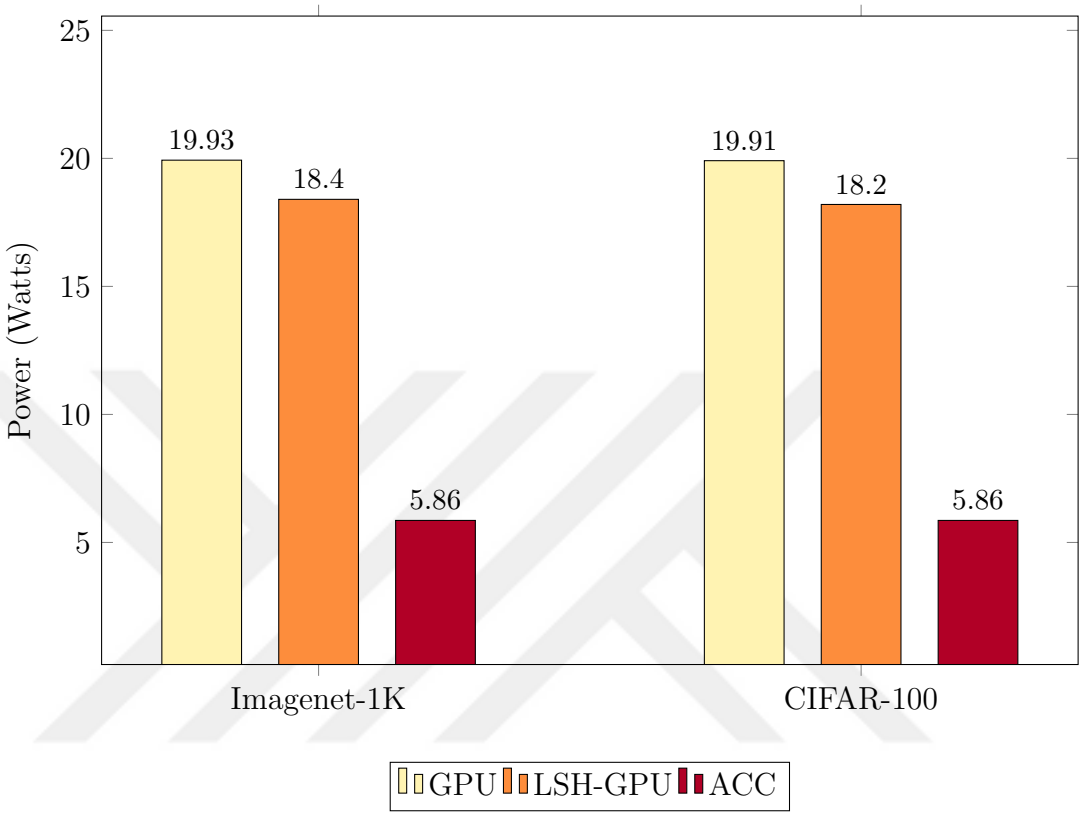

Figure 6.2: Comparison of LSH-GPU and ACC power consumption with GPU.

Figure 6.2 compares LSH-GPU and ACC power consumption with GPU. It is tested with two separate datasets with similar outcomes. GPU showcases a power usage of about 19.9 Watts, reflecting the elevated energy demands of the CPU and GPU cores necessary for deep learning tasks. For LSH-GPU, power decreased a little, which can be negligible compared to ACC results. ACC exhibits a substantially lower power consumption of about 5.86 Watts, indicating an almost 75% reduction in energy requirements. This notable disparity underscores the hardware accelerators' ability to execute the same computations more efficiently, rendering it a favorable platform for energy-constrained applications and deployments prioritizing power efficiency. One would expect the power consumption to be much lower than this in an ASIC implementation as FPGA designs consume significantly more than ASIC implementation for custom hardware [51].

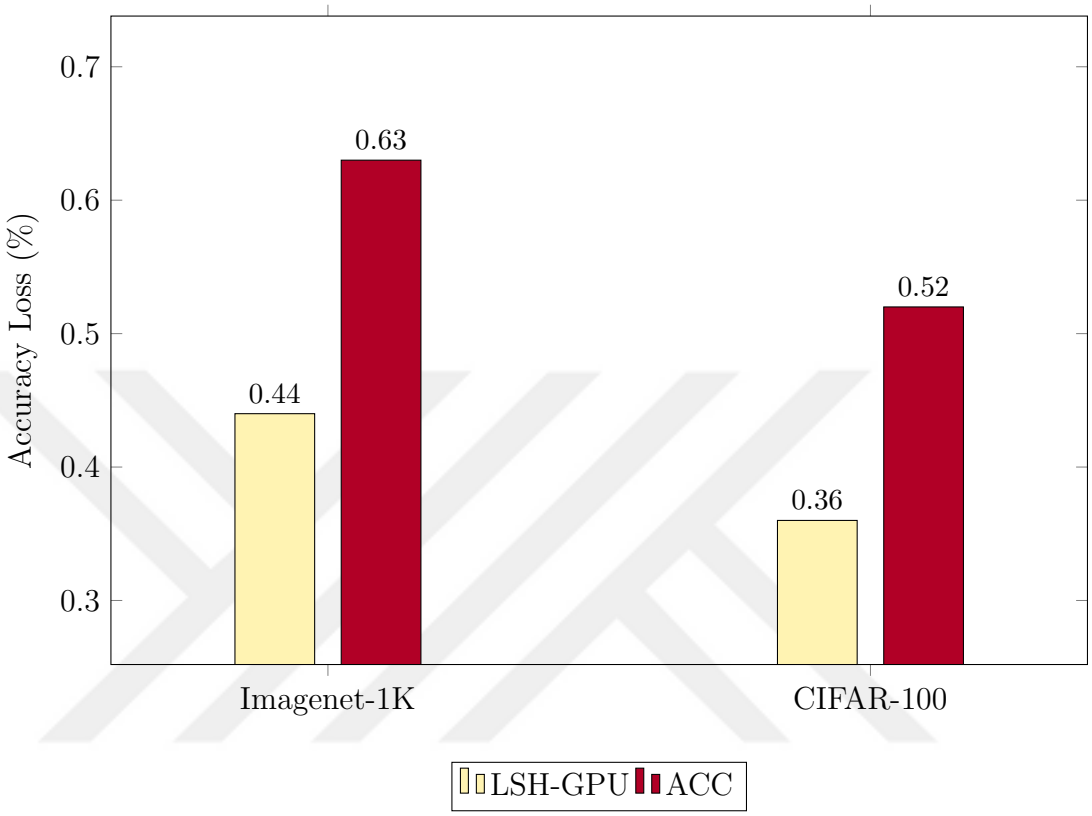

Figure 6.3: Comparison of LSH-GPU and ACC accuracy loss with GPU.

Next, we analyze the effects on accuracy loss. As shown in Figure 6.3, our approach preserves the model's accuracy with negligible loss. More specifically, the accuracy loss is 0.44% for Imagenet-1K and 0.36% for the CIFAR-100 dataset in LSH-GPU. In ACC, accuracy losses are higher because int8 precision is used. However, note that this minor increase in accuracy loss is still negligible. The experiments performed for optimal parameter settings provide a positive direction for future investigations and potential practical implementations of accelerated deep learning models.

6.3 Sensitivity Analysis

In this section, we experiment with various design parameters. This critical analysis plays a pivotal role in elucidating the impact of alterations in input parameters on the outcomes, thus enhancing our comprehension of the model's performance across diverse circumstances. In all experiments, LSH-GPU and ACC are compared with the GPU using the default parameters unless stated otherwise.

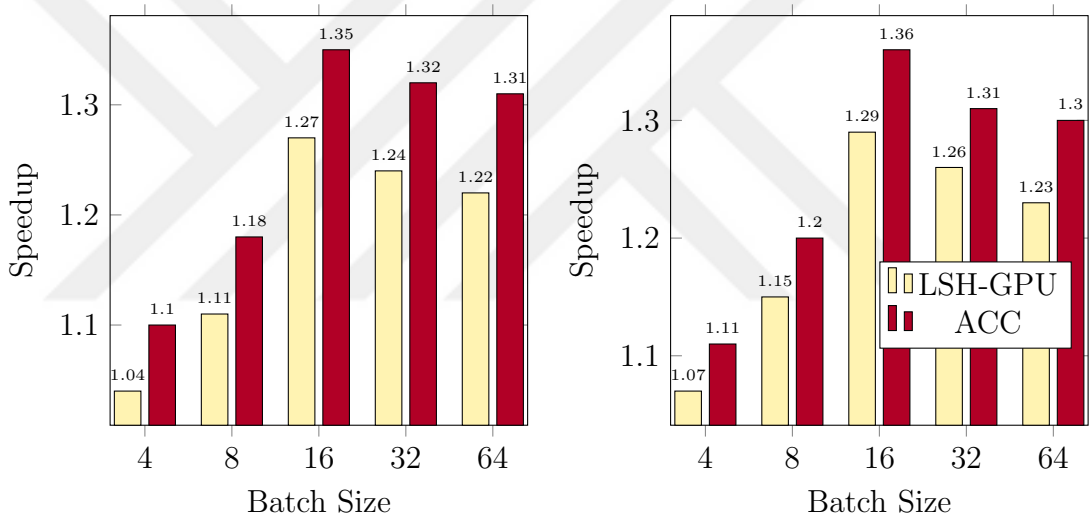

Figure 6.4: Speedup with different batch size values for Imagenet-1K (left) and CIFAR-100 (right).

The performance of the Swin Transformer model is improved when utilizing LSH-GPU, leading to varying speed enhancements across different batch sizes. As can be seen in Figure 6.4, for the Imagenet-1K dataset and CIFAR-100 dataset, LSH-GPU shows a maximum speedup of 1.27x and 1.29x, respectively, for a batch size of 16. In contrast, the ACC platform exhibits higher efficiency, showing a maximum speedup of 1.35x and 1.36x, respectively, for batch size 16. For both datasets, the trend is very similar. Speedup increases with batch size increasing until batch size 16, and then values decrease with larger batch sizes. The speedup values do not increase as batch size increases because after a batch size of 16, generating centroid matrix and other LSH-related jobs takes more time because

of big matrix sizes. Therefore, the best speedup values are observed at batch size 16.

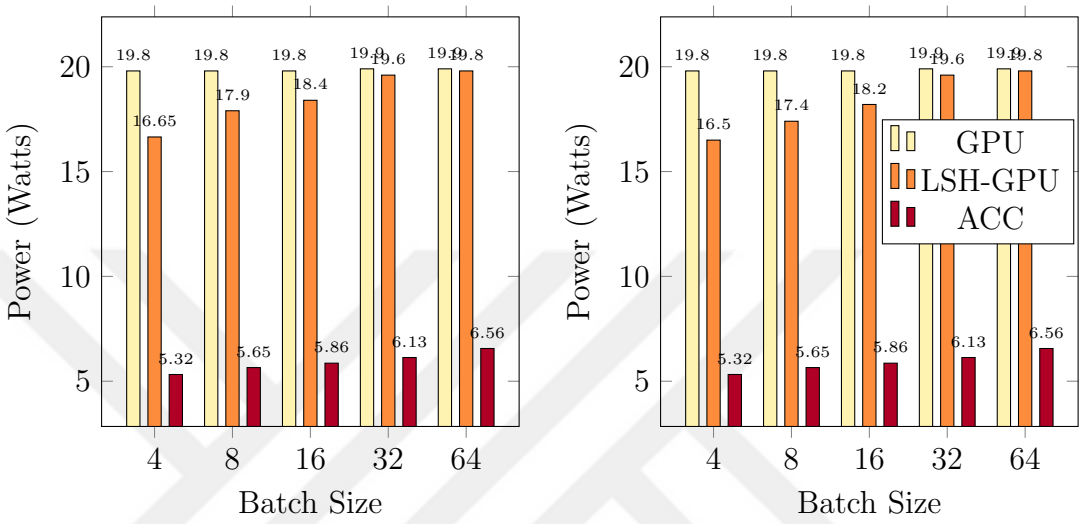

Figure 6.5: Power consumption with different batch size values for Imagenet-1K (left) and CIFAR-100 (right).

Figure 6.5 illustrates the power consumption of the Swin Transformer model when running the Imagenet-1K and CIFAR-100 dataset on both LSH-GPU and ACC platforms compared to GPU with varying batch sizes. GPU shows a consistent power usage of approximately 19.8 Watts. When the LSH-GPU is applied, the power consumption decreases slightly to about 16 Watts for a batch size of 4 for both datasets and gradually increases with larger batch sizes. LSH-GPU power values are lower than GPU's because of the LSH clustering algorithm. The algorithm reduces the number of matrix multiplications, impacting power consumption. On the other hand, ACC demonstrates a notable decrease in power consumption, requiring only 5.32 Watts at a batch size of 4 for both datasets and an increase with large batch sizes. Increasing batch size means more power consumption values because a larger batch size means a bigger matrix. This causes higher power consumption when the batch size value is increased. ACC consumes less power than both GPU implementations since the accelerator significantly reduces the execution time, resulting in less power [52]. Furthermore, inherent parallelism in the hardware accelerator provides power savings [53].

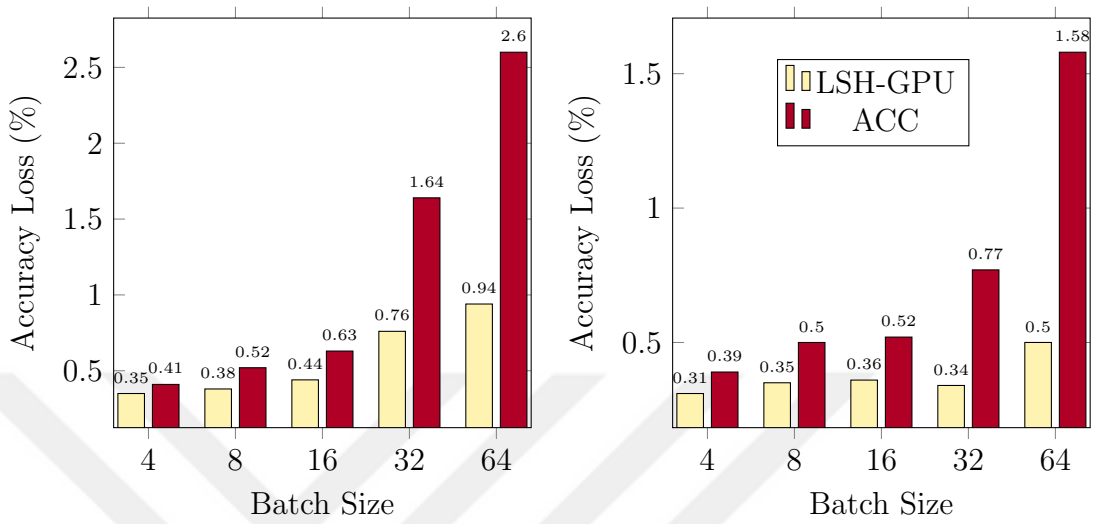

Figure 6.6: Accuracy loss with different batch size values for Imagenet-1K (left) and CIFAR-100 (right).

Next, we examine the impact of batch size on accuracy loss for the Imagenet-1K and CIFAR-100 dataset on both the LSH-GPU and ACC in Figure 6.6. The results show that on LSH-GPU, the accuracy loss starts at a very low value for both datasets for a batch size of 4 and gradually increases as batch size increases. In contrast, ACC experiences a significantly higher accuracy loss because of int8 precision. Also, accuracy loss increases with higher batch sizes for ACC. Accuracy loss in ACC and LSH-GPU increases with batch size increase for both datasets because as batch size increases, matrix size increases. A larger matrix results in decreasing LSH clustering resolution because cluster sizes increase. In some cases, ACC and LSH-GPU behave differently because, in ACC, int8 precision is used. This causes a dramatic increase in accuracy loss after a batch size of 16 because of low precision compared to float16 used in LSH-GPU. Based on the results collected for performance, power, and accuracy, the best batch size, giving the best combination of results, is 16. Therefore, we set the default batch size parameter to 16.

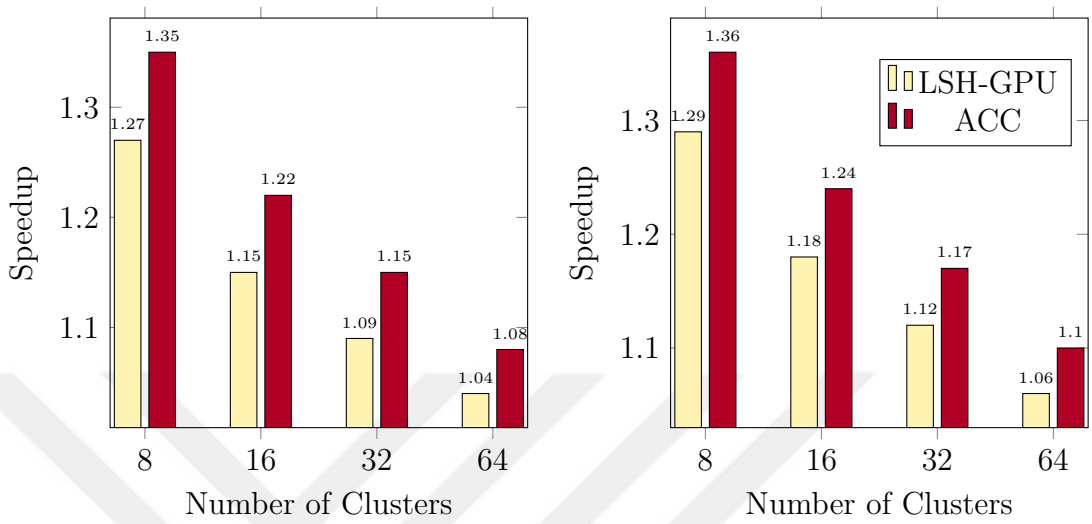

Figure 6.7: Speedup with different numbers of clusters for Imagenet-1K (left) and CIFAR-100 (right).

Figure 6.7 illustrates the variations in speedup with different numbers of clusters in processing the Imagenet-1K and CIFAR-100 dataset on LSH-GPU and ACC. LSH-GPU achieves its highest speedup of 1.27x and 1.29x with 8 clusters, respectively. However, as the number of clusters increases, the speedup gradually reduces for both datasets. On the other hand, ACC initially exhibits a higher speedup of 1.35x and 1.36x with 8 clusters, respectively. As the number of clusters increases, the speedup reduces for both datasets, as in LSH-GPU. This is due to the fact that it takes more time to generate more clusters. Also, for each cluster, the algorithm needs to calculate cluster centers. Speedup decreases because generating clusters and calculating centroid matrices take more time for a higher number of clusters.

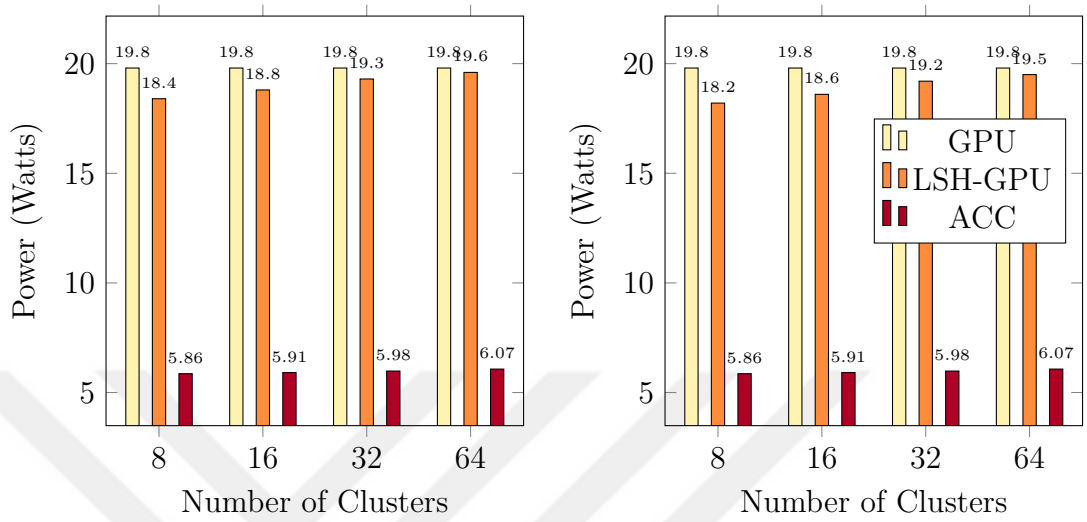

Figure 6.8: Power consumption with different numbers of clusters for Imagenet-1K (left) and CIFAR-100 (right).

Figure 6.8 illustrates the power consumption in Watts for the Imagenet-1K dataset for varying cluster sizes. The power consumption of the GPU remains consistently high at approximately 19.8 Watts, regardless of the cluster size and dataset. However, when LSH-GPU is applied, a slight increase in power usage is observed, particularly noticeable at higher cluster sizes. For instance, the power consumption reaches up to about 19.6 Watts for a cluster size of 64 from about 18.4 Watts for a cluster size of 8 for both datasets. On the other hand, ACC exhibits a substantial reduction in power consumption across all cluster sizes. It starts at 5.86 Watts for a cluster size of 8 and increases to 6.07 Watts for a cluster size 64 for both datasets. For both datasets, power consumption increases with a higher number of clusters. That is because a high number of clusters results in more centroid matrix calculations.

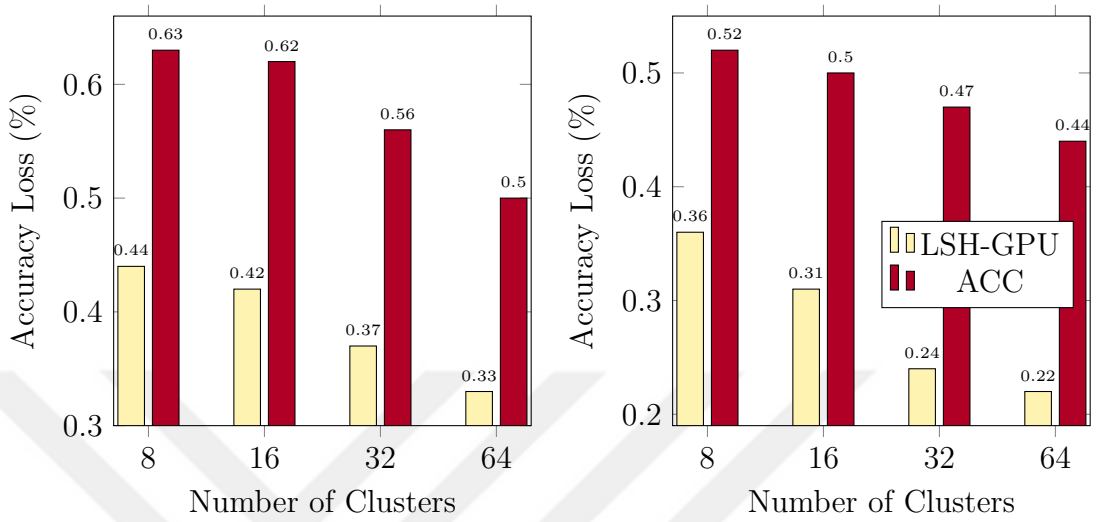

Figure 6.9: Accuracy loss with different numbers of clusters for Imagenet-1K (left) and CIFAR-100 (right).

Figure 6.9 illustrates the accuracy loss percentage for the Imagenet-1K and CIFAR-100 datasets when comparing different cluster sizes on LSH-GPU and ACC. ACC and LSH-GPU exhibit a slight decrease in accuracy loss as the number of clusters increases. This suggests that larger cluster sizes result in better accuracy retention for LSH-GPU and ACC. This follows because as the number of clusters increases, the number of centers also increases. This results in a higher matrix precision, thereby reducing accuracy loss. For ACC, accuracy loss is higher, as one can expect since int8 precision is used for the matrix. When speedup, power, and accuracy are considered together, 8 clusters are observed to give the best results and are set to be the default number of clusters.

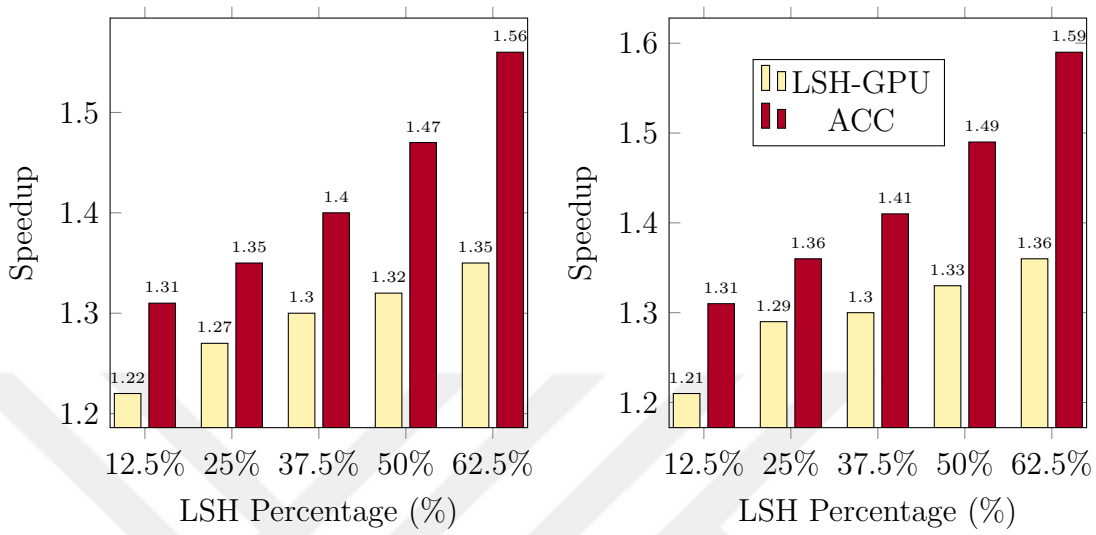

Figure 6.10: Speedup with different LSH percentage values for Imagenet-1K (left) and CIFAR-100 (right).

Figure 6.10 illustrates the speedup achieved by implementing LSH on different percentages of matrix multiplication operations for the Imagenet-1K and CIFAR-100 datasets, using LSH-GPU and ACC. In the case of LSH-GPU, there is a positive correlation between speedup and the extent of LSH application. The speedup increases from about 1.22x and 1.21x at 12.5% LSH to a peak of 1.35x and 1.36x at 62.5% LSH for both datasets, respectively. Similarly, ACC also benefits from LSH. The maximum speedup of 1.56x and 1.59x is achieved when 62.5% of matrix multiplication operations are optimized with LSH, respectively. This is due to enhancing a larger proportion of the underlying operations.

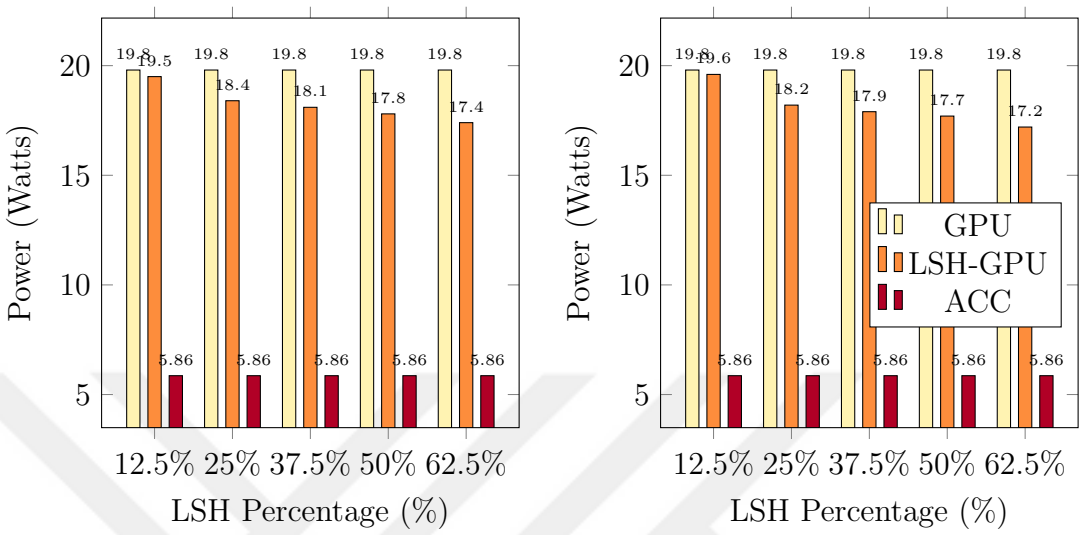

Figure 6.11: Power consumption with different LSH percentage values for Imagenet-1K (left) and CIFAR-100 (right).

Figure 6.11 illustrates how power consumption varies with different levels of LSH optimization on LSH-GPU and ACC when compared with GPU for Imagenet-1K and CIFAR-100. GPU has a consistent power consumption of around 19.8 Watts, while LSH-GPU shows a slight decrease with higher percentage values. In contrast, ACC consistently exhibits lower power consumption, which is 5.86 Watts across all levels of LSH application for both datasets. In LSH-GPU, power decreases with higher percentages because a higher percentage of matrix multiplication operations are optimized with LSH clustering. However, power remains the same in ACC because the same algorithm is used for all LSH clustering operations. Therefore, this does not affect power consumption.

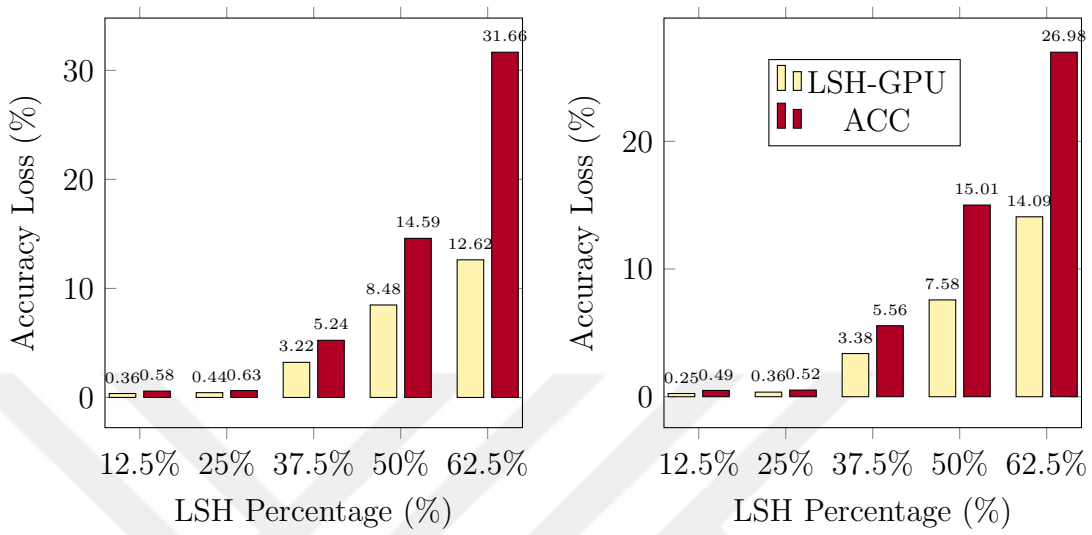

Figure 6.12: Accuracy loss with different LSH percentage values for Imagenet-1K (left) and CIFAR-100 (right).

Figure 6.12 gives the percentage of accuracy loss with different LSH utilization levels. The data presented in the graph demonstrates a modest rise in accuracy loss for LSH-GPU as the percentage of LSH increases. However, as the LSH percentage continues to increase, the accuracy loss experiences a significant increase, reaching up to 12.62% and 14.09%, respectively. In contrast, ACC initially displays a higher accuracy loss of 0.58% and 0.49% at 12.5% for Imagenet-1K and CIFAR-100, respectively. However, the accuracy loss dramatically increases up to 32% in ACC. This comparison suggests that although both platforms exhibit increased accuracy loss with higher LSH utilization, ACC performs worse due to data precision. Also, high accuracy losses occur because optimizing a big part of the Swin Transformer means losing most of the matrices. Due to the unacceptable accuracy levels, by default, we choose to apply LSH clustering on 25% of the linear transformations.

Chapter 7

Conclusion

In edge computing, implementing machine learning models on hardware platforms requires a thorough evaluation of computational efficiency, power usage, and accuracy. This research has investigated these considerations by employing the Swin Transformer model with LSH clustering on our custom hardware accelerator and GPU. By conducting experiments on CIFAR-100 and Imagenet-1K datasets, we have evaluated the influence of different batch sizes, cluster numbers, and LSH percentages on various parameters, including performance, power, and accuracy. The findings from our experiments consistently indicate that our accelerator performs better than GPU implementations. Our results with default parameters show that the hardware accelerator achieves a 1.35x speedup. Furthermore, our accelerator reduces power consumption from 18.2 Watts to 5.86 Watts, a 68% power reduction. This is particularly important for power-constrained edge computing applications. This significant power reduction is obtained with a 0.55% negligible accuracy loss.

In summary, our comparative analysis highlights the potential of hardware accelerators as a robust and effective platform for implementing advanced machine learning algorithms in edge computing. The inherent benefits of accelerators in managing parallel computations and their energy-efficient design make them an

attractive option for future advancements in this area. As machine learning algorithms become more intricate and the need for on-site, real-time processing increases, optimizing these algorithms for edge deployment will become increasingly important. This study enables further research focused on enhancing the implementation of Swin Transformers at the edge, focusing on achieving an optimal balance between computational performance, energy efficiency, and accuracy.

Bibliography

[1] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 10012–10022, 2021.

[2] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.

[3] Y. Shen, M. Ferdman, and P. Milder, "Maximizing cnn accelerator efficiency through resource partitioning," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 2, pp. 535–547, 2017.

[4] E. Cai, D.-C. Juan, D. Stamoulis, and D. Marculescu, "Neuralpower: Predict and deploy energy-efficient convolutional neural networks," in *Asian Conference on Machine Learning*, pp. 622–637, PMLR, 2017.

[5] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[6] K. Cao, Y. Liu, G. Meng, and Q. Sun, "An overview on edge computing research," *IEEE access*, vol. 8, pp. 85714–85728, 2020.

[7] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 869–904, 2020.

[8] J. Mocnej, M. Miškuf, P. Papcun, and I. Zolotová, "Impact of edge computing paradigm on energy consumption in iot," *IFAC-PapersOnLine*, vol. 51, no. 6, pp. 162–167, 2018.

[9] A. A. Süzen, B. Duman, and B. Şen, "Benchmark analysis of jetson tx2, jetson nano and raspberry pi using deep-cnn," in *2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, pp. 1–5, IEEE, 2020.

[10] Y. Kortli, S. Gabsi, L. F. L. Y. Voon, M. Jridi, M. Merzougui, and M. Atri, "Deep embedded hybrid cnn–lstm network for lane detection on nvidia jetson xavier nx," *Knowledge-based systems*, vol. 240, p. 107941, 2022.

[11] Z. Liu, J. Ning, Y. Cao, Y. Wei, Z. Zhang, S. Lin, and H. Hu, "Video swin transformer," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 3202–3211, 2022.

[12] X. Wang, S. Zou, Y. Jiang, L. Zhang, and X. Deng, "Swin-flownet: Flow field oriented optimization aided by a cnn and swin-transformer based model," *Journal of Computational Science*, vol. 72, p. 102121, 2023.

[13] I. Kuon, R. Tessier, J. Rose, *et al.*, "Fpga architecture: Survey and challenges," *Foundations and Trends® in Electronic Design Automation*, vol. 2, no. 2, pp. 135–253, 2008.

[14] D. T. Nguyen, T. N. Nguyen, H. Kim, and H.-J. Lee, "A high-throughput and power-efficient fpga implementation of yolo cnn for object detection," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 8, pp. 1861–1873, 2019.

[15] N. Kitaev, Ł. Kaiser, and A. Levskaya, "Reformer: The efficient transformer," *arXiv preprint arXiv:2001.04451*, 2020.

[16] R. Patel and S. Kumar, "Exploiting computation reuse in convolutional neural networks," *Journal of AI Research*, vol. 77, no. 3, pp. 569–590, 2019.

[17] L. Ning, H. Guan, and X. Shen, "Adaptive deep reuse: Accelerating cnn training on the fly," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pp. 1538–1549, IEEE, 2019.

[18] N. K. Jha and S. Mittal, "Modeling data reuse in deep neural networks by taking data-types into cognizance," *IEEE Transactions on Computers*, vol. 70, no. 9, pp. 1526–1538, 2020.

[19] Y. Zhang and M. Tan, "Adaptive computation reuse in deep neural network training," *Journal of Machine Learning*, vol. 31, no. 4, pp. 776–791, 2020.

[20] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer cnn accelerators," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1–12, IEEE, 2016.

[21] A. Gupta and R. Chandra, "Application of winograd's algorithm for convolutional neural networks on fpgas," in *Proceedings of the Symposium on FPGA for Machine Learning*, pp. 122–130, FPGA Symposium, 2021.

[22] C. Wu and S. Das, "General matrix-matrix multiplication (gemm) optimization on fpgas," *Journal of Parallel Computing*, vol. 64, pp. 17–31, 2018.

[23] A. Srivastava and H. Malik, "Data locality optimization in convolutional networks for edge devices," *Journal of Edge Computing*, vol. 6, no. 2, pp. 88–102, 2019.

[24] T. O'Donnell and M. Singh, "Simcnn: Leveraging computational similarity for accelerated cnn training," *Journal of Accelerated Computing*, vol. 3, no. 1, pp. 34–50, 2021.

[25] C. Zhang, D. Wu, J. Sun, G. Sun, G. Luo, and J. Cong, "Energy-efficient cnn implementation on a deeply pipelined fpga cluster," in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, pp. 326–331, 2016.

[26] J. Yu, Y. Hu, X. Ning, J. Qiu, K. Guo, Y. Wang, and H. Yang, "Instruction driven cross-layer cnn accelerator with winograd transformation on fpga," in

2017 International Conference on Field Programmable Technology (ICFPT), pp. 227–230, IEEE, 2017.

[27] X. Wei, C. H. Yu, P. Zhang, Y. Chen, Y. Wang, H. Hu, Y. Liang, and J. Cong, "Automated systolic array architecture synthesis for high throughput cnn inference on fpgas," in *Proceedings of the 54th Annual Design Automation Conference 2017*, pp. 1–6, 2017.

[28] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *International Conference on Learning Representations*, 2021.

[29] K. Han, Y. Wang, H. Chen, X. Chen, J. Guo, Z. Liu, Y. Tang, A. Xiao, C. Xu, Y. Xu, *et al.*, "A survey on vision transformer," *IEEE transactions on pattern analysis and machine intelligence*, vol. 45, no. 1, pp. 87–110, 2022.

[30] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah, "Transformers in vision: A survey," *ACM computing surveys (CSUR)*, vol. 54, no. 10s, pp. 1–41, 2022.

[31] W. Wang, E. Xie, X. Li, D.-P. Fan, K. Song, D. Liang, T. Lu, P. Luo, and L. Shao, "Pyramid vision transformer: A versatile backbone for dense prediction without convolutions," in *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 568–578, 2021.

[32] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, vol. 30, 2017.

[33] Z. Liu, H. Hu, Y. Lin, Z. Yao, Z. Xie, Y. Wei, J. Ning, Y. Cao, Z. Zhang, L. Dong, *et al.*, "Swin transformer v2: Scaling up capacity and resolution," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 12009–12019, 2022.

[34] N. M. Cicek, L. Ning, O. Ozturk, and X. Shen, "General reuse-centric cnn accelerator," *IEEE Transactions on Computers*, vol. 71, no. 4, pp. 880–891, 2021.

[35] A. Bhaskara and M. Wijewardena, "Distributed clustering via lsh based data partitioning," in *International Conference on Machine Learning*, pp. 570–579, PMLR, 2018.

[36] L. Ning and X. Shen, "Deep reuse: Streamline cnn inference on the fly via coarse-grained computation reuse," in *Proceedings of the ACM International Conference on Supercomputing*, pp. 438–448, 2019.

[37] T. N. Mau and V.-N. Huynh, "An lsh-based k-representatives clustering method for large categorical data," *Neurocomputing*, vol. 463, pp. 29–44, 2021.

[38] NVIDIA Corporation, "Jetson Xavier NX Developer Kit." `https://developer.nvidia.com/embedded/jetson-xavier-nx-devkit`, 2020. Accessed: 2024-02-05.

[39] "Jetpack sdk." Online. Available: `https://developer.nvidia.com/embedded/jetpack`.

[40] Online. Available: `https://developer.nvidia.com/embedded/jetpack`.

[41] "Jetson power gui." Online. Available: `https://docs.nvidia.com/jetson/archives/r35.4.1/DeveloperGuide/`.

[42] Xilinx Inc., "ZCU102 Evaluation Kit." `https://www.xilinx.com/products/boards-and-kits/ek-u1-zcu102-g.html`, 2021. Accessed: 2024-02-05.

[43] "Vivado 2022.1." Online. Available: `https://docs.xilinx.com/r/2022.1-English/ug912-vivado-properties`, journal=AMD Adaptive Computing Documentation Portal.

[44] "Axi bus." Online. Available: `https://developer.arm.com/documentation/ihi0022/latest/`.

[45] "Ila." Online. Available: `https://docs.amd.com/v/u/en-US/pg172-ila`.

[46] "power tool." Online. Available: `https://docs.amd.com/r/2022.1-English/ug906-vivado-design-analysis/Report-Power`.

[47] I. Dataset, "The imagenet large scale visual recognition challenge." Online, 2010. Available: http://www.image-net.org/challenges/LSVRC/.

[48] T. Ridnik, E. Ben-Baruch, A. Noy, and L. Zelnik-Manor, "Imagenet-21k pretraining for the masses," *arXiv preprint arXiv:2104.10972*, 2021.

[49] C.-. Dataset, "The cifar-100 dataset for image classification." Online, 2009. Available: https://www.cs.toronto.edu/~kriz/cifar.html.

[50] S. Singla, S. Singla, and S. Feizi, "Improved deterministic l2 robustness on cifar-10 and cifar-100," *arXiv preprint arXiv:2108.04062*, 2021.

[51] N. Samir, A. S. Hussein, M. Khaled, A. N. El-Zeiny, M. Osama, H. Yassin, A. Abdelbaky, O. Mahmoud, A. Shawky, and H. Mostafa, "Asic and fpga comparative study for iot lightweight hardware security algorithms," *Journal of Circuits, Systems and Computers*, vol. 28, no. 12, p. 1930009, 2019.

[52] J. Cong, Z. Fang, M. Lo, H. Wang, J. Xu, and S. Zhang, "Understanding performance differences of fpgas and gpus," in *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 93–96, IEEE, 2018.

[53] K. O'Neal and P. Brisk, "Predictive modeling for cpu, gpu, and fpga performance and power consumption: A survey," in *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 763–768, IEEE, 2018.