西北工業大學

Undergraduate Graduation Design (Thesis)

(2023)

Title: Energy Efficiency Analysis and Optimization of A Typical Deep Learning Application

School:	School of Computer Science
Major:	Computer Science and Technology
Name:	Abid Ali
Student Id:	2019380141
Supervisor:	Zhengxiong Hou
Graduation Time:	06/19/2023

Energy Efficiency Analysis and Optimization of A Typical Deep Learning Application

A Thesis Submitted

to Faculty of Computer Science and Technology

of Northwestern Polytechnical University

in Partial Fulfillment of the Requirements

for the Degree of Bachelor of Science

by

Abid Ali

Advisor: Prof Zhengxiong Hou

摘要

深度学习应用往往需要使用 GPU 资源，已成为高性能计算数据中心的主要应用。在大型数据中心中，这些主要应用的节能降耗非常重要，有利于降低运营成本和碳排放。如今，人们不仅关注应用程序的性能，还关注应用程序的能源效率。深度学习应用的性能、功耗、能耗与组件功耗上限、CPU 核心总数、CPU 频率等有关。本文拟针对典型的应用，对其能效进行分析，研究优化方法。深度学习应用，即卷积神经网络（CNN）。本论文的主要工作如下：

（1） 基于 DVFS 的 CNN 在 CPU 上的能效分析

深度学习应用程序主要依靠 GPU 进行训练。然而，它也需要 CPU 和 GPU 的合作，这是主要的能源消耗来源。CPU 频率将对深度学习应用程序的性能和能耗产生一些影响。DVFS 是动态调整 CPU 频率的主要方法。因此，我们在 CPU 上使用 DVFS 对 CNN 的性能和能效进行了分析研究。对性能、节能/保守、按需和用户空间等策略进行了定量评估和分析。

（2） GPU 和 CPU 功率封顶的能效分析

功率封顶技术也是功率控制和优化能效的一种很好的方法，可以应用于单个节点上的 CPU、存储器和 GPU，然后应用于多个节点。我们分析了功率封顶技术对 CNN 性能和能耗的影响。我们主要对 GPU 和 CPU 的功率封顶进行了定量评估和分析，这可以为能效优化提供基础。

（3） 利用模型优化实现能源效率

模型优化技术可以有效地应用于卷积神经网络以提高能量效率。量化和模型修剪与微调方法，用于优化细胞神经网络并提高其能源效率。

关键词：深度学习，卷积神经网络（CNN）, 性能，DVFS，Power Capping，能效，GPU

Abstract

Deep learning applications often require the use of GPU resources and have become a major application in high-performance computing data centers. In large data centers, energy saving and consumption reduction for these major applications are very important, which is conducive to reducing operating costs and carbon emissions. Today, people are not only concerned about the performance of the application, but also the energy efficiency of the application. Performance, power consumption, and energy consumption of deep learning applications are related to the upper limit of component power, the total number of CPU cores, and CPU frequency, etc. This thesis intends to analyze its energy efficiency and study optimization methods for a typical deep learning application, i.e. Convolution Neural Network (CNN). The main work of this thesis is as follows:

(1) Energy efficiency analysis of CNN with DVFS on CPU

Deep learning applications mainly rely on GPU for training. However, it also needs cooperation of CPU and GPU which are main sources of energy consumption. CPU Frequency will have some impacts on performance and energy consumption of deep learning applications. DVFS is the main approach for dynamically adjusting CPU frequencies. Thus, we conducted research on analysis of performance and energy efficiency of CNN using DVFS on CPU. Several strategies such as performance, powersave/conservative, on-demand and userspace were quantitatively evaluated and analyzed.

(2) Energy efficiency analysis with power capping on GPU and CPU

Power capping technology is also a good approach for power controlling and optimizing energy efficiency, which can be applied to CPU, memory, and GPU on a single node after that in multiple nodes. We analyzed the impact of power capping technology on performance and

energy consumption of CNN. We mainly quantitatively evaluated and analyzed power capping on GPU and CPU, which can provide a foundation for energy efficiency optimization.

(3) Energy Efficiency by using Model optimization

Model optimization techniques can be effectively applied to convolutional neural networks (CNNs) to enhance energy efficiency. Quantization and Model Pruning with Fine-tuning approaches for optimizing CNNs and improving their energy efficiency.

KEY WORDS: deep learning, convolution neural network (CNN), performance, DVFS, power capping, energy efficiency, GPU

Table of Contents

Chapter One : Introduction

1.1 Background

Computer technology has a transformative impact on various aspects of human life and significantly contributed to scientific research, industrial development, education, and national defense and military. Here are some key contributions of computer technology in these fields:

1. Scientific Research: Computers have revolutionized scientific research by enabling complex simulations, data analysis, and modeling. High-performance computing (HPC) systems have enabled researchers to tackle complex problems, perform large-scale simulations, and process massive amounts of data.

2. Industrial Development: Computers are playing a crucial role in industrial development and automation. It have enhanced manufacturing processes, enabled precision control systems, and increased productivity. Computer-aided design (CAD) and computer-aided manufacturing (CAM) have revolutionized product design and production.

3. Education and Teaching: Computers have transformed education and teaching methodologies.It facilitated e-learning, online courses, and access to vast educational resources. Computers have enabled interactive and multimedia-based learning, making education more engaging and accessible. Educational software and simulations have enhanced the learning experience and provided personalized learning opportunities.

4. National Defense and Military: Computers are vital in national defense and military operations. It support communication systems, command and control centers, and intelligence analysis. Computers enable sophisticated weapon systems, radar systems,

and surveillance systems. Cybersecurity and encryption technologies are critical in protecting sensitive information and securing military networks.

It's important to note that the contributions of computer technology continue to evolve rapidly, and new advancements such as artificial intelligence, big data analytics, and the Internet of Things (IoT) are opening up new possibilities and applications across these fields.

1.2 Significance of the Topic

Recently, other than scientific and engineering computing, deep learning applications have been one type of main applications in HPC and data centers, which usually need GPU computing resources.Energy consumption reduction of the main applications is important in HPC and data centers.Existing research works mainly focusing on energy efficiency of scientific applications [1]. In this thesis, we are going to analyze and optimize energy efficiency of deep learning applications. This research topic is significant in the theory and in practice.

1.3 Research Status

1.3.1 Linux DVFS Technology:

Linux operating systems has the following governors [2], [3]: ① **performance** directly adjust the CPU frequency to the maximum so that we get highest performance and highest power consumption.② **powersave** directly adjust the CPU frequency to the lowest so that we get the lowest performance and lowest power consumption. ③ **userspace** adjust the CPU frequency to the frequency manually set by the user.④ **ondemand** according to CPU the load dynamically adjusts the frequency. When the load is higher than a certain threshold, the frequency is adjusted to the maximum, and when it islower than a certain threshold, the frequency is reduced. The frequency is adjusted to the appropriate frequency to reduce power consumption while meeting a certain performance. ⑤ **conservative** depends on the CPU load

dynamically adjust the frequency. Compared with ondemand, the frequency adjustment range is smallerand the strategy is more conservative but more power saving

1.3.2 Power Capping Technology:

We see that,powercap settings to help compute nodes perform fine-grained powercap allocation. Paper [4], [5] proposed a fine-grained differential approach (FGD) is used to quantitatively analyze how inappropriate power capping degrades latency-sensitive applications,application's performance. By using FGD, we can set fine-grainedaccurate power budget, thereby minimizing provisioned power per server.

1.3.3 Optimizing Energy Efficiency of Deep Learning Applications in a GPU Cluster Environment

In a GPU cluster, deep learning applications usually need some computing nodes for distributed training. Performance and energy consumption of deep learning applications will be impacted by many factors, such as number of computing nodes, total number of CPU cores and GPU cards, CPU frequency, Power Capping of GPU and CPU, as well as applicationfeatures. We will conduct research on how to optimize energy efficiency of deep learning applications based on the main factors.

We are going to select a typical deep learning model CNN (Convolutional Neural Networks) [6], [7] and AI converged application DeepMD [8] for experiments on a GPU cluster. All the codes are executed based on the Anaconda framework [9] that is the virtual environment with various python packages.Processing rate (such as image/sec.) [7] and running time will be adopted as a performance metric for CNN applications and DeepMD application respectively.Watt, joules, and Performance/Power and EDP will be used as metrics for evaluating power, energy consumption and energy efficiency.

1.3.4 Energy Efficiency by using Model Optimization

As power shortages and cost rise, energy efficiency has become more crucial in high-performance computing data centers Workload and system characteristics form a complex optimization search space in which optimal settings for energy efficiency and performance often diverge. Therefore, in order to achieve the ideal balance between both, by identify trade-off choices for performance and energy efficiency. In the paper [1] proposed, an innovative statistical model that accurately predicts the Pareto optimal performance and energy efficiency trade-off options using only user-controllable parameters.

I have proposed model optimization which simplify a complex model to a lightweight model with the existing work of the paper [10] that addresses the issues .

1.4 Thesis Organization

The organizational structure of this thesis mainly consists of seven parts.

- The first part is the introduction part, which introduces the background of the selected topic of this thesis, the significance of the selected topic, the research status of this direction and the research content of this thesis.
- The second part introduces the experimental environment of this graduation project.
- The third part mainly describes about the CNN model and MNIST dataset
- The fourth part mainly describes Energy efficiency analysis of CNN with DVFS on CPU.
- The fifth part mainly describes the Power Capping technology and the Deep Learning applications test using Power Capping technology and CPU frequency adjustment strategy to find out the best CPU frequency adjustment strategy.
- The sixth part describes how Optimize Energy Efficiency by Selecting the Optimal Parameters, DVFS policies, Power Capping Values and Model optimization(Light-weight CNN model)

Chapter Two : Typical Deep Learning Applications and Experimental Environment

2.1 Typical Deep Learning Applications

2.1.1 Deep Learning Applications

The deep learning (DL) computer paradigm has recently recognized by the machine learning (ML) community as the Gold Standard. Additionally, it has steadily grown to be the most popular computational strategy in the area of ML, producing exceptional outcomes on a number of challenging cognitive tasks that are on par with or even above human performance. The capacity to learn from enormous volumes of data is one advantage of DL. The DL field has grown fast in the last few years and it has been extensively used to successfully address a wide range of traditional applications.

More importantly, DL has outperformed well-known traditional ML techniques in many domains, e.g., cybersecurity, natural language processing, bioinformatics, robotics and control, and medical information processing, among many others.

2.1.2 CNN Based Model

In recent times, machine learning (ML) has witnessed widespread adoption in research and has been integrated into various applications, such as text mining, spam detection, video recommendation, image classification, and multimedia concept retrieval [11, 12]. Deep learning (DL), which is often referred to as representation learning (RL), has emerged as a popular choice among the different ML algorithms used in these applications [13]. The field of deep and distributed learning continues to witness new studies, primarily driven by the exponential growth in data availability and remarkable advancements in hardware technologies like High-Performance Computing (HPC) [14]. These factors have contributed to the increasing interest and progress in DL, enabling researchers to tackle complex problems and achieve remarkable results in various domains.

2.1.3 Performance Characteristics of CNN

Typical convolutional neural networks (CNNs) are usually computationally intensive, making computation one of the primary bottlenecks in single GPU training. Efficiently using modern GPUs (or other hardware accelerators) requires training with large mini-batches, which could be limited due to modern GPU main memory capacity. Training DNNs in a distributed environment with multiple GPUs and machines, brings with it yet another group of potential bottlenecks, network and interconnect bandwidths. Even for a specific model, setting up implementation and hardware for pinpointing whether performance is bounded by computation, memory, or communication is not easy due to the limitations of existing profiling tools.

2.1.4 Framework Selection

There are many open-source DNN frameworks, such as TensorFlow [15], Theano [16], MXNet [17], CNTK [18],Caffe [19], Chainer [20], Torch [21], Keras [22], PyTorch [23]. Since no single framework has emerged as the dominant leader in the field and since different framework-specific design choices and optimizations might lead to different results,

2.1.5 Training Differs Significantly from Inference

The algorithmic differences between training and inference lead to many differences in requirements for the underlying systems and hardware architecture. First, backward pass and weight updates, operations unique to training, need to save/stash a large number of intermediate results in GPU memory, e.g., outputs of the inner layers called feature maps or activations . This puts significant pressure on the memory subsystem of modern DNN accelerators (usually GPUs) - in some cases the model might need tens of gigabytes of main memory . In contrast,the

memory footprint of inference is significantly smaller, in the order of tens of megabytes , and the major memory consumers are model weights rather than feature maps. Secondly, training usually proceeds in waves of mini-batches, a set of inputs grouped and processed in parallel . Mini-batching helps in avoiding both overfitting and under utilization of GPU's compute parallelism.

Thus, throughput is the primary performance metric of concern in training.Compared to training, inference is computationally less taxing and is latency sensitive.

2.2 Summary

The deep learning techniques previously have outperformed conventional methods, but they still have some disadvantages. An input layer, an output layer, and several hidden layers make up a CNN. Convolutional, pooling, and fully linked layers are frequently seen in a CNN's hidden layers.Convolutional layers apply a convolution operation to the input, followed by a non-linear activation function. and require bulky networks to achieve accuracy.

In CNNs, convolution takes the majority of the energy. Specifically, convolution with multiple layers consumes more than 90% of the total energy resources [24].

To solve the above problems, I have chosen to introduce a light-weight attention-fused CNN for Energy Efficiency and Optimization.

2.2 Experimental Environment of Thesis

2.2.1 System Software and Hardware Configuration for One GPU Node

1）The experimental hardware environment and software environment of a single node are shown in Table 2-1 and Table 2-2 respectively:

Table 2-1

Single-node Hardware Environment

Hardware Name	Hardware Configuration
CPU	(Intel(R) Xeon(R) Silver 4114 CPU @ 2.20GHz, 20core) * 2 "Hyperthreading" and "Turbo boost" disabled, power capped at 85Watt
GPU	NVIDIA A100 40GB PCIE
DRAM	Power Cap 21 Watt

Table 2-2

Single-node Software Environment

Name of Software	Describe	Version
OS	GNU/Linux	CentOS Linux release 7.9.2009
Python	Script Language	Python -3.7
CUDA		cuda-11.3

2.3 Experimental Dataset

We used MNIST dataset for the experiment.The MNIST database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems. The database is also widely used for training and testing in the field of machine learning.

It was created by "re-mixing" the samples from NIST's original datasets. The creators felt that since NIST's training dataset was taken from American Census Bureau employees, while the testing dataset was taken from American high school students, it was not well-suited for machine learning experiments. Furthermore, the black and white images from NIST were normalized to fit into a 28x28 pixel bounding box and anti-aliased, which introduced grayscale levels.

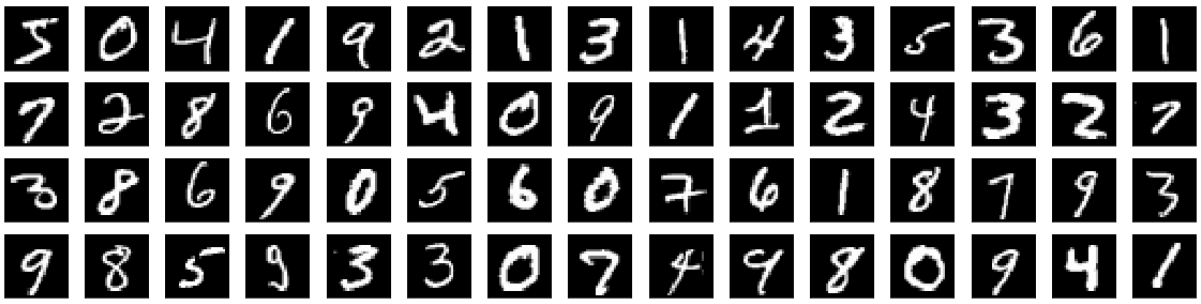
Chapter Three : Simple Convolutional Neural Network Models For Mnist Digit Recognition

3.1 Introduction

Data set for the MNIST handwritten digit recognition (Figure 1) is one of the most basic data sets used to test performance of neural network models and learning techniques Using 60,000 images as the training set, learning techniques including k-nearest neighbors (KNN), random forests, support vector machines (SVM), and basic neural network models made it straightforward to attain a 97%–98% accuracy on the test set of 10,000 images. Convolutional neural networks (CNN) improve this accuracy to over 99% with less than 100 misclassified images in the test set.

Figure 1

Images from The MNIST Training Set.



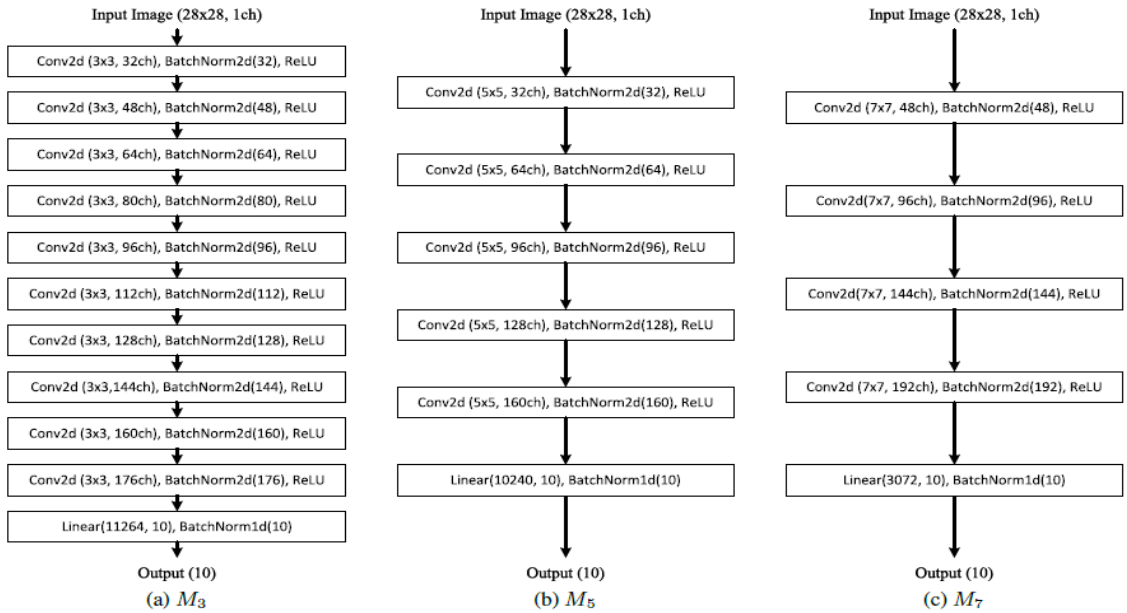
We require more complicated models, careful adjustment of hyperparameters using learning rate and batch size, regularization techniques such as batch normalization and dropout, and augmentation of training data to enhance accuracy after 99%. According to the publications, the maximum accuracy on the MNIST test set was between 99.7% and 99.84% [25] [26] [27].

3.2 Network Design and Training

At the end of a model's several convolutional layers lies a fully linked layer. Every convolution layer carries out a 2D convolution, 2D batch normalization, and ReLU activation. After convolution, maximum or average pooling are not applied. Instead, the size of feature map is reduced after each convolution because padding is not used. For example, we used a 3 × 3 kernel, the width and height of the image is reduced by two after each convolution layer. Similar approach is taken in other networks [28] feature map size becomes small enough, a fully-connected layer connects the feature map to the final output. A 1D batch normalization is used at the fully-connected layer, while dropout is not used.

Figure 2

Network Models used for MNIST Digit Classification



We use three different networks and combine the results from these networks. The networks differ only in the kernel sizes of the convolution layers: 3×3, 5×5, and 7×7. Because different kernel size lead to different size reduction in feature maps, the number of layers is different for each network. The first network, M3, uses 10 convolution layers with 16(i + 1)

channels in ith convolution layer. The feature map becomes 8×8 with 176 channels after the 10th layer. The second network, M5, uses 5 convolution layers with 32i channels in ith convolution layer. The feature map becomes 8×8 with 160 channels after the 5th layer. The third network, M7, uses 4 convolution layers with 48i channels in the convolution layer. The feature map becomes 4×4 with 192 channels after the 4th layer.

3.3 Impact of Data Augmentation

Data augmentation is a method for broadening the range of training data without actually gathering and categorizing new data. A huge data set is necessary for this crucial supervised learning approach in order for the network model to function well. When training the proposed network, we used two schemes for data generation: random rotation and random translation. There are many other schemes such as cropping, flipping, and resizing, and the best augmentation schemes depend on the data.We compared performance of four M5 networks with different combinations of augmentation schemes applied.It can be observed that data augmentation is helpful in general. For the MNIST data set,applying random rotation has slightly higher contribution than random translation, but both schemes are needed to achieve the best accuracy.

Chapter Four : Energy Efficiency Analysis of CNN with DVFS on CPU

4.1 DVFS Technology

DVFS（Dynamic Voltage and Frequency Scaling）Dynamic voltage frequency adjustment is a real-time voltage and frequency adjustment technology. Its function is to evaluate the current actual load and power demand of the components, and then dynamically set the clock frequency and working voltage of the components, so that the power provided by the system can meet the requirements of certain components. In the case of performance requirements, the component power consumption will not be too high.

The principle of the influence of adjusting voltage frequency on chip power consumption is that power consumption in CMOS circuits can be mainly divided into dynamic power consumption and static power consumption. The formula is as follows:

$$\text{power} = \sum\left(CV^2\alpha f + VI_{dq}\right) \quad (4\text{-}1)$$

Table 4-1

Meanings of Letters in Formula (4-1)

C	The capacitance value of the load capacitor
V	Operating Voltage
α	Toggle rate at current frequency
f	working frequency
I_{dq}	Quiescent current in the circuit

The first part of Equation 4-1 represents the dynamic power consumption, and the second part represents the static power consumption. It can be seen from the formula that if you wan t to reduce the dynamic power consumption, you can start with C, V, α, and f. For software, the commonly used adjustment methods only involve two factors, V and f.

The operation logic of DVFS technology is mainly divided into four steps:

- Firstly, we collect signals related to system load and calculate the current system load
- Secondly, according to the current load of the system, predict the performance required by the system in the next time period;
- Then convert the predicted performance into The required frequency is used to adjust the clock setting of the chip;
- Finally, the corresponding voltage is calculated according to the new frequency. Notif y the power management module to adjust the voltage to the CPU.

4.2 Linux Dynamic Frequency Modulation Method

4.2.1 Cpufreq Subsystem

At present, the dynamic frequency modulation technology already exists in the Linux system. There is a cpufreq subsystem in the Linux system. cpufreq implements cpufreq driver and cpufreq governor by registering with the system. The governor implements the strategy of frequency modulation, and the driver implements the actual operation of frequency modulation, thus completing the dynamic adjustment of frequency and voltage. The Linux CPU frequency modulation module is mainly divided into three parts [29]: CPUFreq core module, CPUFreq driver module and CPUFreq governor module.

4.2.2 Principle of FM Method

Frequency modulation methods commonly used in Linux operating system [29]：

- performance：Directly adjust the CPU frequency to the maximum, the highest performance, the highest power consumption.

- powersave：Directly adjust the CPU frequency to the lowest, the lowest perfor mance, the lowest power consumption.

- userspace：Adjust the CPU frequency to the frequency specified by the user.
- ondemand：Dynamically adjust the frequency according to the CPU load. When the load is higher than a certain threshold, the frequency will be adjusted to the maximum. When it is lower than a certain threshold, the frequency will be reduced to an appropriate frequency to reduce power consumption while meetin g certain performance.
- conservative：Dynamically adjust the frequency according to the CPU load. Compared with ondemand, the frequency adjustment range is smaller, the strategy is more conservative, but it saves more power.

The more complex ondemand principle is: when the load is higher than the preset thresho ld (od_tuners->up_threshold, the default value is 95%), immediately select the maximum ope rating frequency of the policy as the next operating frequency. If the load does not reach the preset threshold, but the current frequency is already the lowest frequency, then do nothing and return directly. When the system load decreases, the previously set high frequency is not needed, so when the load is lower than another preset value, it is necessary to calculate a new frequency suitable for the load to reduce power consumption.

For conservative, when the load is greater than up_threshold (80%), a certain frequency is increased on the current basis (5% of the maximum frequency), and when the load is less than down_threshold (20%), a certain frequency is reduced on the current basis (5% of the maximum frequency), so as to achieve dynamic frequency tuning.

On the whole, conservative is also dynamically adjusted like ondemand, but more conser vative than ondemand. It only increases or decreases by 5% each time, which will save more power.

4.3 Testing Frequency Modulation Methods for the Performance and Energy Efficiency of CNN

4.3.1 Experiment Description and Configuration

In order to better understand and analyze the characteristics, advantages and disadvantage s of the five frequency modulation methods. We use the CNN model program to test the five f requency modulation methods. And use some tools to monitor the entire operation process of CNN model, obtain important data, and perform quantitative analysis on the results. The main configuration of this experiment is as follows：

1）The experimental environment configuration is: in a single-node environment, use 20 CPU cores, the upper limit of the power of the GPU is 250Watt and use 1 GPU core, the upper limit of the power of the CPU is set to the default value, the upper limit of the power of the DRAM is also set to the default value, and the CNN model program runs at 0-19 CPU cores, each process uses 10 cores.

2）In terms of data acquisition, the Deep Learning applications is run by writing shell scripts, and monitoring tools are added to the shell scripts to obtain various data during the running process. Among the monitoring tools are：

- likwid tool：likwid is a collection of tools for index analysis of multi-threaded progr ams, which is convenient for programmers to perform performance tuning. It can go deep to control the frequency of hardware such as CPU. This experiment uses likwid -powermeter to sample the power of CPU and DRAM.
- top tool：The top command can obtain the CPU load in the current system. This experiment uses top to sample the CPU load in real time.
- nvidia-smi：nvidia-smi is the system management interface of nvidia, where smi is

the abbreviation of system management interface, it can collect various levels of information of the graphics card, check the usage of video memory, etc.

This experiment uses nvidia-smi to sample the GPU power in real time.

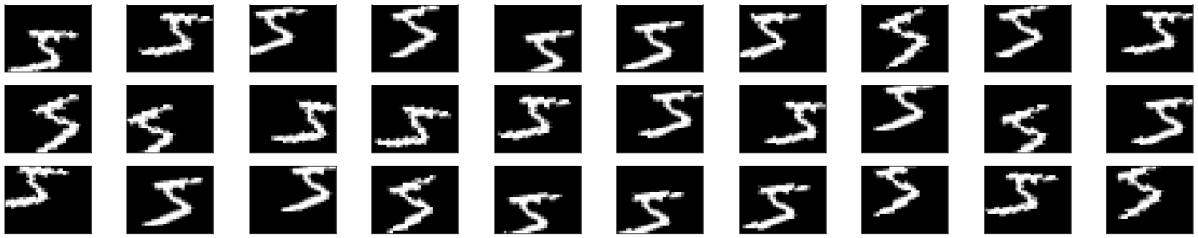
3）Data visualization：Because the data collected in the experiment is a text file, it is difficult to simply analyze it, so python is used to write a program to visualize the data.

4.3.2 Result Quantification and Analysis

During training, we apply transformation on data that consist of random translation and random rotation. For random translation, an image is randomly shifted horizontally and vertically, up to 20% of the image size in each direction. For random rotation, the image is rotated up to 20 degrees in either clockwise or counterclockwise direction. The amount of transformation varies for each image and each epoch, so the network gets to see various versions of an image in the training set (Figure 3).

Figure 3

Random Translation and Random Rotation Applied to a Training Image.



The network parameters are initialized using default initialization methods in PyTorch [30]. For parameter optimization, we use the Adam optimizer with cross-entropy loss function. Learning rate starts at 0.001, and exponentially decays with decaying factor = 0.98. The batch size is 120, and so 500 parameter updates occur in an epoch. We use exponential moving

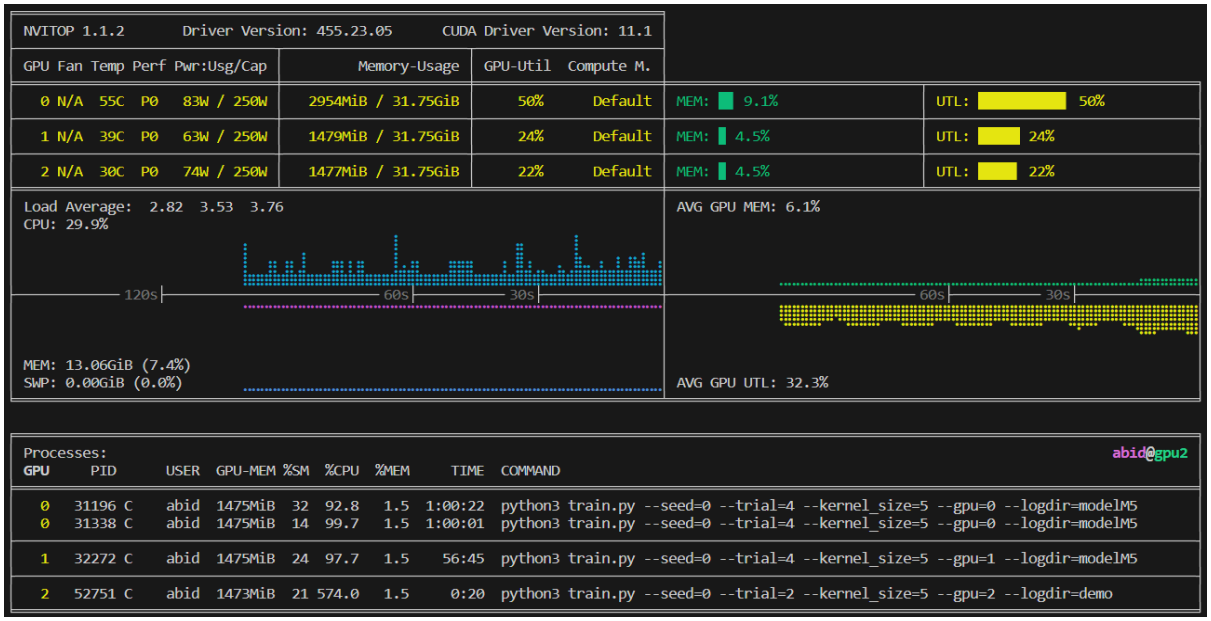
average of weights for evaluation, which may lead to better generalization [31]. The exponential decay used for computing the moving average is 0.999.

We used 5 modes performance, conservative, ondemand, powersave, and userspace (800KHz) The model become more complex modelM3, modelM5 and modelM7 respectively due to increase of convulational layer..

1) The CPU frequency fluctuation of conservative is changing dynamically, especially when the frequency is close to the highest frequency when the two obvious peaks in the figure. The CPU frequency of ondemand also changes dynamically, and it is also close to the highest frequency at the two peaks, but the CPU frequency fluctuation of ondemand is steeper than that of conservative, which is in line with our expectation that "conservative frequency modulation is more conservative and stable than ondemand". The CPU frequency of powersave is at the lowest frequency, and the CPU frequency of userspace (800KHz) has always been the frequency we set. Overall, the five frequency modulation strategies are in line with our expectations based on the principles of frequency modulation strategies.

Figure 4

nvitop-tool:Use for GPU Analysis



2）It can be observed from Figure, that there are mainly two significant peak stages in the entire running process, and the GPU power of these two peaks is above the average power of the GPU. We can speculate that although different frequency modulation strategies are use d to run the CNN program each time, it can be seen from the CPU, GPU power and CPU load that the main running time of the CNN program appears at two peaks. In order to confirm our guess. Use the nvidia-smi or nvitop tool to monitor the running process. Taking the running process of an CNN program based on the performacne strategy as an example.The program tried to use average GPU utilization 32.4 % and I noticed CNN prefer to use CPU more than GPU.

3) The CNN model operation table of performance, conservative, ondemand, powersave, and userspace (800KHz) respectively and GPU was set to 250 Watt (Default) where we used batch_size=120 for training

Finally, important data such as the running time of the CNN program, the power of the CPU, and the performance results of the CNN program are summarized for quantitative comparison and analysis. From the summary results, we can find the following characteristics (the power and energy consumption referred to here refer specifically to the power & energy consumption of the CPU)：

The CNN model operation table of performance, conservative, ondemand, powersave, and userspace (800KHz) respectively and GPU was set to 250 Watt and cpulimit -l 10 where we used batch size=120 for training

Table 4-2

GPU was set to 250 Watt and cpulimit -l 10

		Original	90% training & 10% inference	80% training & 20% inference	70% training & 30% inference
Governor	Model	Runtime (seconds)			
performance	modelM7	8870.50	8761.50	7750.50	7180.50
	modelM5	9005.34	8750.78	8096.86	7563.86
	modelM3	9100.50	8820.80	8378.50	7720.70
conservative	modelM7	18030.50	17531.10	15943.80	15008.90
	modelM5	20050.20	18361.40	16109.90	15573.90
	modelM3	20710.50	18734.00	16421.20	15945.20
ondemand	modelM7	18380.20	16421.74	14982.50	14187.70
	modelM5	18910.50	16743.92	15017.30	14360.60
	modelM3	19131.00	17113.08	15245.80	14677.30
powersave	modelM7	20820.20	18970.40	18850.40	18107.40
	modelM5	21524.60	19428.40	19228.40	18458.40
	modelM3	21500.60	20175.60	20045.60	19092.60
userspace	modelM7	18010.20	17975.00	18865.00	16858.50
	modelM5	18422.90	18136.10	18285.10	17160.10
	modelM3	18914.50	1883.50	17933.00	17588.90

Chapter Five : Research on Power Capping Technology

5.1 Introduction to Power Capping Technology Technology and Implementation

5.1.1 Result Quantification and Analysis

Power capping technology is a mainstream method for optimizing application energy efficiency. Can be applied to CPU, DRAM and GPU. Its function is to modify the power cap of system components (such as CPU, GPU, etc.) to an appropriate value, because in some cases, some components of the system themselves do not use so much power, if the power cap is maintained at the nameplate The highest power displayed, then tends to cause excessive power dissipation. When using power capping technology, it may reduce the performance of CNN program. We hope that in the case of reducing power consumption and energy consumption, the performance will not be reduced too much, so that power consumption-performance and energy consumption-performance “All are well balanced”. This chapter studies the impact of Power Capping technology on the performance, energy consumption, and power consumption of high-performance computing applications in a single-node environment, conducts quantitative evaluation and analysis, and provides a basis for energy efficiency optimization.

5.1.1 Power Capping Technology implementation method

In this experiment, we mainly conduct Power Capping experiments on the core compone nts of the system: CPU, DRAM, and GPU, quantify the results and analyze various results, so as to provide a basis for subsequent system energy efficiency optimization.

The power consumption capping of CPU and DRAM is mainly realized by Intel's RAPL technology. RAPL (running average power limit) is a set of interfaces for power consumption monitoring and control provided by Intel, which can limit the average operating power of CPU and DRAM within a certain period of time. Its principle is to combine DVFS technology

and clock throttling technology to keep the power consumption of components within the powerlimit set by users [16]. RAPL provides several major parts of the power domain:

- Package：Provides energy consumption measurements for the entire CPU socket. It includes power consumption of all cores, integrated graphics and non-core components (and finally caches, memory controllers)

- Power Plane 0：Provides the power consumption of all processor cores on a single socket;
- Power Plane 1：Provides energy consumption measurement of the GPU on the socket;
- DRAM：Provides energy consumption measurements for RAM connected to the integrated memory controller.On the server node of this experiment, the Linux system has mounted the intel-rapl module, which is located under /sys/devices/virtual/powercap. The main modules include intel-rapl:0 and intel-rapl:1, which respectively represent the two physical CPUs on this server node. Each intel-rapl:n has coconstraint_X_power_limit_uw (X = 0, 1, in uw), which represent the CPU's long-term window power consumption limit and short-term window power consumption limit [16]. And under each intel-rapl:n there is also intel-rapl:n:0 representing the RAPL module of DRAM.

Therefore, the constraint_X_power_limit_uw under the intel-rapl module of the CPU and DRAM can be modified to cap the power upper limit of the CPU and DRAM.

In this experiment, the GPU is capped using the nvidia-smi tool provided by Nvidia. nvid ia-smi (also known as NVSMI) provides monitoring and management functions. The specific

commands are "sudo nvidia-smi -pm 1" and "sudo nvidia-smi -pl x" (x is the power limit, in Watt).

5.2 Power Capping Experiment

5.2.1 CPU Power Capping

In this test, other system configurations remain unchanged, and the method of modifying the CPU power upper limit is used to analyze the impact of CPU Power Capping on the energy efficiency of CNN program.

In this experiment, the CNN program was used for testing. The number of CPU cores is 20, the CPU frequency adjustment policy is conservative the CPU power cap is 60, the upper limit of GPU power is 250Watt, and the upper limit of DRAM power is 12Watt.

The results of this experiment are shown in Table 5-1:

Table 5-1

Conservative with CPU Power Cap is 60

		Original	90% training & 10% inference	80% training & 20% inference	70% training & 30% inference
Governor	Model	Runtime (seconds)			
conservative	modelM7	19131.00	18031.10	16243.80	15108.90
	modelM5	21483.20	19121.40	16409.90	15373.90
	modelM3	21718.50	19914.00	16721.20	16225.20

Figure 5

Conservative with CPU power cap is 60

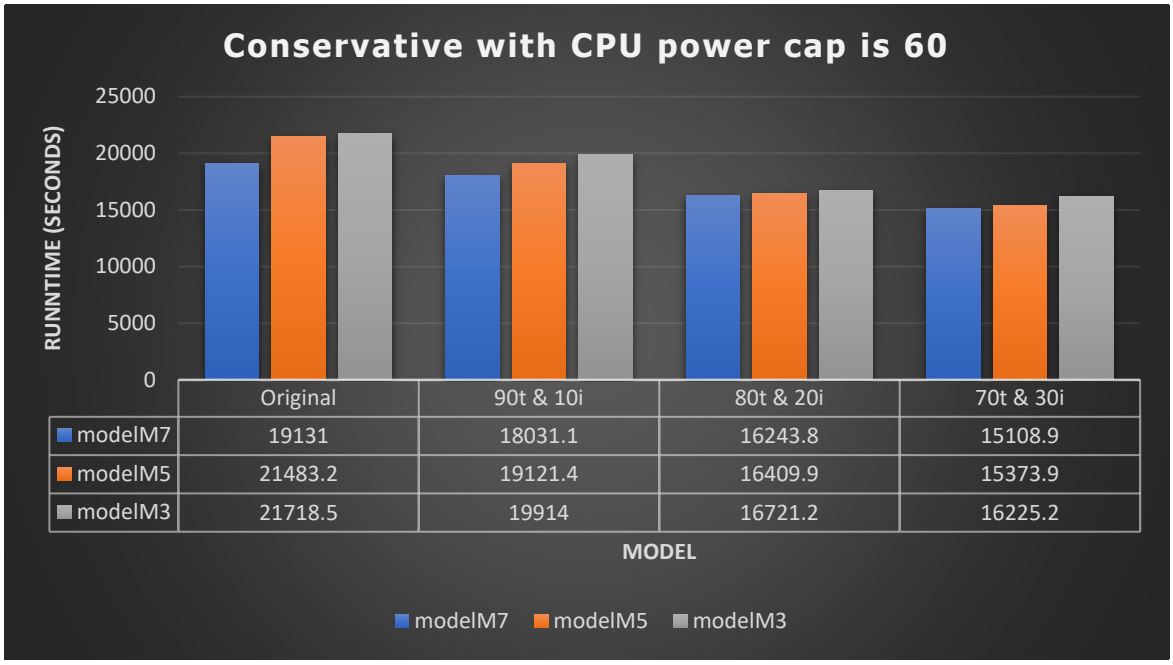


Table 5-2

Conservative Energy and Power consumption

Governor	Model				Original	90% training & 10% inference	80% training & 20% inference	70% training & 30% inference
conservative	modelM3	socket 0 on CPU 0	Domain PKG	Energy consumed (Joules)	85499.0	79525.8	75867.0	69516.5
				Power consumed (Watt)	6.3373	5.62266	4.3457	3.2775
			Domain DRAM	Energy consumed (Joules)	55751.8	47270.2	42802.2	40830.57
				Power consumed (Watt)	3.51973	2.62158	1.61217	1.192528
		socket 1 on CPU 14	Domain PKG	Energy consumed (Joules)	196949	182005	33586.8	20391.4

				Power consumed (Watt)	10.961393	10.0939	2.37467	2.17467
			Domain DRAM	Energy consumed (Joules)	60104.8	57397	31235.5	13924.8
				Power consumed (Watt)	4.656513	3.18321	2.20843	1.02877
	modelM5	socket 0 on CPU 0	Domain PKG	Energy consumed (Joules)	78760.5	61500.2	132095	124634
				Power consumed (Watt)	3.67103	3.11214	8.57212	9.0825
			Domain DRAM	Energy consumed	5620.22	57939.4	50648.6	57638.7
				Power consumed (Watt)	1.261959	2.93195	1.98889	6.51266
		socket 1 on CPU 14	Domain PKG	Energy consumed (Joules)	96082.4	84203.0	81436.9	78918.7
				Power consumed (Watt)	7.3479	6.2476	5.28472	5.8564
			Domain DRAM	Energy consumed (Watt)	42151.1	3494.54	39919.7	25594.4
				Power consumed (Watt)	3.02251	2.176837	2.59053	1.176837
	modelM7	socket 0 on CPU 0	Domain PKG	Energy consumed (Joules)	74496.2	24134.3	12660.1	10898.2
				Power consumed (Watt)	9.49044	8.6862	8.26313	7.8184
			Domain DRAM	Energy consumed (Joules)	54726.7	44805.3	30048.1	4909.79
				Power consumed (Watt)	6.51338	2.54085	1.96121	0.230043
		socket 1 on CPU 14	Domain PKG	Energy consumed	76417.9	63228.7	24944.8	16637.6
				Power consumed (Watt)	4.98772	7.52525	8.16876	9.43495
			Domain DRAM	Energy consumed	58956.6	54682.5	39339.3	14824.9
				Power consumed (Watt)	7.0168	3.10097	2.56764	1.694607

In this experiment, the CPU frequency adjustment policy is conservative the CPU power cap is 50, the upper limit of GPU power and DRAM power are 250 Watt and 12Watt.

The results of this experiment are shown in Table 5-3:

Table 5-3

Conservative with Powercap is 50

		Original	90% training & 10% inference	80% training & 20% inference	70% training & 30% inference
Governor	Model	Runtime (seconds)			
conservative	modelM7	18451.00	17956.10	15843.90	15050.80
	modelM5	21133.20	18911.55	16329.90	15123.90
	modelM3	21658.50	19854.00	16521.20	16065.20

Figure 6

Conservative with Powercap is 50

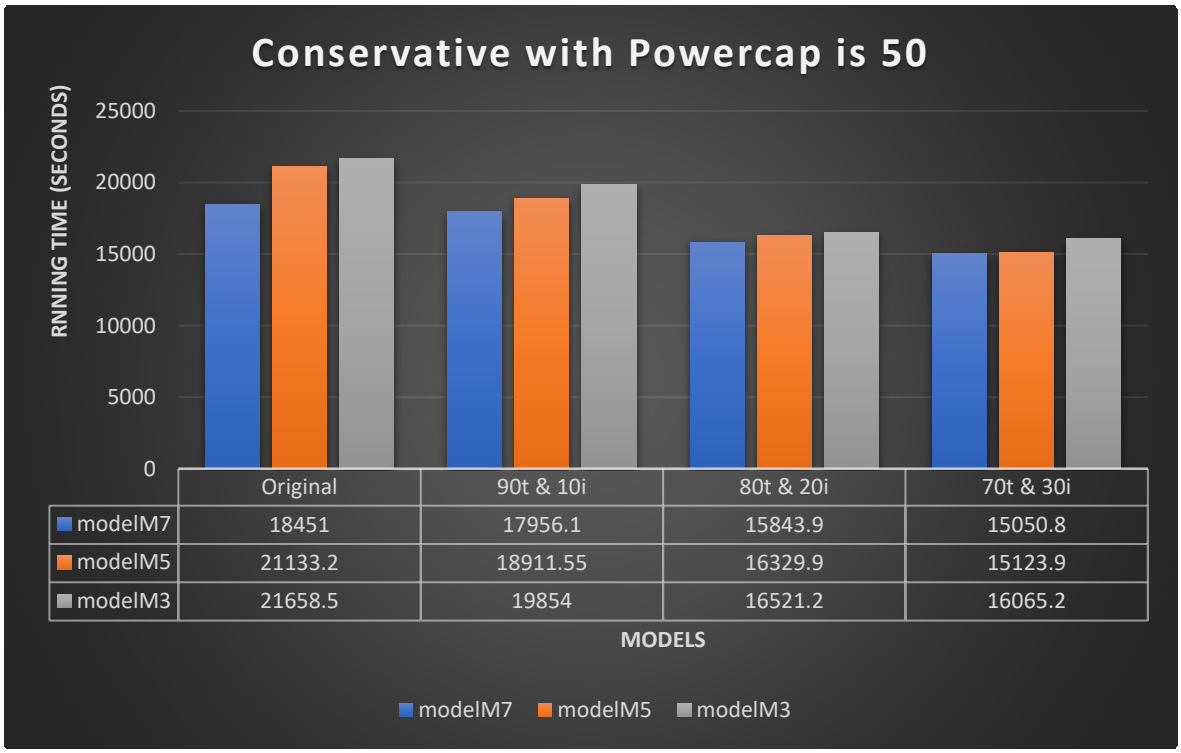


Table 5-4

Energy and Power Consumption in Conservative Mode for CPU Power Cap is 60

Governor	Model				Original	90% training & 10% inference	80% training & 20% inference	70% training & 30% inference
conservative	modelM3	socket 0 on CPU 0	Domain PKG	Energy consumed (Joules)	85499.0	79525.8	75867.0	69516.5
				Power consumed (Watt)	6.3373	5.62266	4.3457	3.2775
			Domain DRAM	Energy consumed (Joules)	55751.8	47270.2	42802.2	40830.57
				Power consumed (Watt)	3.51973	2.62158	1.61217	1.192528
		socket 1 on CPU 14	Domain PKG	Energy consumed (Joules)	196949	182005	33586.8	20391.4
				Power consumed (Watt)	10.961393	10.0939	2.37467	2.17467
			Domain DRAM	Energy consumed (Joules)	60104.8	57397	31235.5	13924.8
				Power consumed (Watt)	4.656513	3.18321	2.20843	1.02877
	modelM5	socket 0 on CPU 0	Domain PKG	Energy consumed (Joules)	78760.5	61500.2	132095	124634
				Power consumed (Watt)	3.67103	3.11214	8.57212	9.0825
			Domain DRAM	Energy consumed	5620.22	57939.4	50648.6	57638.7
				Power consumed (Watt)	1.261959	2.93195	1.98889	6.51266
		socket 1 on CPU 14	Domain PKG	Energy consumed (Joules)	96082.4	84203.0	81436.9	78918.7
				Power consumed (Watt)	7.3479	6.2476	5.28472	5.8564

			Domain DRAM	Energy consumed (Watt)	42151.1	3494.54	39919.7	25594.4
				Power consumed (Watt)	3.02251	2.176837	2.59053	1.176837
	modelM7	socket 0 on CPU 0	Domain PKG	Energy consumed (Joules)	74496.2	24134.3	12660.1	10898.2
				Power consumed (Watt)	9.49044	8.6862	8.26313	7.8184
			Domain DRAM	Energy consumed (Joules)	54726.7	44805.3	30048.1	4909.79
				Power consumed (Watt)	6.51338	2.54085	1.96121	0.230043
		socket 1 on CPU 14	Domain PKG	Energy consumed	76417.9	63228.7	24944.8	16637.6
				Power consumed (Watt)	4.98772	7.52525	8.16876	9.43495
			Domain DRAM	Energy consumed	58956.6	54682.5	39339.3	14824.9
				Power consumed (Watt)	7.0168	3.10097	2.56764	1.694607

It can be observed that as the power limit of the CPU continues to decrease, the power and energy of the CPU is also continuously reducing.The pressure on GPU increased graduall y then GPU starts to take more load to maintain a balance while the power of the DRAM rem ain unchanged .

Here we can see that when CPU power cap is 60 modelM3 needs 19131.00 seconds to co mplete the CNN program but when the CPU power cap is 50 modelM3 is 18451.00 s. In this situation we can see from comparing the above tables that when CPU is power cappped to GPU start to take more load which increase the GPU utilization 20-30% from 6-12% this can be analyze by nvitop tools(nvidia-smi).If we reduce the powercap further more than 40 then GPU and CPU start to consume more power and takes more time to execute the program.

This phenomenon is evident for other two model M5 and M7. Therefore we can say that setti ng the CPU powercap at 40 is the most optimal under this condition

5.2.2 GPU Power Capping

The CNN model operation table of performance, conservative, ondemand, powersave, and userspace (800KHz) respectively and the CPU power limit is 60 Watt, the DRAM and GPU power limit are 12Watt and 250 Watt (Default) where we used, batch_size=120 for training

Table 5-5

Running Time when CPU 60 Watt, DRAM 12Watt and GPU 250 Watt

		Original	90% training & 10% inference	80% training & 20% inference	70% training & 30% inference
Governor	Model	Runtime (seconds)			
performance	modelM7	9180.81	8820.45	8412.78	7569.06
	modelM5	9341.54	8910.78	8509.45	7735.06
	modelM3	9425.47	8954.10	8544.10	8003.88
conservative	modelM7	19131.00	18031.10	16243.80	15108.90
	modelM5	21483.20	19121.40	16409.90	15373.90
	modelM3	21718.50	19914.00	16721.20	16225.20
ondemand	modelM7	19483.20	17621.74	15282.50	14457.70
	modelM5	19948.50	17743.92	15417.30	14618.60
	modelM3	20431.00	17713.08	15943.80	14993.30
powersave	modelM7	22210.20	19970.40	19070.40	19203.40
	modelM5	22954.60	20228.40	19928.40	19638.40
	modelM3	23241.90	20975.60	20445.60	20122.60
userspace	modelM7	19110.20	18775.00	18265.00	17158.50
	modelM5	19842.90	19186.10	18885.10	17665.10
	modelM3	20104.60	19933.00	18533.00	17907.90

The CNN model operation table of performance, conservative, ondemand, powersave, and userspace (800KHz) respectively and the CPU power limit is 60Watt, the DRAM power limit is 12 Watt and GPU was set to 230 Watt where we used batch size=120 for training

Table 5-6

Running Time when GPU 230,Dram 12 & CPU 60

		Original	90% training & 10% inference	80% training & 20% inference	70% training & 30% inference
Governor	Model	Runtime (seconds)			
performance	modelM7	9299.90	8866.00	8441.74 s	7588.00 s
	modelM5	9487.59	8934.18	8535.66	7740.50
	modelM3	9567.78	8999.10	8556.44	8022.58
conservative	modelM7	19155.00	18155.20	16276.30	15540.70
	modelM5	21522.10	19185.50	16450.45	15642.55
	modelM3	21799.00	19989.00	16755.25	16315.90
ondemand	modelM7	19213.60	17688.80	15320.70 s	14510.65
	modelM5	20178.00	17802.90	15717.50 s	14680.90
	modelM3	20888.80	17863.20	16155.80 s	15230.30
powersave	modelM7	22278.50	200890.50	19111.00 s	19264.00
	modelM5	23104.55	20289.10	20176.75 s	19710.20
	modelM3	23457.60	21375.60	20510.70 s	20212.55
userspace	modelM7	20144.40	19210.20	18615.00 s	17611.20 s
	modelM5	20280.50	19525.90	18950.70 s	17812.50 s
	modelM3	20500.45	20010.90	190103.00 s	18050.90 s

The CNN model operation table of performance, conservative, ondemand, powersave, and userspace (800KHz) respectively and and the CPU power limit is 60Watt, the DRAM power limit is 12 Watt and GPU was set to 210 Watt where we used batch size=120 for training

Table 5-7

Running time when GPU 210,Dram 12 & CPU 60

		Original	90% training & 10% inference	80% training & 20% inference	70% training & 30% inference
Governor	Model	Runtime (seconds)			
performance	modelM7	9359.50	8910.10	8490.74	7620.00
	modelM5	9570.77	9010.18	8577.22	7790.30
	modelM3	9640.30	9080.20	8600.50	8078.55
conservative	modelM7	19210.00	18190.20	16310.70	15580.90
	modelM5	21590.20	19245.00	16490.80	15695.55
	modelM3	21850.00	20059.00	16800.20	16375.60
ondemand	modelM7	19790.30	17720.50	15380.70	14550.65
	modelM5	20250.00	17855.70	15787.50	14740.90
	modelM3	20920.80	17910.20	16190.70	15290.60
powersave	modelM7	22310.50	20950.50	19190.00	19320.00
	modelM5	23155.20	20340.70	20250.80	19790.20
	modelM3	23490.60	21430.60	20580.70	20278.45
userspace	modelM7	20189.40	19390.20	18850.00	18278.20
	modelM5	20520.50	19785.960	19020.70	18968.50
	modelM3	20805.00	20150.90	19190.00	19000.90

It can be found from the table that, comprehensive perspective of performance and energy consumption, when the GPU power limit is the highest, that is, at 250 Watt, the performance of the CNN program reaches the highest energy consumption. But when the power consumption is the highest, the power limit is 230 Watt, and the performance and energy consumption ratio of 250 Watt and 230 Watt is almost the same. We can compare that with other configurations unchanged, the GPU power limit with the best performance-to-energy ratio is between 210 Watt and 250 Watt.

We can see that performace take the least time to finish the task but it consumes a large amount of energy compared to other modes.By careful observation and doing lot of experiments in different scenario.We can come to the conclusion that, conservative is the best option based on running time of the program and energy consumed.

Chapter Six : Optimize Energy Efficiency by Selecting the Optimal Parameters, DVFS policies, Power Capping Values and Model Optimization

6.1 Energy Efficiency by using Model Optimization

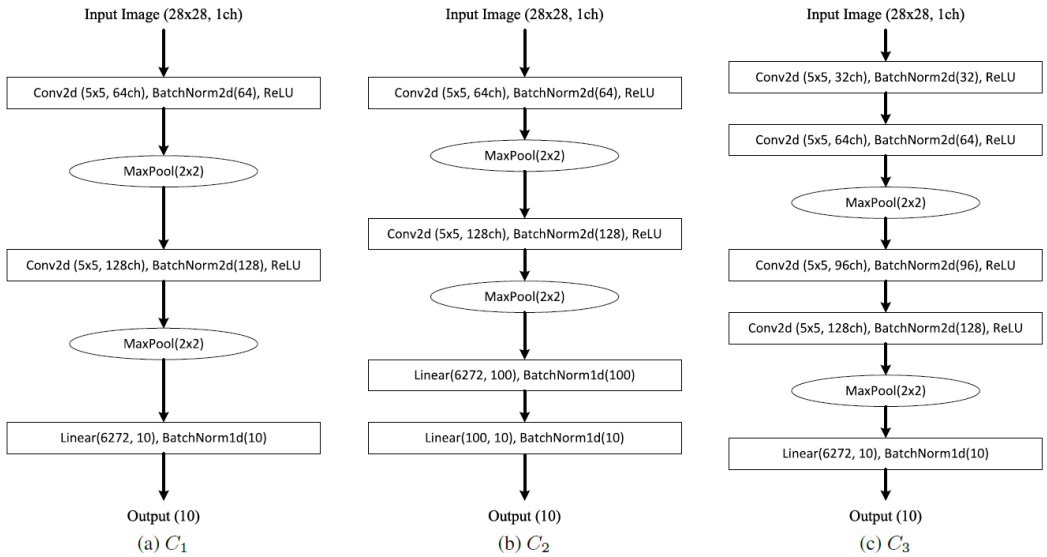
Since the early 2010s, computer vision has been dominated by the introduction of convolutional neural networks (CNNs), which have yielded unprecedented success in previously challenging tasks such as image recognition, image segmentation or object detection, among others. Considering the theory of neural networks was mostly developed decades earlier, one of the main driving factors behind this evolution was the widespread availability of high-performance computing devices and general purpose Graphic Processing Units (GPU). In parallel with the increase in computational requirements [32], the last decades have seen a considerable development of portable, miniaturized, battery-powered devices, which pose constraints on the maximum power consumption.

Attempts at reducing the power consumption of traditional deep learning models have been made. Typically, these involve optimizing the network architecture, in order to find more compact networks (with fewer layers, or fewer neurons per layer) that perform equally well as larger networks. One approach is energy-aware pruning, where connections are removed according to a criterion based on energy consumption, and accuracy is restored by fine-tuning of the remaining weights [33]. Other work looks for more efficient network structures through a full-fledged architecture search [34]. The latter work was one of the winners of the Google "Visual Wake Words Challenge" at CVPR 2019, which sought models with memory usage under 250 kB, model size under 250 kB and per-inference multiply-add count (MAC) under 60 millions.

When building a CNN, a common practice is to use pooling, such as max pooling or average pooling [35]. Pooling is used to obtain translation invariance and also reduce dimension of the feature maps. A commonly used CNN model consists of a set of convolution layers where each convolution layer is followed by a pooling layer, and one or multiple fully connected layers at the end. Some networks have two convolution layers before the pooling layer. Figure 5 show some of the commonly used CNN structures, and we name the three networks C_1, C_2, and C_3

Figure 7

CNN Structures with Max Pooling

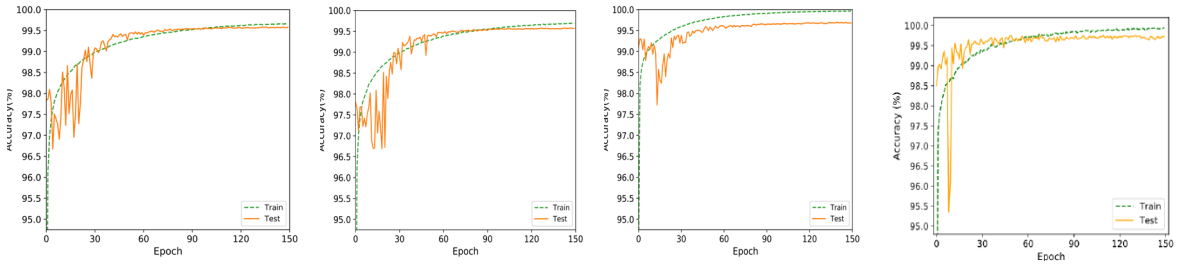


The change in training and test accuracy during training. It can be observed that for networks using max pooling, the test accuracy goes through oscillations in the early stage of training. On the other hand, the test accuracy of M5 increases in a more stable manner. Table 6 shows the test accuracy of 30 networks between 50 epoch and 150 epoch of training. The average test accuracy of C_3 and M_5 is better than that of C_1 and C_2, which means using more convolution layers could result in better feature learning. Having more fully connected layers

at the end did not help, as can be seen from the accuracy of C_1 and C_2. Between C_3 and M_5, M_5 achieves higher accuracy in general, and also can reach higher accuracy in the best case.

Figure 8

Train accuracy and test accuracy of C1, C2, C3, and M5 during training



(a) C_1 (b) C_2 (c) C_3 (d) M_5

Figure 9

Test accuracy of networks measured between 50 epoch and 150 epoch in training.

model	test accuracy			
	min	avg	max	best
C_1	99.3052 ± 0.0865	99.5293 ± 0.0105	99.6419 ± 0.0059	99.70
C_2	99.3594 ± 0.0442	99.5316 ± 0.0090	99.6337 ± 0.0051	99.68
C_3	99.4720 ± 0.04268	99.6448 ± 0.0078	99.7372 ± 0.0033	99.78
M_5	**99.5863 ± 0.0115**	**99.6835 ± 0.0074**	**99.7583 ± 0.0081**	**99.80**

Besides accuracy, computation complexity is another important consideration. Real world tasks often aim at obtaining best accuracy under a limited computational budget, given by target platform (e.g., hardware) and application scenarios (e.g., auto driving requires low latency). This motivates a series of works towards light-weight architecture design and better speed-accuracy trade-off, including Xception , MobileNet , MobileNet V2 , ShuffleNet, and CondenseNet .

6.2 Optimize Energy Efficiency by Selecting the Optimal Parameters, DVFS policies, Power Capping Value

6.2.1 DVFS Policies

Dynamic voltage and frequency scaling (DVFS) techniques are used to dynamically alter the voltage and frequency levels of the CPU to the lowest possible level while still maintaining high performance.

Current research has yielded several different DVFS techniques. The first group of techniques uses known task arrival times, workload, and deadlines to implement algorithms at the task level in the operating system. The second group of techniques uses compiler or application support for performing DVFS. For example, at compile time, the compiler can identify which sections of code will perform many memory accesses and which will not. The third group of techniques, however, achieves DVFS without any compiler support or any information regarding task characteristics. Rather, these techniques use runtime statistics to identify the workload of a task and thus estimate and predict the optimal v/f setting. These techniques can be further classified as fine-grained or course-grained. Course-grained techniques determine the v/f setting on a task-by-task basis. Fine-grained techniques adjust the v/f setting within a task boundary and usually perform better than course-grained techniques. We specifically researched four different DVFS techniques described below that only use run-time statistics for setting CPU voltage and frequency.

We can see that performace take the least time to finish the task but it consumes a large amount of energy compared to other modes.By careful observation and doing lot of experiments in different scenario.We can come to the conclusion that, conservative is the best option based on running time of the program and energy consumed.

6.2.2 CPULIMIT

Cpulimit is a tool which limits the CPU usage of a process (expressed in percentage, not in CPU time). It is useful to control batch jobs, when you don't want them to eat too many CPU cycles. The goal is prevent a process from running for more than a specified time ratio. It does not change the nice value or other scheduling priority settings, but the real CPU usage. Also, it is able to adapt itself to the overall system load, dynamically and quickly. The control of the used CPU amount is done sending SIGSTOP and SIGCONT POSIX signals to processes. All the children processes and threads of the specified process will share the same percentage of CPU.

In many applications, the CPU and GPU work together to perform different tasks. The CPU handles general-purpose computing tasks, while the GPU is specialized for parallel processing and graphics-related computations. In such cases, both the CPU and GPU may be utilized simultaneously to achieve optimal performance.

However, it's important to note that the CPU and GPU utilization can vary depending on the specific application or task. Some tasks may be more CPU-bound, meaning they require more CPU processing power and have a lower demand on the GPU. Conversely, other tasks may be more GPU-bound, placing a heavier workload on the GPU while the CPU utilization remains relatively low.

When we set a CPU limit for a process or application, you are restricting its CPU usage to a specific percentage or value. This limit ensures that the process does not consume excessive CPU resources, which can lead to system slowdowns, unresponsiveness, or even system instability.

By setting a CPU limit, you effectively cap the amount of CPU processing power that the process can utilize. The process will be allowed to use CPU resources up to the specified limit, but it will not be able to exceed that limit.

The CPU limit is typically expressed as a percentage of the CPU's total processing power. For example, setting a CPU limit of 50% means that the process can utilize a maximum of half of the available CPU resources.

When a process reaches its CPU limit, it may experience reduced performance, longer processing times, or slower response times. However, this also ensures that other processes running on the system have access to sufficient CPU resources, preventing one process from monopolizing the CPU and affecting the overall system performance.

CPU limiting can be useful in various scenarios, such as:

- Preventing certain processes from consuming excessive CPU resources, particularly in multi-user systems or shared environments.
- Controlling the CPU usage of resource-intensive applications to prevent system slowdowns or overheating.
- Prioritizing critical processes or ensuring fair distribution of CPU resources among different applications.

It's important to note that the method of setting CPU limits can vary depending on the operating system and the tools available. The specific commands or utilities used to set CPU limits may differ between different platforms.

The CNN model operation table of performance, conservative, ondemand, powersave, an d userspace (800KHz) respectively and GPU was set to 250 Watt and cpulimit -l 10 where we used batch size=120 for training

Table 6-1

cpulimit -l 10,batch_size =120 & GPU 250

		Original	90% training & 10% inference	80% training & 20% inference	70% training & 30% inference
Governor	Model	Runtime (seconds)			
performance	modelM3	8870.50	8761.50	7750.50	7180.50
	modelM5	9005.34	8750.78	8096.86	7563.86
	modelM7	9100.50	8820.80	8378.50	7720.70
conservative	modelM3	18030.50	17531.10	15943.80	15008.90
	modelM5	20050.20	18361.40	16109.90	15573.90
	modelM7	20710.50	18734.00	16421.20	15945.20
ondemand	modelM3	18380.20	16421.74	14982.50	14187.70
	modelM5	18910.50	16743.92	15017.30	14360.60
	modelM7	19131.00	17113.08	15245.80	14677.30
powersave	modelM3	20820.20	18970.40	18850.40	18107.40
	modelM5	21524.60	19428.40	19228.40	18458.40
	modelM7	21500.60	20175.60	20045.60	19092.60
userspace	modelM3	18010.20	17975.00	18865.00	16858.50
	modelM5	18422.90	18136.10	18285.10	17160.10
	modelM7	18914.50	1883.50	17933.00	17588.90

In this case, the "-l 10" parameter indicates that we limited the CPU usage to 10% for the specified process.When we run this command, the process we specified will be restricted to using a maximum of 10% of the CPU's processing power. This can be useful in situations where

we want to prevent a process from consuming excessive CPU resources and causing system slowdowns or overheating.

After completing the experiment using the following commad, I was able to restrict the cpu utilization then the GPU usage and GPU utlization increases using nvitop(nvidia-smi).That gave me the best result compared to other experiment because this was the fastest time to complete the CNN program and also consumed lower energy.

However, it's important to note that the CPU and GPU utilization can vary depending on the specific application or task. Some tasks may be more CPU-bound, meaning they require more CPU processing power and have a lower demand on the GPU. Conversely, other tasks may be more GPU-bound, placing a heavier workload on the GPU while the CPU utilization remains relatively low.

In summary, while it is possible for CPU utilization and GPU usage to increase simultaneously, it ultimately depends on the specific workload and the distribution of tasks between the CPU and GPU in the given application or task.

6.2.3 Power Capping Value

Power capping is a technique used to limit the power consumption of a system or device, particularly in environments where power constraints or energy efficiency are important considerations. It involves setting a maximum power limit, also known as the power cap or power target, which restricts the amount of power that the system can consume.

The power cap is typically expressed in watts and represents the maximum power level that the system is allowed to draw from the power supply. By setting a power cap, the system's power usage is effectively capped at that level, preventing it from exceeding the specified limit.

Power capping is commonly implemented in data centers, where reducing power consumption can lead to cost savings and improved energy efficiency. It helps prevent power usage spikes, optimizes power allocation, and ensures that the overall power draw of the data center remains within manageable limits.

When a power cap is set, the system or its components, such as processors, graphics cards, or other power-consuming devices, will dynamically adjust their performance and power consumption to stay within the defined limit. This adjustment is achieved by employing techniques such as dynamic voltage and frequency scaling (DVFS), where the voltage and frequency of the components are scaled down to reduce power consumption.

Figure 10

Running Time when CPU 60 Watt, DRAM 12Watt and GPU 250 Watt(Default) where we used batch_size=120 for training

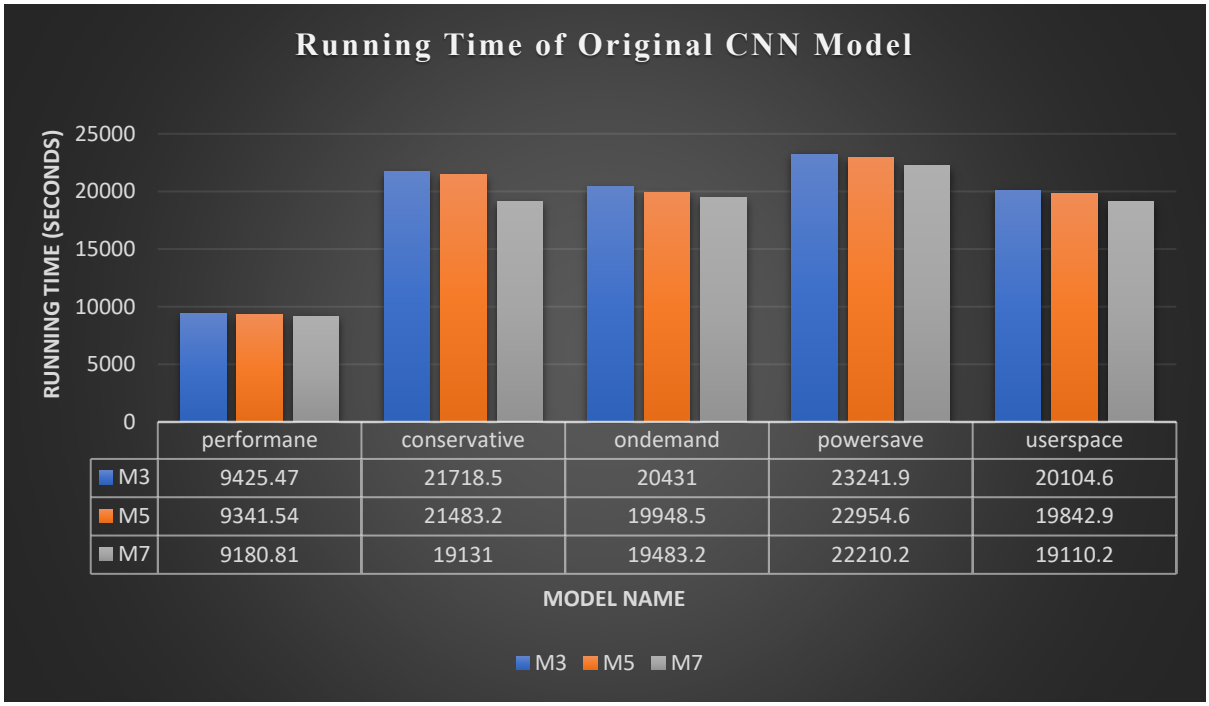


Figure 11

Running Time when CPU 60 Watt, DRAM 12Watt and GPU 250 Watt (Default) where we used batch_size=120 for training

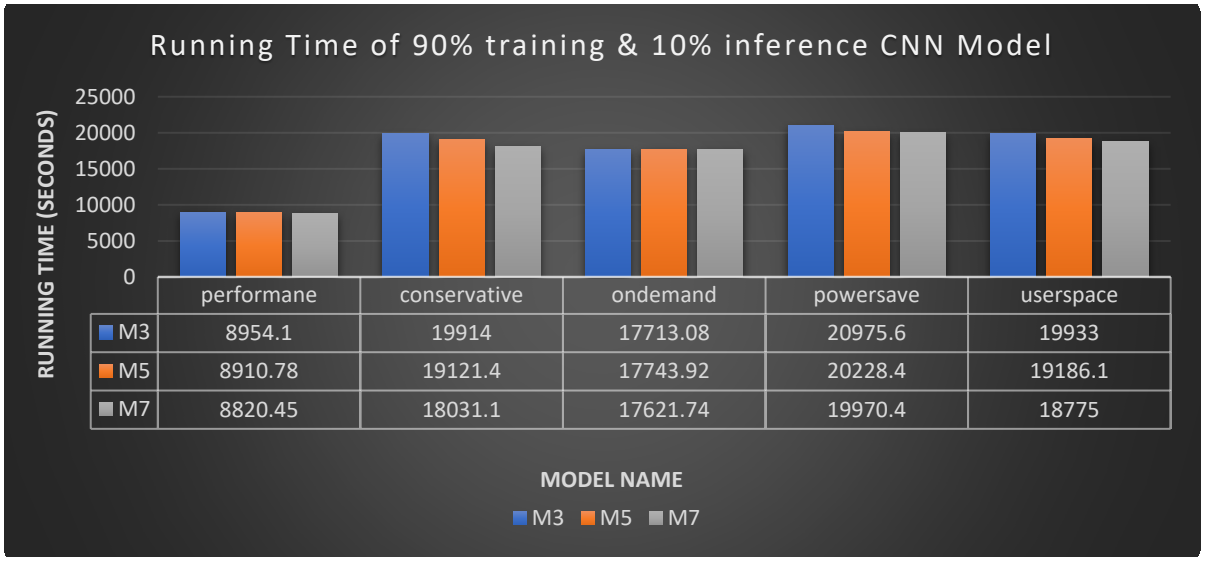


Figure 12

Running Time when CPU 60 Watt, DRAM 12Watt and GPU 210 Watt where we used batch_size=120 for training

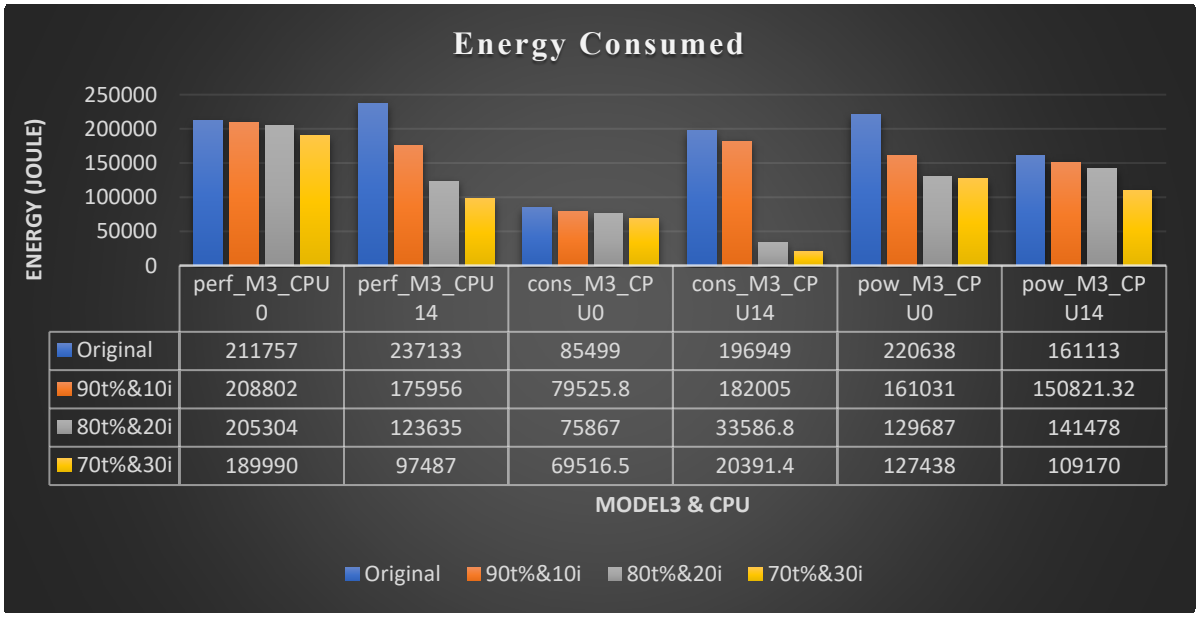


Figure 13

Energy consumed when CPU 60 Watt, DRAM 12 Watt and GPU 210 Watt for Original Model

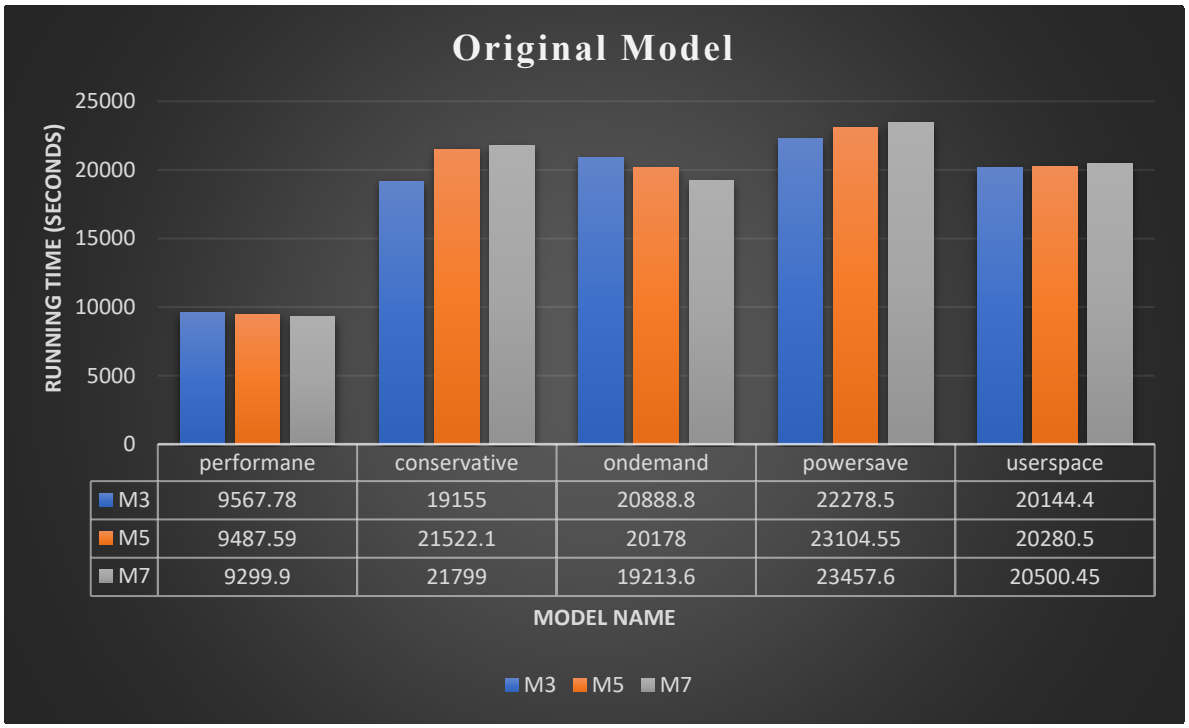


Figure 14

Energy consumed when CPU 60 Watt, DRAM 12 Watt and GPU 210 Watt

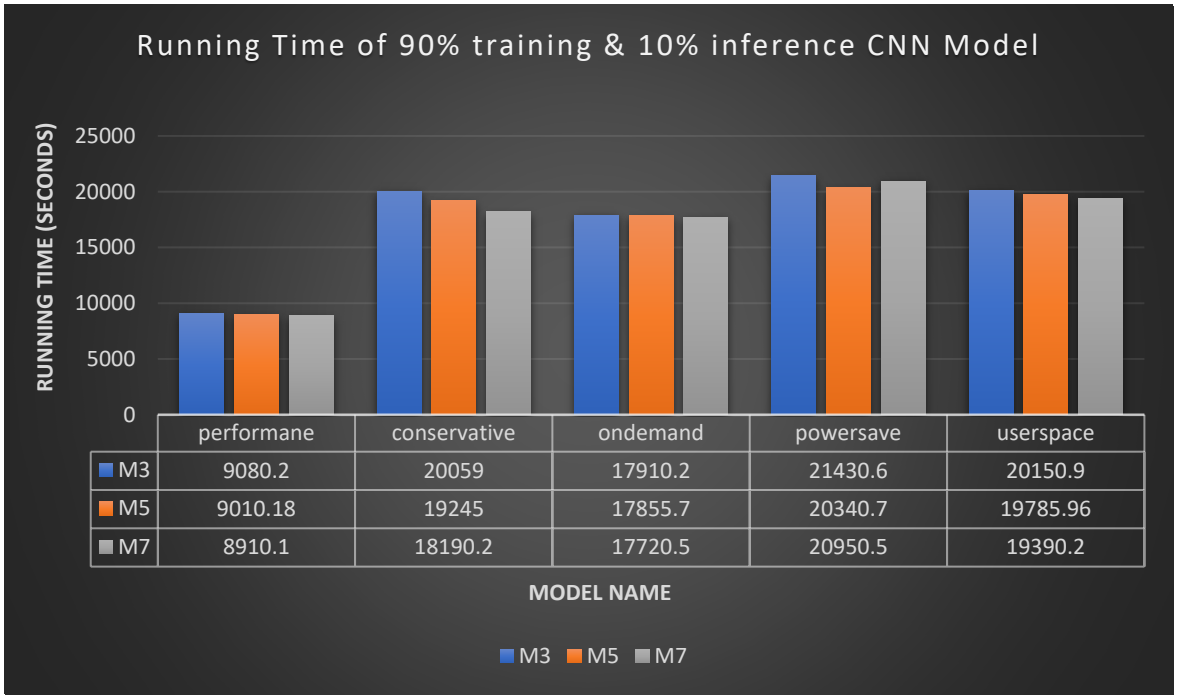
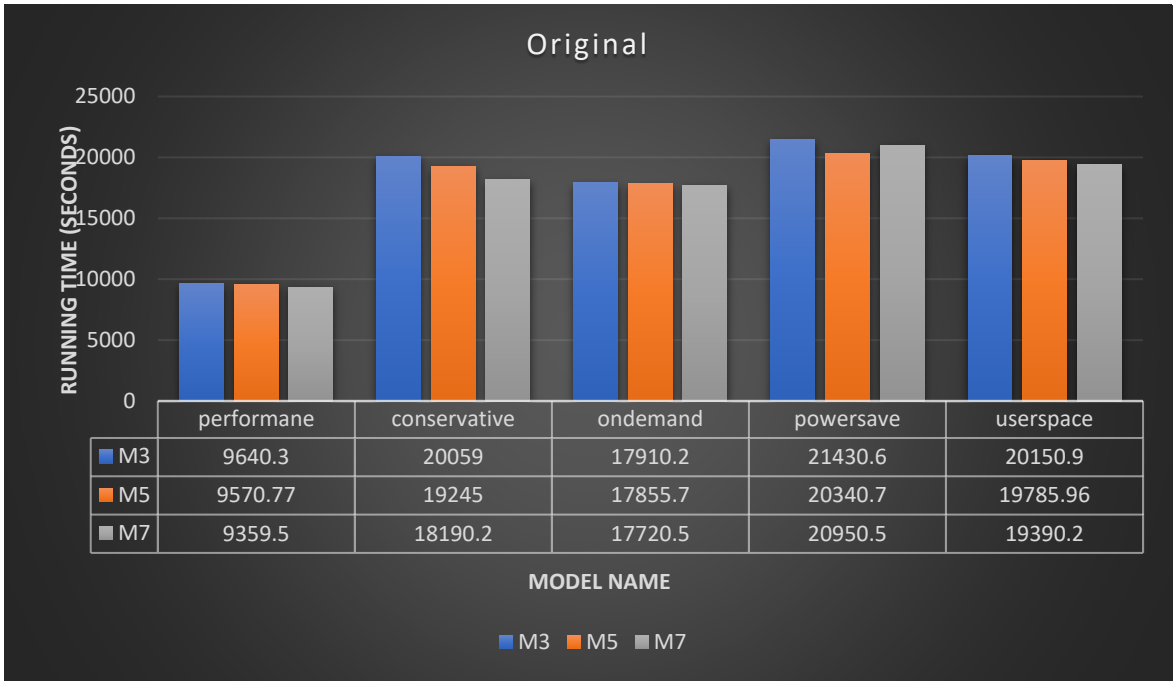


Figure 15

Energy consumed when CPU 60 Watt, DRAM 12 Watt and GPU 210 Watt



We can see that,performance mode(perf) consume maximum energy.Surprisingly powersave also consume a lot of energy because it runs for a long time compared to other modes.By doing different experiment in different ,we can see that conservative mood gives the optimal solution for this problem which has a balance tradeoff between time and energy consumption.

The energy-delay product is particularly relevant in scenarios where power consumption and time constraints are important considerations, such as in battery-powered devices or real-time systems. By evaluating the energy-delay product, designers and engineers can assess the efficiency of a device or system and make informed decisions to optimize its performance.

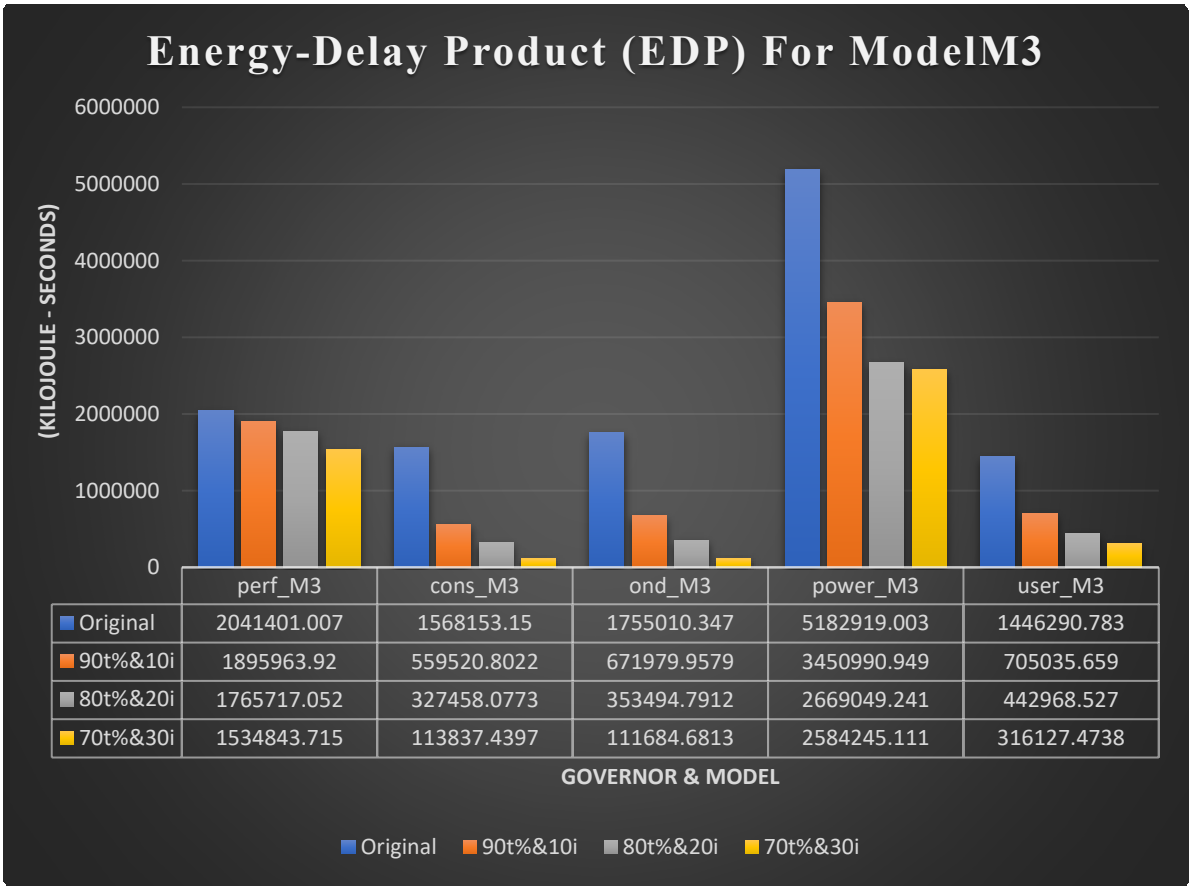
Reducing the energy-delay product often involves finding ways to decrease power consumption without significantly impacting the performance or functionality of the device or system. This can be achieved through various techniques, such as optimizing the circuit design,

using low-power components, employing power management strategies, or implementing efficient algorithms.

By minimizing the energy-delay product, electronic devices and systems can achieve improved energy efficiency, longer battery life, reduced heat dissipation, and enhanced overall performance.

Figure 16

Energy-Delay Product (EDP) For ModelM3



	perf_M3	cons_M3	ond_M3	power_M3	user_M3
Original	2041401.007	1568153.15	1755010.347	5182919.003	1446290.783
90t%&10i	1895963.92	559520.8022	671979.9579	3450990.949	705035.659
80t%&20i	1765717.052	327458.0773	353494.7912	2669049.241	442968.527
70t%&30i	1534843.715	113837.4397	111684.6813	2584245.111	316127.4738

We can see from Figure 17, performance mode(perf) consume maximum energy.Similar to our previous experiment and analysis we can present that, powersave also consume a lot of energy because it runs for a long time compared to other modes.From the EDP table,it's edident that, conservative mood gives the optimal solution for this problem which has a balance tradeoff between time and energy consumption.

Conclusion

This paper generally analyzes and optimizes energy efficiency for A Typical Deep Learnin g Application.

This paper starts with the basic DVFS technology, learns the principle of DVFS technolo gy and understands how DVFS technology is used in the energy efficiency optimization for A Typical Deep Learning Application. Then it introduces the use of DVFS technology on Linux system, furthermore discuss about the CPU frequency regulation strategy of Linux system, and then learns the principles, advantages and disadvantages of various frequency regulation strategies from the source code level, and then uses CNN model test program to test various fr equency regulation strategies. Preliminary tests of performance, energy consumption, power c onsumption and other indicators show that the best frequency tuning method for A Typical Deep Learning Application is conservative.

Then this paper introduces another important topic for A Typical Deep Learning Application energy efficiency optimization technology: Power Capping technology. This paper starts with the introduction of the principle of Power Capping technology, and then shows the energy efficiency optimization of CNN and deep learning model by some researchers using Power Capping technology. Then, we introduced how Power Capping technology is implemented and used in CPU and GPU, in this paper. After learning how to use it, we used Power Capping technology to conduct experiments on CPU and GPU analyzed the test results, and then analyzed the specific functions of Power Capping technology on these two, laying the foundation for subsequent combined tests.

Then, in order to verify that the frequency adjustment strategy of the CPU and the Power Capping technology can be used overlappingly, we conducted an overlap test and found that the Power Capping technology can indeed be used together with the frequency adjustment

strategy of the CPU. Then to the last step, in order to optimize the subsequent energy efficiency, it is necessary to find out the best frequency adjustment method under the limitation of GPU Power Capping. Various frequency modulation strategies, based on performance, energy consumption, power consumption and other indicators, find out that the best frequency modulation strategy is conservative, and subsequent energy efficiency optimization .

Then we used a light weight CNN model for Energy Efficiency Analysis and Optimization known as Model optimization which refers to the process of improving the performance, efficiency, and resource utilization of a machine learning model. It involves various techniques aimed at reducing model size, decreasing computational requirements, improving inference speed, and enhancing overall model quality

References

[1] E. Mark, J. Chao, D. Minh Ngoc, A. David and S. Bronis R, "Energy Efficiency Modeling of Parallel Applications," *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis,* pp. 212-224, 2018.

[2] "The Linux Kernel," 2017. [Online]. Available: https://www.kernel.org/doc/html/v4.14/ ad min gui de/pm/cpufreq.html.

[3] "The Linux Kernel," [Online]. Available: https://www.kernel.org/doc/html/latest/.

[4] . W. Song, C. Yang, W. Xinhou, J. Hai, L. Fangming, C. Haibao and Y. Chuxiong, "Precise Power Capping for Latency-Sensitive Applications in Datacenter," *IEEE Transactions on Sustainable Computing,* pp. 1-1, 2018.

[5] H. Meng, Z. Weizhe, W. Yiming, L. Gangzhao and V. A. V, "Fine-Grained Powercap Allocation for Power-Constrained Systems Based on Multi-Objective Machine Learning," *IEEE Transactions on Parallel and Distributed Systems,* pp. 1-1, 2020.

[6] W. Zhao, H. Fu, W. Luk, T. Yu, S. Wang, B. Feng, Y. Ma and G. Yang, "F-CNN: An FPGA-based framework for training Convolutional Neural Networks," in *IEEE*, London, UK, 2016.

[7] C. Zhaoyun, Q. Wei, W. Mei, F. Jianbin, Y. Jie, Z. Chunyuan and L. Lei, "Deep Learning Research and Development Platform: Characterizing and Scheduling with QoS Guarantees on GPU Clusters," *IEEE Transactions on Parallel and Distributed Systems,* vol. 31, pp. 34-50, 2019.

[8] J. Weile, W. Han, C. Mohan, L. Denghui, L. Jiduan, L. Lin, C. Roberto, W. E. and Z. Linfeng, "Pushing the limit of molecular dynamics with ab initio accuracy to 100 million atoms with machine learning," 2020.

[9] "ANACONDA," [Online]. Available: https://www.anaconda.com/ (accessed on 15 January 2023).

[10] A. Sanghyeon, L. Minjun , P. Sanglee, a. Heerin and S. Jungmin, "An Ensemble of Simple Convolutional Neural Network Models for MNIST Digit Recognition," 2020.

[11] M. Rozenwald, A. Galitsyna , G. Sapunov , E. Khrameeva and M. Gelfand , "A machine learning framework for the prediction of chromatin folding in Drosophila using epigenetic features," *PeerJ Computer Science,* vol. 6, p. e307, 2020.

[12] C. Amrit , T. Paauw , R. Aly and M. Lavric , "Identifying child abuse through text mining and machine learning," *Expert Systems with Application,* vol. 88, pp. 402-418, 2017.

[13] W. Liu , Z. Wang , X. Liu , N. Zeng , Y. Liu and F. Alsaadi , "A survey of deep neural network architectures and their applications," *Neurocomputing,* vol. 234, pp. 11-26, 234.

[14] T. Potok , C. Schuman , S. Young , R. Patton , F. Spedalieri , J. Liu , K. Yao, G. Rose and G. Chakma , "A Study of Complex Deep Learning Networks on High Performance, Neuromorphic, and Quantum Computers," *IEEE Press,* 2017.

[15] M. Abadi, B. Paul , C. Jianmin, C. Zhifeng and Z. Xiaoqiang, "TensorFlow: A system for large-scale machine learning," *USENIX Association,* 2016.

[16] B. James , B. Olivier, F. Bastien, L. Pascal and B. Yoshua, "Theano: A CPU and GPU Math Compiler in Python}," *hgpu.org,* 2010.

[17] C. Tianqi , L. Mu , L. Yutian, L. Min , W. Naiyan , W. Minjie, X. Tianjun, X. Bing , Z. Chiyuan and Z. Zheng, "MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems," *Statistics,* 2015.

[18] Y. Dong , E. Adam, S. M. L, Y. Kaisheng and H. Zhiheng, "An Introduction to Computational Networks and the Computational Network Toolkit," *microsoft research.*

[19] J. Yangqing, S. Evan, D. Jeff , K. Sergey , L. Jonathan , G. Ross, G. Sergio and D. Trevor, "Caffe: Convolutional Architecture for Fast Feature Embedding," *ACM,* 2014.

[20] T. Seiya , O. Kenta and H. Shohei, "Chainer: a Next-Generation Open Source Framework for Deep Learning".

[21] C. Ronan , K. Koray and C. Farabet, "Torch7: A Matlab-like Environment for Machine Learning," in *BigLearn NIPS Workshop*, 2011.

[22] F. Chollet, "Keras," 2015. [Online]. Available: https://github.com/fchollet/keras.

[23] "Automatic differentiation in PyTorch," in *31st Conference on Neural Information Processing Systems (NIPS 2017)*, Long Beach, CA, USA., 2017.

[24] A. Tahmid , S. Colin, K. Amey and M. Tinoosh, "Accelerating Convolutional Neural Network With FFT on Embedded Hardware," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. PP, pp. 1-24, 2018.

[25] B. Adam, K. Tatiana and D. Ian, "A Branching and Merging Convolutional Network with Homogeneous Filter Capsules," arXiv preprint arXiv:2001.09136, 2020.

[26] K. Kamran, H. Mojtaba , B. E. Donald , M. Kiana Jafari and B. Laura E., "RMDL: Random Multimodel Deep Learning for Classification," 2018.

[27] W. Li , . Z. Matthew D, Z. Sixin , L. Yann and F. Rob, "Regularization of Neural Networks using DropConnec," *International Conference on Machine Learning,* 2013.

[28] S. Sabour, N. Frosst and G. E. Hinton, "Dynamic routing between capsules," *Advances in Neural Information Processing Systems 30 (NIPS 2017),* pp. 3859-3869, 2017.

[29] "Linux power management (4) CPUFreq," [Online]. Available: https://blog.csdn.net/zhouhuacai/article/details/78155743.

[30] P. Adam, G. Sam , M. Francisco, A. Lerer, J. Bradbury and G. Chanan, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *in Advances in Neural Information Processing Systems (NeurIPS) (2019).*, 2019.

[31] "Averaging Weights Leads to Wider Optimal and Better Generalization," *Conference on Uncertainty in Artificial Intelligence (UAI) ,* (2018).

[32] S. Emma , G. Ananya and M. Andrew, "Energy and Policy Considerations for Deep Learning in NLP," 2019.

[33] "Pruning convolutional neural networks for resource efficient inference," arXiv:1611.06440, 2016.

[34] C. Han , Z. Ligeng and H. Song, "Proxylessnas: direct neural architecture search on target task and hardware," arXiv:1812.00332.

[35] "Unsupervised learning of invariant feature hierarchies," *IEEE,* pp. 1-8, 2007.

[36] M. Rhu, G. Natalia, J. Clemons, A. Zulfiqar and K. Stephen W, "vDNN: Virtualized deep neural networks for scalable, memory-efficient neural network design," *ACM,* 2016.

[37] G. Priya , D. Piotr , G. Ros, N. Pieter, W. Lukasz, K. Aapo, T. Andrew, J. Yangqing and H. Kaiming, "Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour," 2017.

[38] G. Priya, D. Piotr, G. Ross, N. Pieter, W. Lukasz, K. Aapo , T. Andrew, J. Yangqing and H. Kaiming, "Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour".

[39] H. Song and M. Huizi, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," *Fiber,* vol. 56, pp. 3--7, 2015.

Appendices

Table Appendices 1

Energy & Power consumtion when GPU 210,Dram 12 & CPU 60

Governor	Model				Original	90% training & 10% inference	80% training & 20% inference	70% training & 30% inference
performance	modelM3	socket 0 on CPU 0	Domain PKG	Energy consumed (Joules)	211757	208802	205304	189990
				Power consumed (Watt)	23.8443	23.1038	24.1266	22.8053
			Domain DRAM	Energy consumed (Joules)	58965.7	53892.7	52938.7	51610.5
				Power consumed (Watt)	6.63967	6.3446	6.22116	6.67229
		socket 1 on CPU 14	Domain PKG	Energy consumed (Joules)	237133	175956	123635	97487
				Power consumed (Watt)	27.867	19.813	17.9261	12.6033
			Domain DRAM	Energy consumed (Joules)	62069.2	59376	57885.8	54098.6
				Power consumed (Watt)	6.98913	5.65256	6.80253	6.99394
	modelM5	socket 0 on CPU 0	Domain PKG	Energy consumed (Joules)	215423	205346	198423	189873
				Power consumed (Watt)	24.0924	22.5033	20.5859	18.725
			Domain DRAM	Energy consumed	59348.3	56875.9	52330.1	53354.3
				Power consumed (Watt)	6.63737	6581715	6.22031	6.66606
			Domain PKG	Energy consumed	199841	19332.5	181760	166274

		socket 1 on CPU 14		(Joules)				
				Power consumed (Watt)	20.113	20.30964	22.5486	18.5272
			Domain DRAM	Energy consumed (Watt)	62479	55827.2	57221.4	55964.7
				Power consumed (Watt)	6.98751	6.42237	6.80173	6.9922
	modelM7	socket 0 on CPU 0	Domain PKG	Energy consumed (Joules)	194217	193918	205503	186792
				Power consumed (Watt)	22.5167	20.3227	24.052	19.4301
			Domain DRAM	Energy consumed (Joules)	57312.7	52741.7	53126.7	50516.1
				Power consumed (Watt)	6.6446	6.08061	6.21794	6.67403
		socket 1 on CPU 14	Domain PKG	Energy consumed	159169	128457	120043	84925.1
				Power consumed (Watt)	18.4534	16.16294	18.0946	15.22
			Domain DRAM	Energy consumed	60319.4	52311.3	58144.1	52939.3
				Power consumed (Watt)	6.99318	6.68872	6.80518	6.99418
conservative	modelM3	socket 0 on CPU 0	Domain PKG	Energy consumed (Joules)	85499.0	79525.8	75867.0	69516.5
				Power consumed (Watt)	6.3373	5.62266	4.3457	3.2775
			Domain DRAM	Energy consumed (Joules)	55751.8	47270.2	42802.2	40830.57
				Power consumed (Watt)	3.51973	2.62158	1.61217	1.192528
		socket 1 on CPU 14	Domain PKG	Energy consumed (Joules)	196949	182005	33586.8	20391.4
				Power consumed (Watt)	10.961393	10.0939	2.37467	2.17467
			Domain DRAM	Energy consumed (Joules)	60104.8	57397	31235.5	13924.8
				Power consumed	4.656513	3.18321	2.20843	1.02877

				(Watt)				
	modelM5	socket 0 on CPU 0	Domain PKG	Energy consumed (Joules)	78760.5	61500.2	132095	124634
				Power consumed (Watt)	3.67103	3.11214	8.57212	9.0825
			Domain DRAM	Energy consumed (Joules)	5620.22	57939.4	50648.6	57638.7
				Power consumed	1.261959	2.93195	1.98889	6.51266
		socket 1 on CPU 14	Domain PKG	Energy consumed (Joules)	96082.4	84203.0	81436.9	78918.7
				Power consumed	7.3479	6.2476	5.28472	5.8564
			Domain DRAM	Energy consumed (Joules)	42151.1	3494.54	39919.7	25594.4
				Power consumed (Watt)	3.02251	2.176837	2.59053	1.176837
	modelM7	socket 0 on CPU 0	Domain PKG	Energy consumed (Joules)	74496.2	24134.3	12660.1	10898.2
				Power consumed (Watt)	9.49044	8.6862	8.26313	7.8184
			Domain DRAM	Energy consumed (Joules)	54726.7	44805.3	30048.1	4909.79
				Power consumed (Watt)	6.51338	2.54085	1.96121	0.230043
		socket 1 on CPU 14	Domain PKG	Energy consumed (Joules)	76417.9	63228.7	24944.8	16637.6
				Power consumed (Watt)	4.98772	7.52525	8.16876	9.43495
			Domain DRAM	Energy consumed (Joules)	58956.6	54682.5	39339.3	14824.9
				Power consumed (Watt)	7.0168	3.10097	2.56764	1.694607
ondemand	modelM3	socket 0 on CPU 0	Domain PKG	Energy consumed (Joules)	83888.3	37519.4	218332	53083.9
				Power consumed (Watt)	5.8735	4.35173	10.1629	3.01824
			Domain DRAM	Energy consumed (Joules)	58629.69	55781.8	46753.4	23724.6
				Power consumed (Watt)	6.26205	6.4699	1.6611	2.6583

		socket 1 on CPU 14	Domain PKG	Energy consumed (Joules)	245041	132836	36980	33867.1
				Power consumed (Watt)	13.9325	6.18326	2.58919	3.92811
			Domain DRAM	Energy consumed (Joules)	65741.0	58817.8	54171.2	52891.5
				Power consumed (Watt)	5.732713	6.82204	4.2525	3.12102
	modelM5	socket 0 on CPU 0	Domain PKG	Energy consumed (Joules)	228409	84286.7	73101.9	70645.7
				Power consumed (Watt)	10.5023	6.06486	4.49816	3.92623
			Domain DRAM	Energy consumed (Joules)	72777.78	56538.8	49353	30755.3
				Power consumed (Watt)	7.334634	6.46607	2.74286	1.99486
		socket 1 on CPU 14	Domain PKG	Energy consumed (Joules)	142233	79328.2	40445.3	61130
				Power consumed (Watt)	6.53993	5.1454	4.62553	6.5127
			Domain DRAM	Energy consumed (Joules)	75531.1	59647.1	39938.9	57668.8
				Power consumed (Watt)	5.807095	6.82156	2.59052	3.20502
	modelM7	socket 0 on CPU 0	Domain PKG	Energy consumed (Joules)	122201	42455.2	13408.1	14549.9
				Power consumed (Watt)	6.38759	4.87259	2.626	2.40881
			Domain DRAM	Energy consumed (Joules)	56602.3	56346.1	51520.7	59811.4
				Power consumed (Watt)	2.95867	6.46684	4.02787	3.04561
		socket 1 on CPU 14	Domain PKG	Energy consumed (Joules)	82388.9	66561.9	45592.4	39272.8
				Power consumed (Watt)	5.30045	3.38934	2.38317	4.50733
			Domain DRAM	Energy consumed (Joules)	65428.8	59439.4	40824.1	3178.36
				Power consumed	3.42005	6.82186	2.62639	0.161843

				(Watt)				
powersave	modelM3	socket 0 on CPU 0	Domain PKG	Energy consumed (Joules)	220638	161031	129687	127438
				Power consumed (Watt)	11.4872	18.8234	15.6055	15.1927
			Domain DRAM	Energy consumed	55880.3	53551.1	53503.2	52610.1
				Power consumed (Watt)	6.53201	6.44394	2.78556	6.56095
		socket 1 on CPU 14	Domain PKG	Energy consumed (Joules)	161113	150821.32	141478	109170
				Power consumed (Watt)	8.38807	8.491236	7.5378	13.6144
			Domain DRAM	Energy consumed (Joules)	65596.3	56598.6	60783.6	56896.6
				Power consumed (Watt)	7.41516	6.81065	5.10518	5.09551
	modelM5	socket 0 on CPU 0	Domain PKG	Energy consumed (Joules)	240362	183999	163089.7	123466
				Power consumed (Watt)	20.6033	12.1836	11.9177	13.4223
			Domain DRAM	Energy consumed (Joules)	58203.8	58683.8	56715.4	52487.7
				Power consumed (Watt)	6.51738	6.4346	2.87481	6.55632
		socket 1 on CPU 14	Domain PKG	Energy consumed (Joules)	161586	152032.3	149288	105247
				Power consumed (Watt)	18.0937	10.70527	8.08783	9.1466
			Domain DRAM	Energy consumed (Joules)	63394.3	62105.2	55442.65	56773.1
				Power consumed (Watt)	7.09859	6.80975	6.174502	7.09161
	modelM7	socket 0 on CPU 0	Domain PKG	Energy consumed (Joules)	260539	202474.9	179180	157815
				Power consumed (Watt)	12.8499	15.86742	20.4618	13.9261
			Domain DRAM	Energy consumed (Joules)	57238.5	57576.1	60041.5	50832.5
				Power consumed	6.53646	6.43779	2.96126	6.56585

				(Watt)				
		socket 1 on CPU 14	Domain PKG	Energy consumed (Joules)	198053	90530.4	159577	42406.4
				Power consumed (Watt)	9.76803	11.6935	18.2233	4.74161
			Domain DRAM	Energy consumed (Joules)	72050.72	62190.6	60905.8	54980.1
				Power consumed	8.3553	7.10198	6.8101	7.10159
userspace	modelM3	socket 0 on CPU 0	Domain PKG	Energy consumed (Joules)	69516.5	34987.8	230833	16637.5
				Power consumed (Watt)	3.2775	4.1379	11.7923	0.837802
			Domain DRAM	Energy consumed (Joules)	61256	54800.2	55940.5	40835.7
				Power consumed (Watt)	3.08463	6.48105	2.85776	4.192528
		socket 1 on CPU 14	Domain PKG	Energy consumed (Joules)	234079	174737	175346	135346
				Power consumed (Watt)	11.7873	8.06656	8.95765	7.96139
			Domain DRAM	Energy consumed (Joules)	13924.8	57694.6	2310.4	4701.44
				Power consumed (Watt)	0.656513	6.82336	0.118028	0.236747
	modelM5	socket 0 on CPU 0	Domain PKG	Energy consumed	78760.5	61500.2	132095	124634
				Power consumed (Watt)	3.67103	3.11214	8.57212	9.0825
			Domain DRAM	Energy consumed	5620.22	57939.4	50648.6	57638.7
				Power consumed\ (Watt)	1.261959	2.93195	1.98889	6.51266
		socket 1 on CPU 14	Domain PKG	Energy consumed (Joules)	96082.4	84203.0	81436.9	78918.7
				Power consumed (Watt)	7.3479	6.2476	5.28472	5.8564
			Domain DRAM	Energy consumed (Joules)	42151.1	3494.54	39919.7	25594.4
				Power consumed (Watt)	3.02251	2.176837	2.59053	1.176837

	modelM7	socket 0 on CPU 0	Domain PKG	Energy consumed (Joules)	74496.2	65923.7	34786.0	32916.6
				Power consumed (Watt)	3.49044	7.25505	12.3726	1.62408
			Domain DRAM	Energy consumed (Joules)	63845	58509.1	58742.7	49090.79
				Power consumed (Watt)	5.230043	6.43906	2.9323	3.15005
		socket 1 on CPU 14	Domain PKG	Energy consumed (Joules)	249448	248985	191192	46794.2
				Power consumed (Watt)	12.16876	12.2847	9.54386	5.14981
			Domain DRAM	Energy consumed (Joules)	61921.9	54600.78	44824.9	37499.0
				Power consumed (Watt)	6.8146	5.27258	4.6946	3.36999

This is the data when GPU is set to 210,Dram 12 & CPU 60 then we perform the experiment and obtained the energy & power consumtion data table.

The CNN model operation table of performance, conservative, ondemand, powersave, and userspace (800KHz) respectively and and the CPU power limit is 60Watt, the DRAM power limit is 12 Watt and GPU was set to 210 Watt where we used batch size=120 for training

Table Appendices 2

Energy-Delay Product (EDP) when GPU 210,Dram 12 & CPU 60

		Original	90% training & 10% inference	80% training & 20% inference	70% training & 30% inference
Governor	Model	Energy-Delay Product (EDP) (kilojoule-seconds)			
performance	modelM7	1817774.0115	1727828.7718	1744872.54222	1423.355040
	modelM5	2061763.98571	1850204.42228	1701917.72406	1479.1676319
	modelM3	2041401.0071	1895963.9204	1765717.052	1534.8437145
conservative	modelM7	1431072.002	439007.74386	206495.09307	169.80376438
	modelM5	1700454.9471	1183571.349	2178352.226	1956.1991787
	modelM3	1868153.150	1595208.0222	1274580.7734	1138.3743974
ondemand	modelM7	2418394.4503	752327.3716	206225.96367	211.710502435
	modelM5	4625282.250	1504998.02919	1154096.24625	1041.38119913
	modelM3	1755010.34664	671979.95788	3534947.9124	811.68468134
powersave	modelM7	5812755.3595	4241950.39245	3438464.200	3048.985800
	modelM5	5565630.1824	3742668.4593	3302696.89676	2443.4168332
	modelM3	5182919.0028	3450990.9486	2669049.2409	2584.2451111
userspace	modelM7	1504033.58028	1278273.72774	655716.100	601.65619812
	modelM5	1616204.84025	1216840.49719	2512539.3665	2364.120029
	modelM3	1446290.7825	705035.65902	4429685.270	316.12747375

Acknowledgement

I am sincerely grateful and wish to express my deepest gratitude to the Almighty for continuously blessing and guiding me throughout my journey. I am truly thankful for the unwavering support and guidance provided by my supervisor, Professor Zhengxiong Hou from the School of Computer Science of Northwestern Polytechnical University (NPU). His constant encouragement, diligent supervision, and invaluable knowledge have been instrumental in my growth. Professor Zhengxiong Hou has consistently shown me the importance of perseverance and unwavering faith, and I am immensely grateful for his generous allocation of time and unwavering support whenever I sought assistance.

I would also like to extend my heartfelt appreciation to my parents, family members, and friends for their patience, support, and encouragement throughout this period. Their unwavering belief in me has been a driving force, and I am truly grateful for their presence in my life.

Once again, I express my deepest gratitude to everyone who has played a part in my journey. Your support and encouragement have made a significant impact, and I am forever thankful."

Date:06/19/2023 Abid Ali

Graduation Project Summary

After working one semester of thesis work, this graduation thesis project is basically compl eted. The undergraduate graduation project is my last "big assignment" in the undergraduate period, which is full of unknowns and challenges. But in exploring each unknown and solvin g each difficulty, I have improved my ability. This graduation project is about “Energy Effici ency Analysis and Optimization of A Typical Deep Learning Application”.

Before I came into contact with the graduation project, my own understanding of high-perfor mance Deep Learning Application was negligible, and I was relatively unfamiliar with operat ing under the Linux system, especially the writing of shell scripts. I had very little previous c ontact. But fortunately, during the dissertation work, my supervisor, Mr. Hou Zhengxiong, ha s been paying attention to my thesis work, and often discusses problems with me, urges me to complete the dissertation work, and gives me great help. At the same time, during the graduat ion design process, I also constantly consulted materials and papers, and learned the relevant knowledge of Energy Efficiency and light-weight model for enery efficiency and some algori thms of machine learning. Although the whole process was busy, it was also very fulfilling.

This is the first time I have completed this kind of research-type problem. There are still some defects and insufficient consideration in the whole experimental process, but I finally c ompleted this final project. In this process, I have gained a lot practical knowledge. I was not familiar with the commands of the Linux system at the beginning, so I slowly learned a lot of Linux commands in the process of learning. Then I also learned the relevant content of the re gister in the process of searching for the implementation of Power Capping technology. Then a more important gain is to learn to write shell scripts, python scripts and nest shell scripts in python scripts.

Finally, during the research process, Mr. Hou sent me a lot of cutting-edge articles on en ergy efficiency optimization of deep learning applications.

The completion of the graduation project means that my undergraduate career is coming to an end. As time goes by, university life is the most important part of my life. I will never forget the guidance and care of teachers, the help among classmates, and the training of the school.

Finally, I would like to thank Mr. Hou Zhengxiong for his guidance and help, the school and college for their training and care, parents for their support and encouragement and other students for their help.