Resource Management for Graphics Processing Units in Cloud Computing

By

Abdulaziz Seraj A Alnori

Submitted in accordance with the requirements for the degree of Doctor of Philosophy

The University of Leeds

School of Computing

July 2021

Declaration

The candidate confirms that the work submitted is his/her own, except where work which has formed part of jointly authored publications has been included. The contribution of the candidate and the other authors to this work has been explicitly indicated below. The candidate confirms that appropriate credit has been given within the thesis where reference has been made to the work of others.

A. Alnori, K. Djemame. **"A Holistic Resource Management for Graphics Processing Units in Cloud Computing"**. Proceedings of the 33rd UK Performance Engineering Workshop, Newcastle, UK, December 2017. The material of this paper is owned by the candidate. This paper was reviewed by the co-author Karim Djemame. The content of this paper is included in chapter 4.

A. Alnori, K. Djemame. **"A Holistic Resource Management for Graphics Processing Units in Cloud Computing"**. Journal of Electronic Notes in Theoretical Computer Science. October 2018, 340, pp. 3-22. The material of this paper is owned by the candidate. This paper was reviewed by the co-author Karim Djemame. The content of this paper is included in chapter 4.

Acknowledgements

First of all, I thank God (Allah) for giving me the health, knowledge and patience to complete this thesis.

Special thanks go to my supervisor Prof. Karim Djemame for his guidance, advice and patience. This thesis would never be achievable without his support, knowledge and commitment.

I would like also to acknowledge the Distributed System and Services Research Group members for their valuable discussions and assistance. I would like to thank the IT centre members for their technical support.

I would like to thank my parents, Seraj and Munirah, for their love, support and faithful supplications. I want also to thank my brothers, Faisal and Marwan, and sisters, Fatimah and Siham for their unlimited support and encouragement during the study period.

I should thank from the bottom of my heart my beloved wife, Aminah, and sons, Abdullah and Marwan, for their unlimited support and patience at all the time.

I would like to thank my father-in-law Yahya and my mother-in-law Fatima for their support during the study period.

I would like to acknowledge my friends in Leeds for their encouragement during this journey.

Finally, I would like to thank King Abdulaziz University for granting me the scholarship to complete my postgraduate studies.

Abstract

The persistent development of Cloud computing attracts individuals and organisations to change their IT strategies. According to this development and the incremental demand for the adoption of Cloud computing, Cloud computing providers (CSPs) continuously update their infrastructure to fit this demand, which has led to the introduction of accelerator units to support Cloud computing users' applications requirements. Graphics Processing Units (GPUs) are well-known types of accelerators since they provide high-performance capabilities with low computational power consumption compared with traditional CPUs. The CSPs considerations in terms of adopting accelerators have resulted in an increase in hardware heterogeneity in the Cloud infrastructure and therefore the resource management techniques have challenged. The complexity of managing a heterogeneous Cloud infrastructure while maintaining the Quality of Service (QoS) is an important issue that needs addressing alongside minimising the infrastructure operational costs. A Cloud infrastructure consumes tremendous amounts of energy with a substantial impact on such operational cost. Thus, new energy-aware management techniques need to be developed to efficiently manage heterogeneous Cloud infrastructures.

This thesis introduces a systematic architecture that consists of several novel components to manage heterogeneous GPU resources made available through Virtual Machines (VMs) in a Cloud environment, considering performance and energy consumption as key factors. To fulfil the goals of the introduced architecture, different research contributions are accomplished. First, a power consumption model is introduced to identify the power consumption for heterogenous GPUs located in a Cloud-based system. Hybrid inputs are used to develop the power model consisting of hardware performance counters and resources utilisation (GPU and memory). Second, an agnostic energy model that aims to directly estimate energy consumption to be applicable for different types of GPUs is then developed. Third, this agnostic energy model is enhanced to estimate energy by reducing the cost of the data collection procedure. This enhancement of the energy model is created by developing a novel end-to-end energy framework aimed to predict the required resources for each GPU

application, and then pass the mentioned estimated resources to the heterogeneous GPU energy modeller. Finally, this thesis develops scheduling policies to reduce energy consumption and then to balance the trade-off between energy and performance whilst meeting the Quality of Service (QoS) in a Cloud computing environment. To achieve this goal, the proposed scheduling policies make use of the end-to-end GPU energy framework to obtain the application's energy consumption and execution time proactively.

The prediction models and the energy framework are evaluated on two different GPUs by a real Cloud computing testbed and show that they are capable to effectively predict power and energy consumption. Moreover, the evaluation of the proposed scheduling policies reveals that they can reduce energy consumption and also support the trade-off between energy saving and performance whilst maintaining the QoS requirements, therefore balancing energy consumption and applications' performance.

List of Abbreviations

ACIM	Average Computation Intensity Model
AI	Artificial Intelligence
AIC	Akaike Information Criterion
ALU	Arithmetic Logic Unit
Amazon EC2	Amazon Elastic Compute Cloud
AMG	Activity-Based Model
ANN	Artificial Neural Network
API	Application Programming Interface
ARIMA	Autoregressive Integrated Moving Average
ARMA	Autoregressive Moving Average Method
BP	Back-Propagations
CKE	Concurrent Kernel Execution
CO2	Carbon Dioxide
CPU	Central Processing Unit
CSP	Cloud Service Provider
CU	Compute Units
CUDA	Compute Unified Device Architecture
DES	Double Exponential Smoothing
DL	Deep Learning
DMA	Direct Access Memory
DNN	Deep Neural Network
DRAM	Device RAM
DVFS	Data Voltage and Frequency Scaling
EA	Energy Aware
EADA	Energy and Deadline Aware
ES	Exponential Smoothing
FBO	Frame Buffer Operation
FCFS	First Come First Serve
FFI	First-Fit Increasing
FFT	Fast Fourier Transforms

FIFO	First Input First Output
FPS	Frames Per Second
FR	Random Forest
GAM	Generalized Additive Models
GB	Gigabyte
GHG	Greenhouse Gas
GMM	Gaussian Mixture Model
GPGPU	General Purpose Graphics Processing Unit
GPR	Gaussian Process Regression
GPU	Graphics Processing Unit
GTT	Graphics Translation Table
HaaS	Hardware-as-a-Service
HCIS	Heterogeneous Cloud Infrastructure Scheduler
HDD	Hard Disk Drive HDD
HMM	Hidden Markov Modelling
HPC	High Performance Computing
IaaS	Infrastructure as a Service
ICT	Information and Communication Technology
IOMMU	Input Output Memory Management Unit
IPC	Instruction per Cycle
IT	Information Technology
J	Joules
KNN	K-Nearest Neighbours
KVM	Kernel-based Virtual Machine
LL	Least Loaded
LR	Linear Regression
LXC	Linux Containers Mean
MA	Moving Average
MAPE	Mean Average Percentage Error
MB	Megabyte
ML	Machine Learning
MLR	Multiple Linear Regression

NGC	NVIDIA GPU Cloud
NIST	National Institute of Standards and Technology
NVCC	NVIDIA CUDA Compiler
OpenCL	Open Computing Language
OS	Operating System
PaaS	Platform as a Service
PCA	Principal Components Analysis
PCI	Peripheral Component Interconnect
PE	Processing Element
PFA	Principle Feature Analysis
PI	Proportional Integral
PM	Physical Machine
PSM	Probability Slicing Model
QoS	Quality of Service
RAM	Random Access Memory
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
RMSE	Root Mean Square Error
ROP	Raster Operations Processor
RR	Round Robin
S	Seconds
SaaS	Software as a Service
SDK	Software Development Kit
SIMD	Single Instruction Multiple Data
SJF	Shortest Job First
SLA	Service Level Agreement
SLO	Service Level objective
SLR	Simple Linear Regression
SMO	Sequential Minimal Optimisation
SM	Streaming Multiprocessor
SP	Stream Processor
SPA	Streaming Processing Array

SSD	Solid State Drive
SVM	Support Vector Machine
TF	TensorFlow
TPC	Texture Processor Cluster
TWh	TeraWatt per Hour
VCPU	Virtual CPU
VIM	Virtual Infrastructure Manager
VM	Virtual Machine
VMM	Virtual Machine Monitor
W	Watt

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1 Research Motivation

Cloud computing has changed the shape of IT strategies over the last 15 years. Individuals and organisations have been attracted to change their IT strategies to Cloud computing usage since Cloud computing offers on-demand services at a competitive price. Cloud computing provides a variety of services to cover several users' demands including software developers, organisations, scientists and researchers.

Cloud computing usage has significantly grown, and its popularity has rapidly increased in recent years. For instance, in 2018, the expected number of Cloud computing users was approximately 3.6 Billion around the globe [1]. According to the Cisco report, in 2021, around 50% of the total workloads will be executed outside the organisations' servers, whether in Cloud data centres or other non-Cloud data centres [2]. Moreover, the global market of public Cloud computing size has significantly increased in recent years; the market size was approximately 58.6 Billion US Dollars in 2009, and it is expected to grow to 362.3 Billion US Dollars in 2022 [3]. Due to the rapid and continuous growth of Cloud computing, Cloud Service Providers (CSPs) need to expand and update their Cloud infrastructures to cover the Cloud users' demands. However, several challenges arise due to the expansion of the Cloud infrastructures to fit their users' requests.

To expand the Cloud infrastructure for covering the rapid increase of Cloud user requests, CSPs need to address the issue of hardware resources heterogeneity. Different types of processors, storage and other resources are used in Cloud computing infrastructures. For instance, Intel's CPUs execute workloads with ARM's CPUs, and Solid-State Drive (SDD) devices store data with the conventional Hard Disk Drive (HDD). Moreover, Cloud computing infrastructures have widely adopted accelerators units, such as Graphics Processing Units (GPUs), to boost the computing power capabilities in order to cover the rapid demand on highly intensive computing and data applications. Prominent Cloud computing providers

including Amazon [4], Microsoft Azure [5] and Google Cloud Platform [6] have enabled the usage of different types of GPUs for their customers.

Managing and provisioning these heterogeneous GPU hardware resources in Cloud computing infrastructures are challenging [7] because each hardware element has unique architectural characteristics, processing capabilities, power and energy consumption. As an example, NVIDIA [8] has a range of GPU architectures, such as Fermi [9] and Kepler [10], each with different features.

Because of the increasing demand for Cloud computing, CSPs must continuously maintain the Quality of Service (QoS) for end-users to avoid Service Level Agreement (SLA) violations and enhance their reputation. Maintaining the services' performance for end-users necessitates some operations in the Cloud infrastructure should be performed, such as Virtual Machines (VM) live migration. However, these operations lead to a very significant amount of energy consumption in the Cloud infrastructure and increase the operational cost.

Energy consumption is considered one of the main costs incurred by CSPs [11]. For instance, 42% of the total budget for Amazon EC2 was spent on energy consumption divided between 19% for direct power consumption and 23% for cooling operations [12]. The energy consumed by Information and Communication Technology (ICT) sectors has exponentially increased in recent years because of the proliferation of data centres [13]. The US data centres energy usage report [14] states that in 2014 the total estimated energy consumed by merely US data centres was 70 Billion kWh, which was approximately 1.8% of the total energy consumed in the US. The report also shows that the expected increase in energy consumption in US data centres will be 4% from 2014 to 2020. Globally, the estimated energy consumed by data centres is approximately 200 TeraWatt per Hour (TWh) [15]. According to [16], in 2030, data centres will take between 3 and 13 % of the global electricity compared to 1% in 2010. The estimated energy consumed by these data centres will be approximately 8000 TWh in the worst case scenario and around 1000 TWh in the best case scenario. Approximately 3000 TWh is expected in 2030.

Moreover, the increase in energy consumption harms the environment. ICT sectors produce 2% of the total CO2 emission in the globe while data centres take a

part of 0.3% [15]. Therefore, finding solutions for reducing energy consumed by Cloud computing infrastructures is essential. Efficient resource management [17] and considering energy-efficient hardware resources in Cloud computing infrastructures [18] can contribute to mitigate the energy consumed by Cloud data centres. Introducing GPUs in Cloud computing infrastructures can help to reduce energy consumption because GPUs are known as more energy efficient than CPUs [19]. Furthermore, energy models support CSPs to develop energy-aware policies to allocate several types of applications on heterogeneous Cloud computing physical resources. The Cloud computing physical infrastructures (i.e. CPUs, memories, storages and networks) have been intensively studied by the research community in terms of performance and energy consumption. However, GPU energy modelling and GPU resource management in Cloud computing need more consideration for QoS provision.

Some studies have considered GPU resources in Cloud computing focusing on the performance criterion, such as [20], [21] and [22]. The study performed in [23] has considered the energy consumption criterion without considering the QoS requirements. Moreover, the study conducted in [24] has considered performance and energy consumption for GPU resources with maintaining the QoS requirements, but without considering the heterogeneity of GPU architectures.

Therefore, this research aims to bridge the gap between effective mechanisms to manage heterogeneous GPU resources in Cloud computing environments and QoS provision. Besides combining the effective management mechanisms for heterogeneous GPUs resources in Cloud computing environments and QoS provision, this research will also consider performance and energy consumption as key factors.

1.2 Research Aim and Objectives

The aim of this research is to enhance the energy awareness of using GPU applications for general purpose running in Cloud computing environments by maintaining adequate performance and minimising energy consumption simultaneously. This will be performed using energy consumption modelling and

resource management. This research also aims to develop agnostic GPU energy models compatible with different GPU architectures in Cloud computing to reduce the complexity of heterogeneous GPU architectures. Then, the common inputs parameters will be used to develop an end-to-end GPU energy framework to eliminate the need for using profiling tools. This framework will be used for developing energy aware scheduling policies to reduce energy consumption in Cloud computing, and it will be used for developing energy and deadline aware scheduling policies to balance the trade-off between energy and performance whilst maintaining the QoS.

The outcomes of this research can be useful to develop energy aware GPU applications and efficiently support managing heterogeneous resources in Cloud computing in terms of energy and performance.

Thus, this research aims to address the following questions:

- **Q.1:** How can a prediction model estimate the power consumed by GPU applications running on heterogeneous Cloud computing infrastructures?
- **Q.2:** Following Q1, how can an agnostic model automatically estimate the energy consumed by GPU applications running on heterogeneous Cloud computing infrastructures?
- **Q.3:** How can a framework enhance the energy model that measures the energy consumed by GPU applications running on heterogeneous Cloud computing infrastructure without the need for collecting real data to train the model?
- **Q.4:** Based on the energy framework outcomes, how can an energy aware scheduling policy allocate the GPU applications efficiently to heterogeneous Cloud computing infrastructures? And what is the trade-off between performance and energy consumption whilst maintaining QoS requirements?
- **Q.5:** How can the trade-off between performance and energy consumption be mitigated whilst maintaining the QoS requirements?

To address the previous questions, the research objectives in this thesis are:

- **O.1:** *Exploring the existing GPU power and energy consumption models and GPU resource management techniques in Cloud computing*. Enhancing energy awareness in Cloud computing has been an important research topic to reduce its impacts. Thus, it is important to investigate and understand the current energy-aware techniques. This will enable the development of other solutions that can support the enhancement of energy efficiency in Cloud computing environments.
- **O.2:** *Understanding the architectural behavioural differences of heterogeneous GPU generations in terms of performance, power and energy consumption in a Cloud computing environment.* Analysing the heterogeneous resources in Cloud computing infrastructures is essential to understand the behaviour of the architectural components of Cloud resources for modelling goals. More specifically, this will require the identification of the main factors affecting the performance, power and energy consumption of GPUs when executing a specific workload.
- **O.3:** *Investigating Deep Learning (DL) techniques to develop power and energy models*. New models to predict the ***power*** consumed by GPU applications running on heterogeneous Cloud resources are developed. Furthermore, new ***energy*** models to automatically estimate the energy consumed by executing GPU applications on heterogeneous Cloud resources are developed.
- **O.4:** *Enhancing energy models by eliminating the need for collecting real data to train the models*. Using the built-in profiling tools to profile power consumption and other criteria like hardware performance counters incur a large overhead and are time-consuming. Therefore, a new framework to overcome the data collection overhead issue and predict energy consumption for heterogeneous GPUs in a Cloud computing environment is developed.
- **O.5:** *Developing an energy aware scheduling policy alongside the energy framework and **assessing** the trade-off between performance and energy whilst meeting QoS requirements*. Energy models can support the

development of efficient resource management techniques because energy models predict the application's energy consumption before it is deployed. Therefore, a new energy aware scheduling policy to allocate GPU applications on heterogeneous GPUs in a Cloud computing infrastructure is developed. The trade-off between performance and reducing energy consumption whilst maintaining the QoS requirements is then identified.

- **O.6:** ***Enhancing*** *the variation between energy and performance whilst meeting the QoS requirements* by *developing dynamic energy and deadline aware scheduling policy.* Considering energy consumption as a single purpose for scheduling resources produces significant variations on other factors such as performance. Moving from considering a single purpose for managing resources in Cloud computing to simultaneously consider different purposes, such as performance factors and energy consumption whilst meeting the QoS requirements will produce several benefits for CSPs. This will increase resource availability, enhance the reputation and reduce the operational cost for the CSPs. Moreover, considering performance and energy consumption for Cloud computing resource management will contribute to reducing energy consumption. Therefore, a scheduling policy that considers both performance and energy consumption whilst maintaining QoS requirements to allocate GPU applications on a heterogeneous Cloud computing infrastructure is developed.

1.3 Research Methodology

There are two main research methodology streams: quantitative and qualitative approaches [25]. The quantitative analysis paradigm is used to accomplish the research objectives in this thesis. The field of distributed systems contains three conventional research methods:

- **Direct experiments**

Direct experiments, such as in [26] and [27] are utilised for evaluating the research results in a real-world environment and validating the research hypotheses. Direct experiments are characterised by the simplicity of implementation and produce accurate dependable results. The shortage of

available resources, significant time and effort needed when repeating experiments are the disadvantages of this method. Additionally, using direct experiments for large scale Cloud computing environments is difficult and expensive [28].

- **Mathematical modelling**

Mathematical modelling, such as in [29] and [30], is the process of understanding, representing and solving a real-world problem by mathematical concepts. Fundamentally, the aim of mathematical modelling is the description of the behaviour of a system for predictive purposes. Mathematical modelling has been used in various fields, such as physics, biology, economics and computer science. Mathematical models results can be validated through direct experiments or simulations.

- **Simulation**

A simulation framework, such as in [31] and [23], is an application that sequentially mimics the real system. Simulation methods are widely adopted in the research community since they have several advantages, such as controllability, scalability, reduce time and cost compared with direct experiments [28]. However, the simulation method includes some randomness in the results. It produces less accuracy and reliability results compared to direct experiments results. Furthermore, the simulation method needs further validation by combining it with direct experiment or mathematical modelling methods [32].

To fulfil the research objectives of this thesis, direct experiments, mathematical modelling and simulation are adopted. In this research, the purpose of using mathematical modelling is to develop the power and energy prediction models for GPU applications in heterogeneous Cloud computing resources. Another purpose of using mathematical modelling is to develop the scheduling policies that allocate GPU applications to VMs. Direct experiments are used to collect data from the Cloud testbed, evaluate the developed models and showcase their applicability in Cloud computing environments. Simulation is used to evaluate the developed

scheduling policies. Figure 1.1 shows the conducted research methodology stages in this thesis based on the aforementioned research methods.

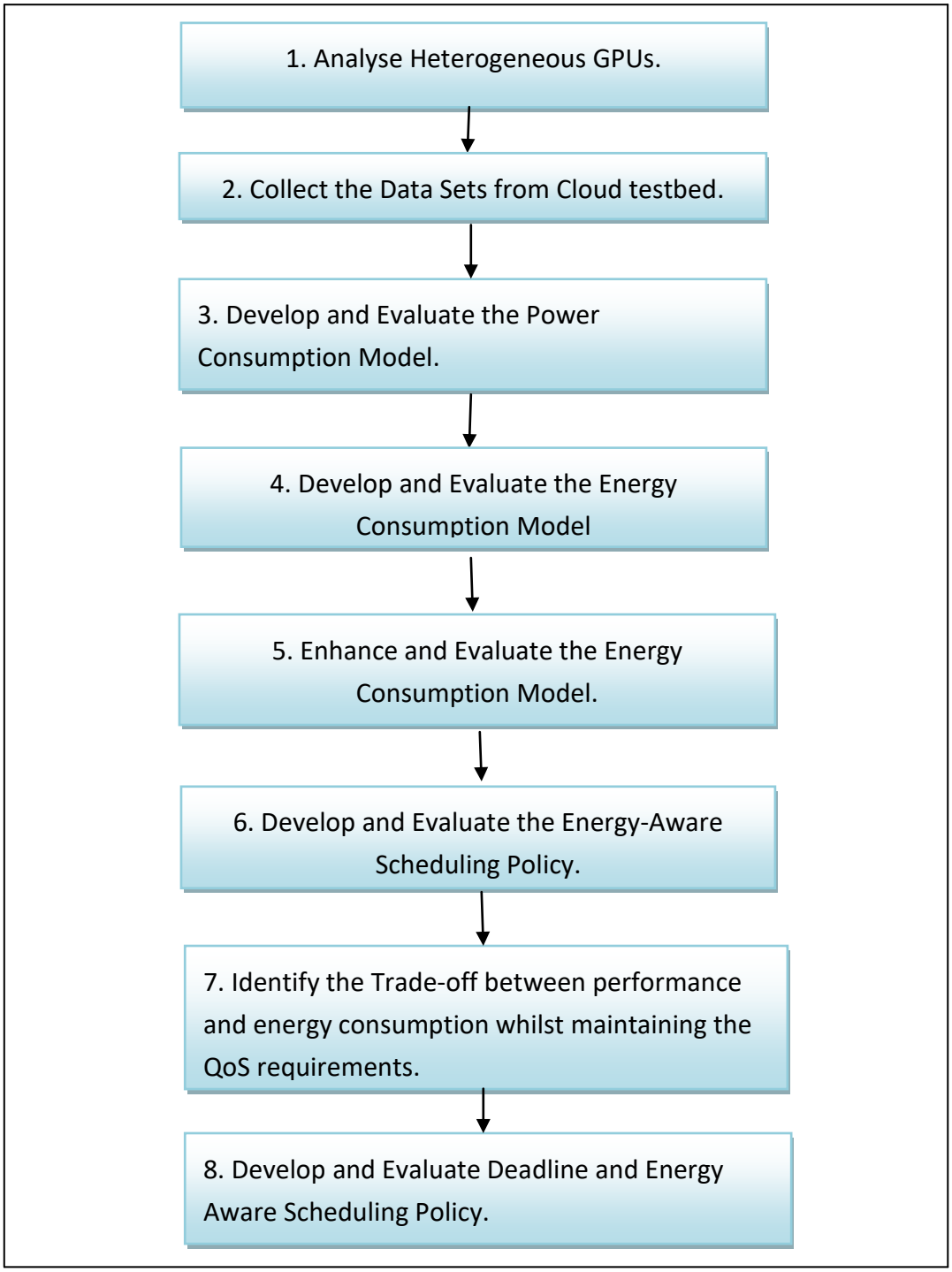


Figure 1.1: The Research Methodology Stages

1. Analyse Heterogeneous GPUs. Different experiments in a Cloud computing environment are performed to compare and understand the architectural behaviour of heterogeneous NVIDIA GPUs (Fermi and Kepler) in terms of performance, power and energy consumption.

2. Collect the Data Sets from a Cloud testbed. Data is collected by the execution of various GPU applications on heterogeneous GPUs in a Cloud computing environment to train power and energy consumption models. Part of the collected data (a testing set) is set aside to be used to evaluate the developed models.
3. Develop and Evaluate the *Power* Consumption Model Based on the collected data. The power consumption is developed using Deep Neural Networks (DNNs). The power consumption is also evaluated by comparing the actual power value with the predicted power value.
4. Develop and Evaluate the *Energy* Consumption Model. Like the power model, and based on the collected data, the energy model that automatically estimates the energy consumed by GPU applications running on heterogeneous Cloud computing resources using DNNs is developed. The energy consumption model is also evaluated by comparing the predicted energy values with the actual energy values in the testing set.
5. Enhance and Evaluate the Energy Consumption Model. The energy model is enhanced to automatically estimate energy consumption without the need to initially run different GPU applications on heterogeneous GPUs in a Cloud computing environment for collecting data to train the model. To perform this enhancement, an end-to-end framework that consists of various models that predict the required resources of executing GPU applications such as GPU utility, and temperature is developed. This framework's outputs are evaluated by comparing them with the actual experimental values.
6. Develop and Evaluate the Energy-Aware Scheduling Policy. The scheduling policy aims to reduce the energy consumption of executed GPU applications on a heterogeneous Cloud infrastructure. The proposed scheduling algorithm is evaluated through simulation and is compared with different scheduling algorithms in terms of energy and performance whilst maintaining the QoS requirements.
7. Identify the trade-off between performance and energy consumption whilst maintaining the QoS requirements. The outcomes of the energy aware policy in

terms of performance and energy consumption with the requested QoS are identified.

8. Develop and Evaluate *Energy and Deadline* Aware Scheduling Policy. The scheduling that aims to balance the trade-off between energy consumption and maintaining the QoS requirements is developed. This policy is then evaluated through simulation and compared with other scheduling algorithms in terms of energy and performance whilst maintaining the QoS requirements.

1.4 Main Contributions

The major contributions in this thesis are:

- *A systematic architecture for managing heterogeneous GPU resources in Cloud computing environments for general purpose usage.* The proposed architecture focuses on the application's deployment stage. In this architecture, the considered factors are performance and energy consumption. The proposed architecture consists of interacting components to achieve the objectives of this research, each one with a specific role.
- *A GPU **power** consumption model in Cloud computing*. This model aims to predict the power consumed by GPU applications running on a heterogeneous Cloud computing infrastructure. This model addresses the first research question **(Q.1)**.
- *A GPU **energy** consumption model in Cloud computing*. This model aims to automatically estimate the energy consumed by GPU applications executing on a heterogeneous Cloud computing infrastructure and addresses the second research question **(Q.2)**.
- *A GPU end-to-end energy prediction framework in Cloud computing.* This framework aims to directly estimate the energy consumed by running GPU applications on a heterogeneous Cloud infrastructure without performing experiments to collect the features of GPU applications to train the energy model. This framework contains various mathematical models for addressing the research question **(Q.3).**

- *An energy aware scheduling policy*. This policy communicates with the end-to-end framework to obtain the predicted energy and execution time of GPU applications. This policy aims to reduce energy consumption by scheduling GPU applications to the appropriate Virtual Machine (VM) in a heterogeneous Cloud computing infrastructure. The policy is evaluated through simulation. The trade-off between performance and energy consumption whilst maintaining the QoS requirements is then identified to address the research question **(Q4)**.
- *An energy and deadline aware scheduling policy*. This policy interacts with the end-to-end framework to obtain the predicted energy and execution time of GPU applications. This policy aims to *balance* the trade-off between performance and energy consumption when allocating GPU applications to different VMs on a heterogeneous Cloud infrastructure. Then, this policy is evaluated via simulation, and the policy's results are compared with different scheduling algorithms. This addresses the research question **(Q.5)**.

1.5 Thesis Overview

The remaining chapters in this thesis are structured as follows:

- **Chapter 2** introduces the background material in relation to this research including the Cloud computing concepts, Graphics Processing Units (GPUs) details and Machine Learning.
- **Chapter 3** presents the literature review and techniques of energy awareness for GPU resource management in Cloud computing. This chapter presents the GPU power modelling, GPU energy modelling and GPU scheduling techniques in Cloud computing.
- **Chapter 4** presents the proposed architecture for GPU resource management with its components and their interaction**.** Various experiments in a Cloud computing environment to analyse heterogeneous GPU architectures in terms of performance, power and energy consumption are performed in this chapter**.**

- **Chapter 5** discusses the power and energy models in details. This chapter begins with the details of developing the GPU *power* model. The GPU energy consumption model development details are then introduced. Finally, the GPU end-to-end energy framework details are presented.
- **Chapter 6** presents the developed scheduling algorithms, which include energy aware scheduling and the energy and deadline aware scheduling policies. The evaluation of these policies is then performed in this chapter.
- **Chapter** 7 summarises the performed work in this thesis as well as contributions. This chapter also discusses the limitations of this work and future work directions.

Chapter 2 Background

2.1 Overview

This chapter presents the essential background materials concerning Cloud computing, virtualisation, Graphics Processing Unit (GPU) and Machine Learning.

It starts by introducing the fundamental concepts of Cloud computing with detailed descriptions in Section 2.2. The aspects of virtualisation are discussed in Section 2.3. A brief description of power and energy consumption is shown in section 2.4. A description of GPU details is presented in Section 2.5. This is followed by a description of the Machine Learning (ML) concept in Section 2.6.

2.2 Cloud Computing

Due to the incremental demand of Cloud computing services, Cloud computing can be considered as one of the five fundamental utilities for the necessities of humans life after water services, gas, electricity services and telephony services [33]. The definition, service types and development types of Cloud computing will be reviewed.

There is no certain definition of Cloud computing [34]. However, the most appropriate definition for Cloud computing is the NIST definition:
"*Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage and applications) that can be rapidly provisioned and released with minimal management effort or service provider interaction*" p.2, [35].

Fundamentally, there are three service types in Cloud computing, these types are Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS), as shown in Figure 2.1. In addition, there are several sub-service types that push the boundaries of the three basic service types, such as Hardware-as-a-Service (HaaS) [36].

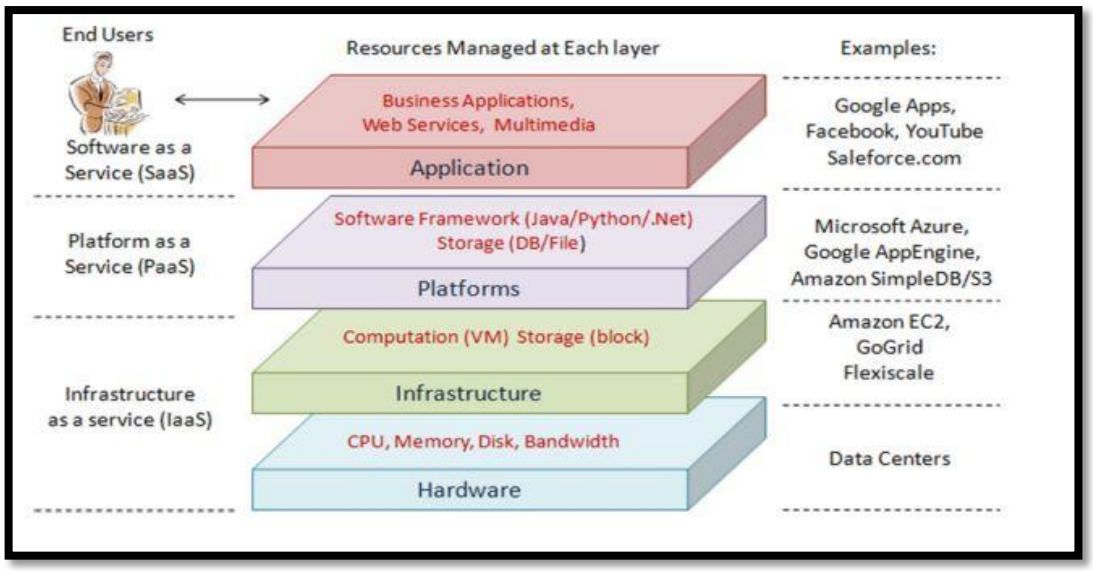


Figure 2.1: Service Types of Cloud Computing [37]

An advantage of Cloud computing is the ability to use it as a service that requires a set of deployment configurations. Each type of service deployment has a different level of security constraints, complexity and operational costs [7]. There are four standard deployment types in Cloud computing: Private, Public, Community and Hybrid.

2.3 Virtualisation

Virtualisation is defined as: *"Virtualisation is a technology that combines or divides computing resources to present one or many operating environments using methodologies like hardware and software partitioning or aggregation, partial or complete machine simulation and many others"* p.2, [38].

Virtualisation has been intensively adopted by Cloud computing as it is capable of abstract and isolate the underlying physical hardware functionalities, such as CPUs and GPUs. So, virtualisation is considered to be the backbone of the Cloud computing infrastructure. This allows the portability of high functionality levels and increases hardware resources, as well as sharing multiple operating systems. In this section, types of virtualisation and hypervisors will be highlighted.

There are two standard types of virtualisation: Full virtualization and Para-Virtualisation. Full virtualization completely emulates the hardware to be configurable with any type of operating systems. Para-Virtualisation partially emulates the hardware so the user can know some of this emulation.

The hypervisor, also called the Virtual Machine Monitor (VMM), is a complex software that is responsible for generating, executing, monitoring, isolating and migrating VMs. Examples of hypervisors are Xen [39], KVM [40] and VMware [41].

Moreover, Virtual Infrastructure Manager (VIM), also called the Cloud Operating System (Cloud OS), is a software platform that used to build virtual environments and manage Cloud infrastructures. VIM allows providers to provision the virtual and physical resources that are utilised by the end-users, such as instantiate, migrate and delete VMs. Open-source examples of VIMs are OpenNebula [42], CloudStack [43], OpenStack [44], and all of these VIMs share some characteristics and functionalities. VIMs provide private, public and hybrid Cloud computing deployment types. They support several types of hypervisors, such as Xen, KVM and VMware. Also, they provide VM management activities including life migration, load balancing and fault tolerance. VIMs support GPUs to be attached with VMs using pass-through virtualisation type.

Additionally, Containers technology is another type of virtualisation that produces a lightweight hardware abstraction and overhead better than the hypervisors virtualisation type [45], [46]. Hypervisors focus on the hardware level to create virtualised hardware and drivers for supporting isolated independent VMs from the physical hardware. Every VM contains an individual Operating System (OS) located on the top of the virtual hardware which creates a large size image. Thus, these large images produce considerable performance overhead. On the other hands, Containers create isolations on the OS level, which is applicable to ignore the hardware and device virtualisation [47]. Therefore, several containers can share a single OS kernel on the same host machine, which can create individual applications that have unique features like different network interfaces, different processing power and different storage capabilities.

With the appearance of Docker [48], a popular high-level containerisation platform, several Cloud computing providers have used containers technology such as Amazon. Linux Containers (LXC) [49] and Warden [50] are other examples of containers platforms. Furthermore, containers platforms support GPUs in Cloud computing environments, such as NVIDIA GPU Cloud (NGC) [51], NVIDIA Docker [52] and LXC can be used for GPUs in containers [53]. Kubernetes, an open-source

containers management tool, [54] can be used to manage a cluster of GPUs in containers environments.

2.4 Power and Energy

This thesis distinguishes between power and energy consumption as the following:

- Power consumption is the electrical usage when conducting a workload at a certain level of time; power consumption is measured in Watts (W).
- Whereas energy consumption is the amount of the average power consumption (P) within a period of time (T) until finishing executing the workload measured by Seconds (S); energy consumption (E) can be measured in Joules (J), and it is represented by the following equation:

  $E = P X T$ (1)

2.5 Graphics Processing Units (GPU)

Initially, Graphics Processing Units (GPUs) have been utilised in 3D graphics, animation and video games presented on the screen of the computer with high visibility. Due to their programmability and compatibility, GPUs have shifted from robust graphics' engines to effective computational engines. GPUs can directly execute parallel algorithms without sharing the processing capabilities of CPUs [55].

2.5.1 GPU Architecture

In 2006, NVIDIA released a new architecture termed Tesla to allow the use of GPUs for general-purpose usage, executing parallel applications developed by specific programming languages [56]. Since then, NVIDIA has been updating GPU architectures for general purpose usage, and these GPUs can be found and used in several system platforms, such as desktops, laptops, mobile phones and servers.

According to Flynn's taxonomy [57], the architecture of GPU is classified as Single Instruction Multiple Data (SIMD), which mean that many computation processors are conducting a single task with various data, to achieve high-performance executions and decrease the overheads.

Fundamentally, the GPU architecture consists of an expandable Streaming Processing Array (SPA) that has several Texture Processor Clusters (TPCs). Each TPC

has two Streaming Multiprocessors (SMs) that execute the exact instructions while each SM has its data. Each SM contains several processing cores named Stream Processors (SPs) in NVIDIA GPUs. Furthermore, the GPU architecture has several types of memories that have different sizes and different characteristics. Figure 2.2 shows the main components of the GPU architecture. GPUs interact with the remaining system components like the host CPU and the main memory through the Peripheral Component Interconnect (PCI) express bus. Starting from the higher level of the GPU architecture, the SPA is responsible to perform all the computation tasks in the GPU. All SPs communicate to other SPs within a single SM via on-chip memory called the shared memory. All SPs within a single TPC communicate with other SPs through the texture unit that contains on-chip the Level-1 (L1) cache memory. The expandable memory system in the GPU also contains Raster Operations Processors (ROPs) and an Exterior Device RAM (DRAM). ROPs are responsible to create colour and perform Frame Buffer Operations (FBOs) precisely on the memory. TPCs communicate with the DRAM through the interconnection network. Moreover, the interconnection network maps the memory reading requests of the texture memory from TPCs to DRAM and also maps reading data transactions from DRAM to TPCs through L2 cache memory.

The remainders of GPU architecture components are responsible to send the data inputs to the SPA for execution. Based on the data input type, The Input Assembler is responsible to gather the vertex work from the host and pass it to the Vertex Work Distribution unit. The Vertex Work Distribution unit will allocate the work to several TPCs for processing. The results will be temporally stored in on-chip buffers. The results then are rasterised and convert to pixel fragments. Pixel fragments are allocated to the suitable TPCs for processing by the Pixel Work Distribution block. Then, Pixel fragments are converted to shaded pixel fragments and pass to ROP units through the interconnection network channel to be processed. The compute threads are allocated to the SMs in the TPCs by the Compute Work Distribution unit. These SMs can work simultaneously, organised by an instruction scheduler to increase performance. Additionally, the GPU threads sizes are equally divided and share the same part of the memory as well as increasing the data exchange.

NVIDIA's GPU architecture for general purpose usage is continuously updating with different features and characteristics. Therefore, there are several types of NVIDIA's GPU architecture, including Tesla [56], Fermi [9], Kepler [10], Maxwell [58], Pascal [59] and recently Volta [60]. This work will consider Fermi and Kepler architectures.

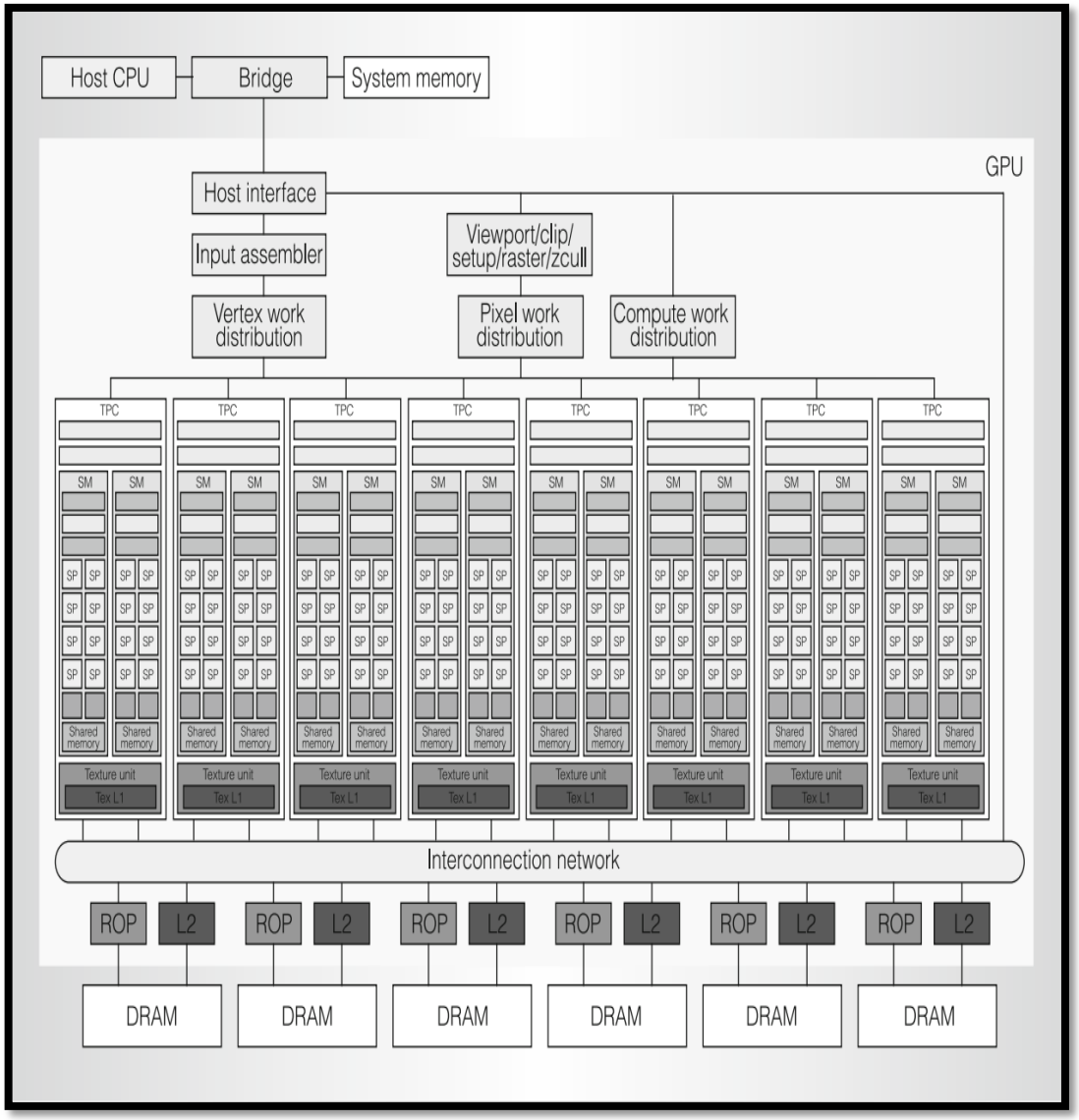


Figure 2.2:GPU Architecture [56]

2.5.2 CPU and GPU Comparison

For design purposes, CPUs are mainly designed to conduct sequential tasks with low latency and utilise their many cores to control logics, for instance, branch prediction. CPUs run their cores with high frequency and utilise a large number of cache memories to reduce the latency in the thread. Therefore, CPUs are suitable for executing applications that concern latency. On the other hand, GPUs are

mainly designed to conduct parallel tasks with high-performance executions and utilise their multiple cores for arithmetic logics, for instance, float point computations, and utilise a small number of cache memories, as shown in Figure 2.3. Therefore, GPUs are suitable for executing applications that concern throughput. Moreover, CPUs contain a limited number of threads, while GPUs have a large number of threads.

Even cores clock frequency of the GPU is lower than the CPU cores clock frequency, the influential impact of the parallel processing capability in the GPU defeats the issue of lower frequency.

Figure 2.3: Architectural Comparison between CPU and GPU [61]

2.5.3 General Purpose Graphics Unit (GPGPU)

Recently, since the programmability of GPUs, the GPU usage has shifted from its standard purpose, showing images and video games on computers' screens, to computational usage, and it shows a substantial leap in high-performance intensive parallelism execution. This new trend of GPU usage can be called General Purpose Graphics Processing Units (GPGPU), GPU Computing or Throughput Computing. Thus, this new trend of GPU usage has been utilised in a wide range of fields and applications which is producing a significant performance increasing, such as complex control systems [62], Database Management systems [63], file systems [64], autonomous vehicles [65], networking systems [66], information security systems [67] and scientific simulations like fluid dynamics [68].

The nature of the GPU architecture facilitates the spread of the usage for Deep Learning (DL) applications, such as image classification [69]. Training DL

applications is a time-consuming task with CPUs since these applications contain massive data that need to be trained. Therefore, GPUs dramatically reduce the DL training phase.

2.5.4 Programming Languages Support GPU Computing

There are two main types of programming languages supporting GPGPU applications that allow the developer to take the advantage of the parallelism in GPUs: Open Computing Language (OpenCL) [70], [71] and Compute Unified Device Architecture (CUDA) [72], [73]. Both of them are based on the programming language C and include APIs to interrelate with the GPUs and manage their parallel cores. In CUDA, the parallel system contains CPU or CPUs that are named as the host, and the GPU is named as the device for performing computation tasks. GPU is considered as a co-processor with the host (CPU). The hierarchy of GPU threads in CUDA contains two levels: the block and the grid, as shown in Figure 2.4. The block consists of a number of threads and each thread is identified by a thread ID. The grid contains several blocks of the same size and dimension. There are four basic processing steps to finalise executing the CUDA code. First, the data is copied from the main memory to the GPU's memory. Second, the CPU instructs the process to the GPU. Then, the GPU executes the process in parallel with each thread in the block. The final step is to copy the result from the GPU memory to the main memory.

In OpenCL, the CPU can be defined as the host or the device while the GPU is set just to be the device for executing the computation tasks. The concept of executing tasks in the device is by writing kernels and APIs managing these kernels. The kernel is a function that is developed to be executed in the OpenCL device and compiled during the runtime to allow host applications to benefit by using all computing devices in the system.

OpenCL is designed to support heterogeneous computing communications on uncertain CPUs and GPUs while CUDA is designed to merely support NVIDIA GPU products.

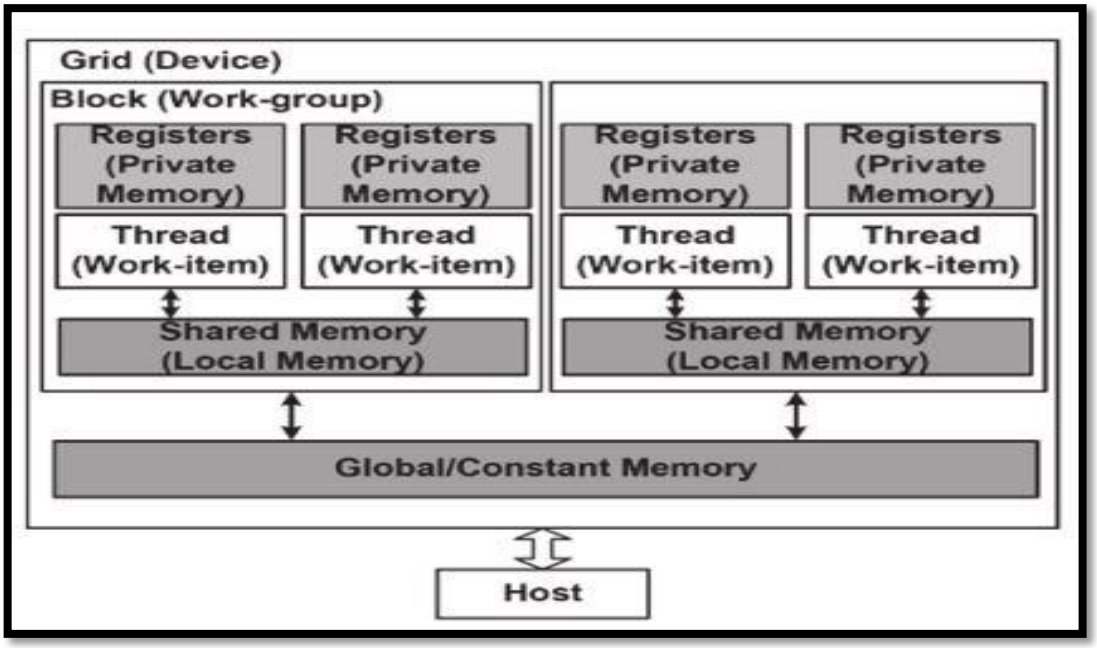


Figure 2.4: Thread and Memory Architecture in CUDA and OpenCL [74]

2.5.5 GPU Virtualisation

To increase GPU resource sharing in Cloud computing environments, several studies in GPU virtualisation have been performed. These studies can be classified into API Remoting, I/O Pass-through, Para-Virtualisation and Full Virtualisation [75].

API Remoting is a middleware framework that enables the end-user (i.e. the user who is running applications on VM) to access the physical GPU device remotely. This type of GPU virtualisation operates by dividing the GPU driver stack into front-end and back-end services. The front-end part contains the API libraries, such as CUDA, by using a wrapper library in the guest OS, that are sent to the back-end service through the network interface card or the shared memory. The back-end service will perform the API libraries call rather than the VM. API remoting approach supports a variety of API libraries like CUDA OpenCL, OpenGL and other libraries. In the API remoting approach, the guest OS can execute GPU applications without a physical GPU existence in the VM. In addition, API remoting virtualisation's performance fundamentally relies upon the application's type and the way of implementing the remoting technique. In terms of performance, Compute intensive applications are mostly better than Bandwidth and network-

based applications. To overcome this issue, high-speed network solutions should be implemented. Figure 2.5 shows the architecture of API remoting.

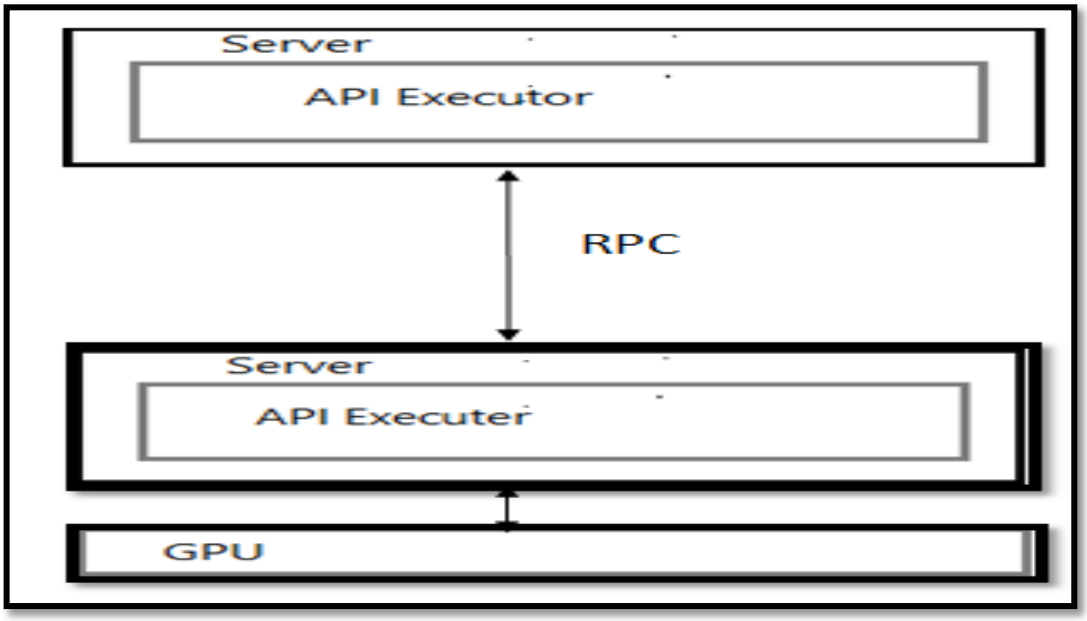


Figure 2.5: API Remoting Architecture [75]

Examples of this type of GPU virtualisation are GViM [20], vCUDA [76], GVirtuS [77] and rCUDA [78].

I/O Pass-through Virtualisation is another GPU virtualisation technique that enables the end-user direct access to the GPU. The physical GPU is dedicated to a single VM. Input Output Memory Management Units (IOMMUs) systems are the essential units of the I/O Pass-through virtualisation approach. There are two products that support IOMMU implementations Intel VT-d and AMD-Vi. IOMMU enables a Direct Access Memory (DMA) of the I/O bus to the main memory. IOMMU connects the virtual memory address to the physical memory address of the device to guarantee the device isolation and disable other requests. I/O Pass-through Virtualisation technique is a good choice for applications that need high performance or close to a native GPU performance. I/O Pass-through does not need to modify API stacks and drivers in the VM to access the physical GPU. GPU I/O pass-through approach is supported by many hypervisors, such as Citrix Xen server [79] and KVM. Figure 2.6 illustrates the architecture of the GPU I/O pass-through virtualisation approach. Cloud infrastructure management tools such as OpenNebula and OpenStack support this type of GPUs virtualisation.

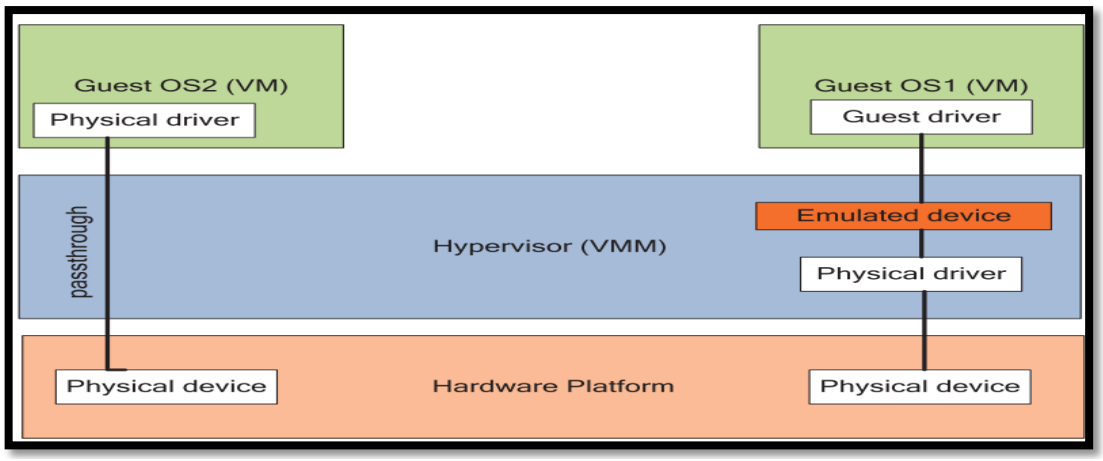


Figure 2.6: GPU I/O Pass-through Virtualisation Architecture [80]

Then, the Para-Virtualisation technique enables several VMs to share the physical GPU simultaneously through the hypervisor. Para-Virtualisation supports the low-level control to reduce the virtualisation overhead. However, it needs some modifications in the gust GPU drivers and OS system. Examples of GPU Para-Virtualisation are SVGA2 [81] and LoGV [82].

Finally, the Full Virtualisation technique provides resources that are fully emulating the hardware like GPUs through the hypervisor. It supports the GPU resource sharing among several VMs, and the guest OS does not need to be modified. Examples of GPU Full Virtualisation are gVirt [83] and GPUvm [75]. Figure 2.7 shows the architecture of Para-Virtualisation and full Virtualisation of GPU. However, the Full Virtualisation technique is not mature yet.

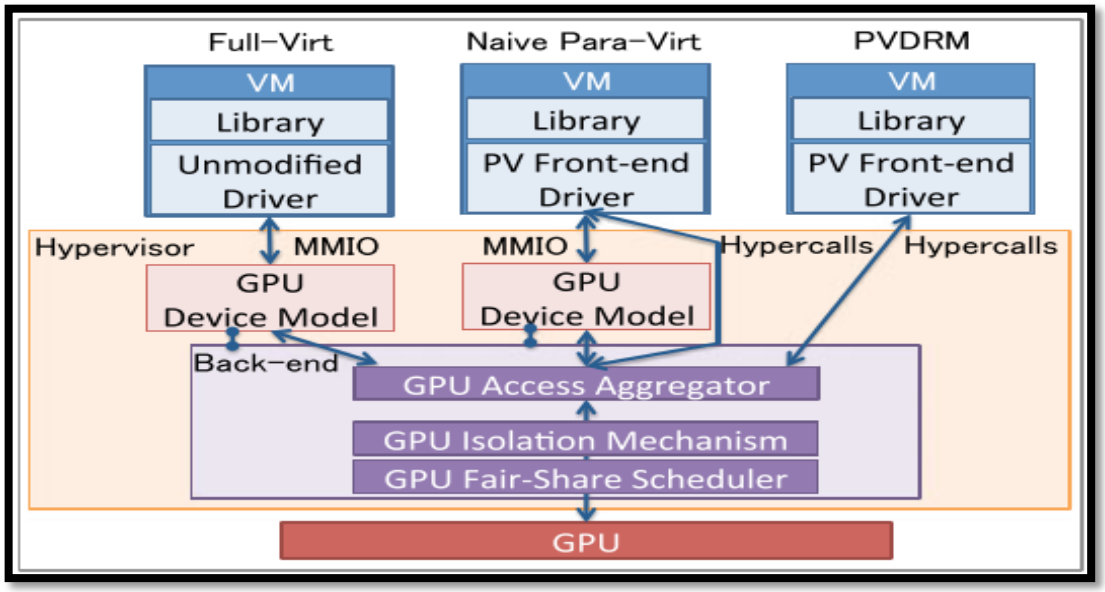


Figure 2.7:Full and Para GPU Virtualisation Architecture [75]

2.5.6 GPU in Cloud Computing

In the last decade, High-Performance Computing researchers and developers began to adopt a new processing system architecture that merges multi-core processors (CPUs) with accelerators, a well-known example for accelerators is the GPU devices, which have a massive number of cores to accelerate data-parallel applications and compute applications. The successful evidence of such computing architectures is that the top 5 supercomputers in the top 500 supercomputing list [84] use CPU/GPU systems.

Scalability is considered to be one of the essential advantages of Cloud computing. According to this scalability, hardware heterogeneity is introduced in the Cloud computing infrastructure. Consequently, the use of GPUs has been introduced in Cloud computing infrastructures. Therefore, this recent situation of GPUs existence in Cloud computing is changing the taxonomy of the Cloud data centres and the methods of managing these resources. Figure 2.8 shows the architecture of heterogeneous data centres.

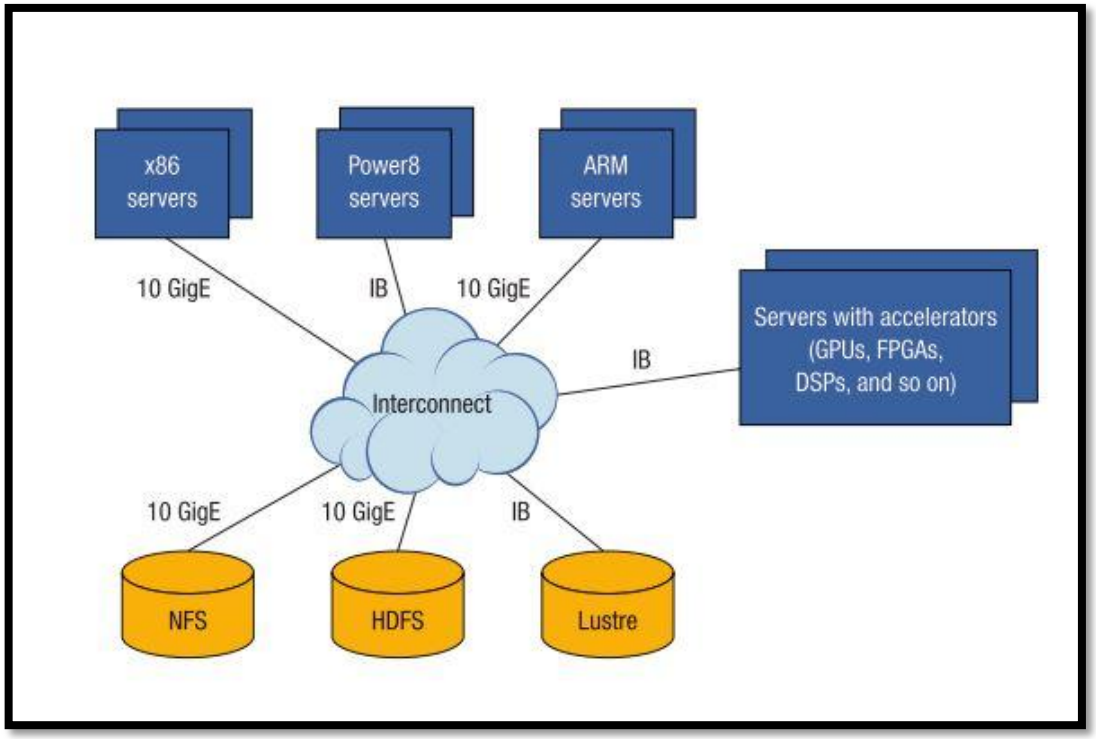


Figure 2.8: The Architecture of Heterogeneous Data Centres [85]

A number of Cloud computing providers like Amazon [86], Microsoft Azure [5], IBM Bluemix [87], NIMBIX [88], Alibaba [89] and Google Cloud Platform [6] have

adopted and integrated their Cloud data centres with GPUs to fit the customers' demands, increase profits and reduce the operational costs [90].

Amazon enables its users to access GPUs to conduct HPC applications through VMs. Amazon EC2 provide the P2 instance that contains VMs with GPU support. This instance can contain several NVIDIA Tesla K80 GPUs, up to 8 GPUs. The GPU virtualisation type that Amazon uses is the I/O pass-through technique [91]. Amazon has issued EC2 P3 instance which enables the user for using a VM supported by Volta V100 NVIDIA GPUs until 8 GPUs which provide high-performance capabilities for ML training algorithms and intensive simulations [4].

Additionally, Microsoft Azure has used GPUs in its infrastructure. The aim of intergrading GPUs is for supporting compute-intensive processing and high-end visualisation. Microsoft Azure uses the pass-through technique to virtualise GPUs via the Microsoft Hyper-V hypervisor; also, CUDA and OpenCL are supported in this platform. Microsoft Azure allows users to build their GPU clusters. Microsoft Azure offers N-Series types of VMs depending on the task type that the end-user wants to perform at different prices. Microsoft Azure is corroborating with NVIDIA on this platform.

IBM Bluemix is another Cloud provider that offers GPUs to accelerate applications and machine learning with different types of GPU configurations in Cloud computing environments. IBM Bluemix allows users to select virtual GPU servers or the bare metal GPU configuration options. In virtual GPU servers, the user can use merely NVIDIA Tesla P100. However, users can select different types of GPUs in bare metal options like NVIDIA Tesla V100, NVIDIA Tesla P100, NVIDIA Tesla M60 or NVIDIA Tesla K80.

NIMBIX also provides users to access GPUs through Cloud computing environments with different VM capabilities. NIMBIX enables to deal with GPUs by a variety of platforms, such as OpenCL, OpenGL and CUDA.

Moreover, Google Cloud Platform has introduced GPUs that can be attached with VMs instances to accelerate complex workloads and machine learning algorithms training. The GPU virtualisation type that Google Cloud Platform uses is the I/O pass-through technique.

2.6 Machine Learning

Machine Learning (ML) is defined as the ability to learn a computer application through experience to automatically obtain a certain task, and it enhances the performance of achieving the task by experience [92]. ML is a part of the Artificial Intelligence (AI) field. The outcomes of ML include models that represent learning algorithms usage. These models are aimed to make predictions from data. ML algorithms are known with lightweight computational capacities and good results when they learn efficiently. ML algorithms can extract data patterns, and ML is a data-driven approach. Therefore, ML algorithms differ from conventional analytical models since analytical models need to be developed manually by humans. Unlike the analytical models, ML models are created by historical data features automatically to predict the future and can be used for supporting decisions. ML can be utilised in several applications, such as computer vision, objects recognition, robotics and web searching. ML algorithms are divided into Supervised Learning, Unsupervised Learning and Reinforcement Learning (RL).

2.6.1 Supervised Learning

Supervised Learning is a well-known type in ML types. Supervised learning is illustrated by a set of pairs that consists of inputs and outputs targets $(\vec{\imath}, o)$. Inputs and outputs pairs are *labelled* to create a *training data* set containing *training examples*. These *training examples* have several inputs objects and targeted outputs elements. Each input object is represented as a vector $\vec{\imath}$. Supervised learning algorithms aim to learn the model by mapping inputs vectors to the targeted output element o. The targeted output element o value can be continuous or discrete; therefore the Supervised Learning is divided into regression and classification outcomes. Several supervised Learning algorithms are available, such as Linear Regression (LR), Non-Linear Regression, Support Vector Machine (SVM) and decision tree.

The single regression model is a type of regression techniques that aims to estimate the relationship between two variables. These variables are called the dependent variable y (outcome variable) and the independent variable x (predictor

variable). The model aims to estimate y informed by x. For example, the single linear regression can be formulated by the following equation:

$y = \alpha + \beta x$ (2)

Where α is the intercept, β is the slope value and x the given variable.

2.6.2 Unsupervised Learning

In contrast to Supervised Learning, the training data set containing labelled data is not considered in Unsupervised Learning [93], and the data is used without any structure. In Unsupervised Learning algorithms, the data patterns features are explored without human configurations. Unsupervised Learning algorithms do not differentiate the input data; they deal with data features similarly. Unsupervised Learning algorithms aim to discover the similarities in the data inputs and gather similar data features together in groups. Unsupervised Learning algorithms are divided into principal components and clustering. Clustering algorithms are used to gather several data features that contain common characteristics into groups, datasets or segments. Principal Components Analysis (PCA) is an algorithm that aims to alter the possible correlated variables located in a certain set to linear uncorrelated variables set by using the orthogonal basis method. Principal Components refer to the number of non-linear correlated variables. PCA aims also to reduce the set dimensions to minimise the complexity. Examples of Unsupervised Learning are K-means and Gaussian Mixture Model (GMM).

2.6.3 Reinforcement Learning

The behaviour of Reinforcement Learning (RL) is located between Supervised Learning and Unsupervised Learning [93]. RL deals with software agents to train them for taking actions. The interaction of RL algorithms is located in dynamic environments and these algorithms must have a defined goal to learn, such as how to play a game. The model is learnt through continuous feedback, or called rewards, to maximise the model's outputs. Examples of RL algorithms are Q-learning and Monte Carlo.

2.6.4 Deep Learning

Machine Learning (ML) algorithms work efficiently to deal with several problems; however, these ML algorithms have not dealt successfully with challenging problems in AI like objects recognition and speech recognition since these AI algorithms have not provided good performance. Moreover, increasing the data dimensionality has become a challenging task for ML to deal with [94]. Therefore, these previous challenges of ML have become a motivation to develop Deep Learning (DL) algorithms. DL is a new part of the Machine Learning field that utilises many layers to explore more features from raw data [95]. Deep Learning structure relies on Artificial Neural Networks (ANNs) with several hidden layers, it can be called Deep Neural Networks (DNNs). The major difference between ANNs and DNNs is the number of hidden layers. DNNs have several hidden layer numbers, while ANNs have a limited number of hidden layers. Similar to ML, learning algorithms of Deep Learning can be divided into Supervised Learning, Unsupervised Learning and Reinforcement Learning. The structure of Deep Learning types can be classified into Deep Feed-forward Networks, Convolutional Networks and Deep Recurrent Networks that have been used in a variety of applications, such as image processing, objects recognition, autonomic driving, Natural Language Processing, speech processing, robotics and many others.

Deep Neural Network (DNN), Multilayer Perceptron (MLP) is a type of Deep Learning (DL) techniques consisting of several processing hidden layers that are greater than one hidden layer [96]. DNN aims to reduce human engineering intervention and increase automation and abstraction levels. Therefore, DNN is capable to extract features from data automatically through a large number of hidden layers [97]. DNN has a better performance than other machine learning algorithms in terms of predicting objects [97]. DNN shows a substantial improvement in different complex fields.

The structure of the DNN consists of three main layers. These layers are the input layer, hidden layers and the output layer. Each layer consists of several neurons. These neurons are connected through a weighted link called synapse or edge. Each neuron contains an activation function that has a non-linear threshold. The activation function defines the output of each neuron.

2.7 Summary

This chapter has presented some information regarding the main streams of this research. Firstly, it has informed the definition of Cloud computing, its service types and the Cloud computing deployment types. Secondly, the chapter has illustrated the types of virtualisation technology, Hypervisors with their types, examples of hypervisors, providing some information about Virtual Infrastructure Managers and providing containers as well. It then has introduced the GPU architecture, general-purpose usage in GPUs, a brief comparison between CPU and GPU, programming languages for supporting GPU computing, GPU virtualisation and GPU in Cloud computing. This chapter also has presented the difference between power consumption and energy consumption. Finally, the chapter has presented Machine Learning and its types including Supervised Learning, Unsupervised Learning and Reinforcement Learning; details about Deep Learning have been also provided.

The next chapter will review the related studies of this research.

Chapter 3 Energy-Aware GPU Resource Management in Cloud Computing

3.1 Overview

Cloud data centres consist of a large number of resources with different characteristics. However, these resources consume a significant amount of energy. To enhance energy efficiency in Cloud data centres, substantial investment has been spent in research and industry since the consequences of increasing energy consumption are harmful to the environment, economy and performance [98]. For instance, the energy cost of running a Cloud data centre is estimated to double every five years [99]. Therefore, this cost is considered to be one of the major expenditures in Cloud data centres for Cloud computing providers [100]. Moreover, the incremental energy consumed by Cloud data centres leads to a significant negative impact on the environment. For example, Information and Communication Technology (ICT) industries produce 2.3% of detrimental gases from the total global emission in 2020 [101]. Also, their sectors constitute 2% of the total global Greenhouse Gas (GHG) emission [102]. To solve this issue, more funds and efforts have been made to enhance energy efficiency in ICT systems, to reduce this emission to 1.97% by 2030 [103].

From the performance perspective, It is challenging to simultaneously obtain high performance with energy saving [98]. However, most of this effort in this concern is to find an adequate and balanced trade-off between performance and energy consumption, which also remains difficult to achieve with different end-users requirements. Thus, a choice has sometimes to be made between high performance or energy reduction selections for energy cost reduction.

This chapter aims to review the studies related to energy-aware GPU management techniques. GPU power modelling techniques and studies in Cloud and non-Cloud computing environments are discussed in section 3.2. GPU energy modelling techniques are reviewed in section 3.3. Workload and energy prediction techniques are reviewed in section 3.4. Moreover, section 3.5 reviews the GPU scheduling studies in Cloud computing. Finally, an overall discussion is proposed in section 3.6.

3.2 GPU Power Consumption Modelling

This section covers the studies conducted for GPU power modelling in conventional (non-virtualised) and Cloud-based systems.

3.2.1 GPU Power Consumption Modelling in non-Virtualised Environments

3.2.1.1 Statistical and Machine Learning Models

Several studies have been performed for GPU power modelling in conventional (non-cloud) systems with different techniques. For example, Ma et al. [104] considered the first research for GPU power modelling by proposing a statistical model using the Support Vector Machine (SVM) technique to estimate GPU power consumption. The model was applied to an old generation of NVIDIA GPUs (GeForce 8800GT). The model was relied upon the signal data of the run time GPU workload to train the model. The model produced moderate results. However, this approach has not investigated the impact of performance counters on the power consumption model.

Nagasaka et al. [105] proposed a statistical model to estimate GPU power consumption using the linear regression technique. The features considered for training the model were performance counters of several GPU applications. The used hardware performance counters were selected without justifications. A total of 49 different applications were considered to develop the model. The power model was applied on NVIDIA GeForce 285 GTX, and the evaluation results showed a power estimation accuracy of 4.7%. But, the authors have not considered heterogeneous GPU architectures when developing the power model.

Ghosh et al. [106] explored the difference between linear and non-linear statistical models in terms of accuracy for estimating GPU power consumption using performance counters. These models were classified into parametric and non-parametric techniques. The authors considered for comparison were the Simple Linear Regression, Linear Regression with Transformation and the Generalized Additive Models (GAM) for smoothing the independent variables. The simple linear regression was improved to Linear Regression with Transformation because of the relationship between the dependent variables and the independent variable, which

was non-linear. The Akaike Information Criterion (AIC) was used to select the predictors. To estimate power consumption, the selected predictors were instructions issued numbers, DRAM read transactions, L2 Cache misses transactions and the number of operations per second. These models were evaluated on a small set that had merely five scientific applications on a cluster containing four GPUs belong to a single homogeneous architecture (Nvidia Tesla M2050). The average percentage error of power models of the Simple Linear Regression, Linear Regression with Transformation and the GAM were 7%, 3.62% and 3.5%, respectively. Hence, the non-linear models performed better than the linear model. Nevertheless, this approach has not considered heterogeneous GPU architectures.

Song et al. [107] developed another GPU power modelling for modern GPU architectures using traditional Artificial Neural Networks (ANNs) as a type of machine learning techniques. The authors used hardware performance counters on different GPU applications to train the ANN model. The selected performance counters were simple and did not need to understand the architecture's low-level components. The performance counters represented the micro-architecture components, such as instruction pipeline and memory access features. The ANN model used the sigmoid function as an activation function. The authors compared their ANN model with the Multiple Linear Regression (MLR) technique, and their model showed better accuracy. The experimental results on NVIDIA Fermi C2075 revealed that the power model was accurate with 2.1%. Also, for robust results, the model was evaluated on a cluster contains homogeneous Fermi M2090s GPUs. However, the authors have evaluated the power model only on a single type of GPU architecture.

Additionally, Wang and Chen developed statistical models to estimate the power consumption of GPU applications [108]. The model aimed to study power consumption from the software side. The application's components were analysed by using programme slicing to estimate the GPU application's power consumption. Programme slicing is a technique to study the programme's parts as slices, and it is between the low level and the functional level. The authors mainly focused on computation intensity and the active number of Stream Multiprocessors (SMs)

which had a great impact on power consumption. The authors developed two models for GPU applications power consumption estimation. The first model was the Average Computation Intensity Model (ACIM) used for sparseness-branch applications power estimation. The second model was the Probability Slicing Model (PSM) used for denseness-branch applications power estimation. The experimental results performed on GeForce GTX280 NVIDIA GPU showed that the average errors of the models were less than 6%. However, this approach has not considered performance counters for developing the power model.

An agnostic power model for heterogeneous GPU architectures was also proposed by Abe et al. [109]. The selected GPU generations were NVIDIA Kepler (GTX 680), Fermi (GTX 460, GTX 480) and Tesla (GTX 285). The authors used the MLR technique to predict power consumption for heterogeneous GPUs. The authors used hardware performance counters to develop the model, and they grouped these counters into two categories. These categories were GPU core frequency counters and GPU memory counters. The accuracy error in this agnostic model was quite high. The average percentage error in GTX 285, GTX 460, GTX 480 and GTX 680 were 15.0, 14.0, 18.2 and 23.5, respectively. However, the authors have merely investigated the linear models to build the power model which is not appropriate with modern GPU generations.

Moreover, an online GPU power estimation model was proposed by Adhinarayanan et al. [26]. The model aimed to estimate the power consumption of GPU applications instantaneously on the runtime. The model relied upon performance counters that carefully selected. The authors first created a procedure that relied on correlation analysis to find the performance counters that had a great impact on GPU power consumption, and they considered temperature for power estimation as well. Then, the authors used several types of statistical regression techniques for power consumption estimation. These statistical regression models were Simple Linear Regression (SLR), Multiple Linear Regression (MLR), MLR with Interaction, Basic Quadratic Regression and Quadratic Regression with Interaction. The authors grouped the power model into application-independent and application-dependent models. For the application-independent model, the data set was divided into training and testing set, and for the

application-dependent model, the authors reduced the model cost by reducing the data set number. The model was evaluated on two heterogeneous GPU generations (Fermi C2075 and Kepler K20c NVIDIA GPUs). The best mean error results for application-independent regression models were 4.49% and 6.14 on Fermi C2075 and Kepler K20c, respectively. The best mean error results for application-dependent regression models were 1.02 and 0.88 on Fermi C2075 and Kepler K20c, respectively. However, the power models have produced overheads in some applications at runtime.

Zhao and Chen introduced a GPU electric running current, measured by Ampere, model by analysing the CUDA applications source codes [110]. The authors analysed three source code characteristics: computing operations, programme branch and memory access operations. Then, these characteristics were used to create the model as inputs. The authors developed two different models. The first model was developed using Multiple Linear Regression (MLR), and the second model was developed by Back-Propagations (BP) neural networks. The models were evaluated on Kepler GTX 670, Kepler GTX 680 and Maxwell GTX970 NVIDIA GPUs. The results showed that the models were capable to predict running current with an average of less than 6% and less than 10% for single application accuracy. BP model performed better than MLR with a small difference. But, this approach has not directly reflected the GPU power consumption (W) since the proposed model has predicted the GPU electric running current measured by Ampere (A).

Braun et al. [111] proposed an accurate, simple and portable GPU power model by using Random Forest (FR) algorithm. Independent source code instructions, such as memory instructions and kernels configurations like blocks number were used as model's inputs. Several CUDA applications types of GPU architectures were used to evaluate the model. The used GPUs were Kepler K20, Turing GTX 1650, Pascal Titan Xp, Pascal P100 and Volta V 100. The experimental results showed the range of Mean Average Percentage Error (MAPE) within 1.8% and 2.9% for all used GPUs. However, the model has not performed well in applications with a short execution time.

3.2.1.2 Low-Level Analytical Models

Hong and Kim [29] presented an analytical model for estimating GPU power consumption dealing with intensive details of GPU microarchitecture. The authors considered GPU instruction pipeline components, memory types operations and the temperature as well. The model was an integration of power and performance for a given GPU architecture to estimate the optimal number of active GPU processing cores when executing a specific GPU application that leads to a better performance with saving more energy. Therefore, the estimated required cores could be only activated and the others switched off using power gating, circuit gate to remove the power leakage. The average error of the GPU power model on NVIDIA GTX280 GPU was 8.94% and the model can be obtained up to 22.09% energy saving when running GPU applications by predicting the required cores and switching off the unnecessary cores. However, the model has been evaluated by a single GPU, and the heterogeneity of GPU architectures has been neglected.

Furthermore, an analytical model that aimed to estimate GPU power consumption was developed by Kasichayanula et al. [112]. This used model called Activity-Based Model for GPU (AMG) to estimate the power consumption of each GPU micro-architectural component and the total GPU power consumption by analysing micro-benchmarks like global memory, shared memory and floating-point unit. Also, the authors considered the GPU temperature for GPU power modelling. The authors evaluated their model on a power meter called Kill-A-Watt on NVIDIA Fermi C2075 GPU by running linear algebra applications, and the accuracy percentage was 10%. Similar to [29], heterogeneous GPU architectures have not considered evaluating the power model.

3.2.1.3 Simulation-based Models

Leng et al. introduced a configurable GPU power model called *GPUWattch* [31]. This model was an extension of *McPAT* CPU power model [113]. The model had three main characteristics: configurable, able to calculate power consumption in the GPU cycle-level and strongly validated. The model had a deep investigation of the GPU micro-architectural components. These micro-architectural components were used as input models to calculate the dynamic GPU power consumption and combined with the static GPU power. The model was validated on two different

GPU generations. These GPUs were G80 Quadro FX5600 and Fermi GTX 480. The percentage errors for Quadro FX5600 and GTX 480 were 13.4% and 9.9%, respectively. In terms of configuration, the model was intergraded with *GPGPU-Sim* that considered a GPU cycle-level simulation tool [114]. This simulation tool is used to simulate old GPU generations; however, it is not suitable for simulating modern GPU architectures [26].

Similarly, *GPUSimPow* that aimed to predict and simulate GPU power consumption by running a GPU application workload was introduced by Lucas et al. [115]. *GPUSimPow* was capable to estimate the GPU micro-architectural components like gate leakage, chip area and peak dynamic power consumption. These features could help the computer architect to evaluate the hardware design in terms of power consumption in the early stage and software developer to evaluate power consumed by GPU applications before running these applications in a real GPU device. The power model was able to model the static and dynamic power consumption. *GPUSimPow* could support CUDA and OpenCL GPU applications. The analytical power model was based on *McPAT* CPU power model, and *GPGPU-Sim* microarchitectural design was used for GPU cycle-level performance design. The power model was evaluated on Tesla GT240 and Fermi GTX580 NVIDIA real GPUs. The average error between real and simulated GPU power consumption for Tesla GT240 GPU was 11.7% and 10.8% for Fermi GTX580 GPU. Nevertheless, *GPUSimPow* does not support Kepler GPUs yet.

3.2.1.4 Power Modelling with DVFS Technique

Other recent studies aim to model GPU power consumption with the integration of several Dynamic Voltage and Scaling Scaling (DVFS) configurations. For example, Dutta et al. [116] presented a power consumption model cooperating with different DVFS settings using several machine learning techniques and applying a combination of machine learning techniques to reduce the prediction error. The authors used hardware performance counters as the model's inputs parameters. The used machine learning techniques were Sequential Minimal Optimisation (SMO) regression, Simple Linear Regression (SLR), ZeroR, random forest, neural network, K- Nearest Neighbours (KNN), bagging and decision tree. Experimental results conducted on Pascal Quadro P4000 NVIDIA GPU showed SMO regression

technique performed well to reduce the average error by 4.5% while random forest had the most stable results on several frequency configurations. The combination of SLR, SMO regression and decision tree techniques led to reduce the average error to 3.5% and reduce the highest error from 15% to 11%.

Similar studies can be found in [117], [118] and [119]. However, GPU power modelling with the integration of the DVFS power conservation technique is out of the scope of this research.

3.2.2 GPU Power Modelling in Cloud Computing

Siavashi and Momtazpour presented a GPU simulation framework tool for Cloud computing environments called *GPUCloudSim* [23]. The simulator was implemented on a well-known Cloud computing simulator called *CloudSim* [28]. *GPUCloudSim* has some features that deal with GPUs in Cloud computing environments, such as power modelling and VM scheduling. The authors used NVIDIA GRID GPU virtualisation technology [120] to deal with GPUs in Cloud computing environments. In terms of power modelling, the authors developed a GPU power using a linear regression model based on the linear relationship between GPU frequency and the SM utilisation following [121]. The authors evaluated their work on NVIDIA GRID K1 and K2 [122], and both of them are based on NVIDIA Kepler architecture. However, the authors have merely considered the linear relationship which cannot efficiently represent the complexity of new GPU architectures [107].

3.3 GPU Energy Consumption Modelling

3.3.1. Indirect Energy Modelling

This sub-section will review GPU energy modelling that performed in many stages to estimate energy consumption.

3.3.1.1 Analytical models

Hong and Kim [29] presented analytical models on NVIDIA GTX280 GPU to predict power, performance and temperature of GPU workloads separately using intensive details of GPU microarchitecture, such as GPU instruction pipeline components, memory types operations. By using these models, the predicted energy consumption can be computed. The predicted energy consumption showed energy

saving with an average of 10% when the power gating technique has been activated in the GPU architecture in the runtime. But, the model has been evaluated by a single GPU.

Song et al. [10] developed a model to predict energy consumption in several steps. First, the authors developed a power consumption model using a neural network using performance counters. Second, an analytical model was developed to estimate the performance (execution time) of the GPU application based on GPU memory (global, shared and texture) access types and some GPU computation instruction types like Arithmetic Logic Unit (ALU) and active warp number. The error of the performance model was within 6.7%. Then, power and performance models were used to estimate the energy consumption on homogeneous GPUs, but the authors have not used heterogeneous GPUs for evaluating the energy model.

Similarly, Xie et al. [123] proposed an analytical GPU power model based on the native instructions level. The authors divided the native instructions level of the GPU into computational and memory types parts. The energy consumption of each micro-architecture component was measured. Then, the total energy of each GPU application was computed when the instructions number of the application was calculated. The model was evaluated on NVIDIA Fermi C2050 GPU. The model's results showed that the percentage error did not exceed 15%. However, Kepler based architecture GPUs have not considered for evaluating the model.

Additionally, Boughzala et al. [124] presented a simple and lightweight energy consumption prediction model for CUDA applications using a simulation tool. Instead of adopting hardware performance counters and low-level architectural details, the model merely considered the block number per Stream Multiprocessor (SM) as an essential input parameter because it had a strong impact on energy consumption. To model energy consumption, a performance model was developed using simple linear regression. Then, the energy consumed by a single block was calculated including static and dynamic energy consumption. Finally, the energy consumption of all number of blocks in SMs was calculated integrating with single linear regression. The model was conducted on a simulation tool called *SimGrid* [125] to simulate two heterogeneous NVIDIA GPUs architectures and compare the

simulated energy values with the real measured energy values. The selected GPUs were Kepler K20Xm and Tesla M2075. The model achieved the highest average error of 6.86% energy on the Tesla M2075 GPU. On Kepler K20Xm GPU, the model achieved the highest average error of 8.33% in energy. However, the model is not applicable for all CUDA applications, and the energy model is not able to estimate the energy consumption of CUDA applications that have a small number of blocks. Also, simulation tools produce an unavoidable overhead compared to real experiments.

3.3.1.2 Statistical and Machine Learning Models

Abe et al. [109] developed and evaluated separately power and performance models for several GPUs generations using MLR statistical technique. The results of the model showed that the errors of the models were less than 20% in all used GPUs. But, the authors have used linear models, as mentioned in section 3.2.1.1.

Wu et al. [27] presented GPU power and performance models using machine learning algorithms on real GPU measurement. The model was based on performance counters and focusing on OpenCL applications. The data that aimed to train the neural networks model was collected on several GPU configurations. These configurations variations relied upon the GPU Compute Units (CU) and the GPU memory frequencies. The neural networks aimed to estimate the appropriate scaling curve of the application. Then, the scaling curve was employed to predict the application's power and performance in several GPU configurations. The model was evaluated on Radeon HD 7970 AMD GPU. The results showed the average errors of power and performance were within 10% and 15%, respectively. However, this work merely considers AMD GPU and OpenCL applications. Also, the authors consider a homogeneous AMD GPU. This thesis only emphasizes NVIDIA GPU architectures and CUDA applications. Moreover, the heterogeneity of GPU architectures has not investigated in this work.

3.3.2 Direct Energy Models

Ghosh et al. [106] used parametric and non-parametric techniques to estimate energy consumption. The selected statistical models were the Simple Linear Regression, Linear Regression with Transformation and the Generalized Additive

Models (GAM) for smoothing the independent variables. The Akaike Information Criterion (AIC) was applied to select the predictors. The selected predictors to estimate energy consumption were instruction issued, DRAM read transactions, L2 catch miss transactions and execution time. These models were evaluated on a small set that had merely five scientific applications on a cluster containing four homogeneous GPUs (Nvidia Tesla M2050). The average percentage error of energy consumption models of the Simple Linear Regression, Linear Regression with Transformation and the Generalized Additive Models (GAM) were 23.49%, 12.5% and 4%, respectively. The results have shown that the non-linear models performed better than the linear model.

3.3.3 Cloud computing environments

Makaratzis et al. developed an analytical energy model based on the binary model to estimate GPU energy consumption [126]. This model was based on a previous CPU model that existed in [127] to integrate both CPU and GPU energy consumption. The authors used three GPU intensive applications. These applications were N-body, Matrix multiplications and Fast Fourier Transforms (FFT) convolution applications. The model was evaluated on three different GPUs. The used GPUs were Maxwell GeForce GTX 980, Kepler Tesla K20C and Pascal Tesla P100 NVIDIA GPUs. The range of the percentage error was between 0.53% and 5.91%. The authors claimed that the developed model is suitable to be integrated for Cloud environments simulators. However, there is no indication that the work has been performed in Cloud computing environments.

3.4 Workload and Energy Prediction

3.4.1 GPU performance Modelling

Hong and Kim [128] introduced an analytical model to estimate the CUDA GPU applications running on GPUs. The performance model was used to determine the applications performance bottleneck. Mainly, the model's process to estimate the execution time was first estimating the parallel memory number requests (can be called the memory wrap parallelism) by focusing on the active memory and threads bandwidth number. Then, the cost of the memory request is estimated relied upon

the degree of parallel memory number requests, and finally, the execution time was calculated. The experimental results showed the average error of estimating GPU applications execution time was 13.3%.

Baghsorkhi et al. [129] also introduced an analytical model, called *string* model, to estimate the GPU application's performance running on GPU by analysing each application and its behaviour on the GPU architecture features. The authors used CUDA applications and an NVIDIA GPU architecture to develop the model. The model aimed to estimate the warp average execution time. The model was evaluated on GeForce 8800 NVIDIA GPU.

Similarly, Kothapalli et al. [130] proposed an analytical model aimed to predict the performance, more precisely execution time, of CUDA GPU applications on NVIDIA GPUs. Several architectural features were considered to develop the model, such as memory hierarchy, scheduling and pipeline procedure. The authors analysed the effect of memory access types on performance. The model was applied on matrix multiplication, histogram and list ranking applications. The model could be used to obtain the performance of the CUDA application. The model was evaluated on GTX 280 NVIDIA GPU.

Jia et al. [131] introduced a framework called *STARGAZER* aimed to automate GPU design space exploration and estimate the GPU performance based on regression statistical modelling. *STARGAZER* used random and sparse samples values that represent the GPU design and simulated this design. Then, these samples were used to develop the performance estimator for exploring the most important architectural factors and how the random samples interact with each other. *STARGAZER* was evaluated using the *GPGPU-Sim* simulation tool on the architecture of Quadro FX 5800 NVIDIA GPU. The results showed that the framework was capable to estimate the execution time of 11 GPU applications with an average error of less than 1.1%.

Abe et al. [109] presented a GPU performance model using MLR statistical technique reliant on all existing performance counters in every GPU architecture. The experimental results performed on four different GPUs showed an average error range between 33.5% and 67.9%.

Karami et al. [132] introduced an analysing framework to find the performance bottlenecks and discovered the relationship among these bottlenecks. To develop the framework, the authors also considered the MLP technique to predict the performance of OpenCL applications running on NVIDIA GPU. Moreover, the authors used principal component analysis (PCA) to extract the most important hardware performance counters and used them as the model's input parameters. The model was evaluated on Fermi GTX 480 NVIDIA GPU, and the model achieved an average accuracy of 99% with less than 3% of individual minimum error and 10.3% of the maximum error for every application.

Wu et al. [27] introduced a performance model using machine learning techniques cooperating with memory frequency configurations. K-means was used to cluster the patterns of several GPU applications with different scaling behaviour. Then, the relationship between the applications patterns and hardware performance counters was found for the training ANNs model, and the model was able to catch some nonlinearity levels. The experimental results showed that the model was able to estimate the performance of GPU applications with a 15% average error conducted on AMD GPU.

Konstantinidis and Cotronis [133] presented a performance model to estimate the execution time of several types of GPU applications, such as memory and compute applications. The presented model was a quantitative approach integrated with the quadrant-split model for visual representation purpose to show the performance limitation on different GPU types. The model was experimentally evaluated on four different GPUs that contain different performance capabilities. These GPUs were NVIDIA GTX 480, GTX 660, Tesla K20c and Tesla S2050. The results showed that the model was able to estimate the execution time of the GPU applications with an average error of 10.1% and 25.8% for the worst estimation result.

Shafiabadi et al. [134] presented a model to predict the performance of different GPU applications based on a non-linear regression model (polynomial type). To develop the model efficiently, several hardware and software parameters were considered, such as the number of the compute unit and branch instruction numbers. The authors used the Instruction per Cycle (IPC) to measure the performance of the GPU. The authors considered OpenCL applications and AMD

GPU architecture type to develop the model. The experimental results based on *Multi2Sim* simulator [135] showed that the model was able to predict the performance of the GPU with an average error of 7%.

Furthermore, Boughzala et al. [124] proposed a performance prediction model for CUDA applications using a simulation tool. The model merely considered the block number per Stream Multiprocessor (SM) as an essential input parameter since it had a strong relationship with the application's performance. The performance model was developed using the simple linear regression technique. The model was performed on a framework called *SimGrid* tool to simulate two heterogeneous NVIDIA GPUs architectures and compare the simulated values with the real measured values. The selected GPUs were Kepler K20Xm and Tesla M2075. The model achieved the highest average error of 6.69% on Tesla M2075. On Kepler K20Xm, the model achieved the highest average error of 6.54%.

Arafa et al. [30] presented a scalable framework called *PPT-GPU* that aimed to predict the performance of GPU applications running on heterogeneous GPU architectures. *PPT-GPU* was a combination of analytical and low-level cycle simulation modelling approaches. *PPT-GPU* depended on the source code analysis approach to model the performance whiteout running the application to reduce the profiling delay. To enable increasing the data size in the simulator, the static code analysis technique was used by dividing the whole workload into several small sub-workloads and dispatched them to available existing Stream-Multiprocessors (SMs). The model was evaluated on two heterogeneous NVIDIA GPUs (Maxwell GeForce GTX TITAN X and Kepler K40m). Additionally, the framework was compared with another simulation tool called *GPGPU-Sim*, and the comparison merely conducted on Maxwell GeForce GTX TITAN X GPU. Experimental results indicated that the error of *PPT-GPU* was within 10% in comparison with the real GPUs measurements. Moreover, it performed better than *GPGPU-Sim* in terms of accuracy and speed; on average, it was 160 times faster than *GPGPU-Sim*.

A similar approach was performed by Wang et al. [136] to develop a light and fast framework to predict the performance of OpenCL applications running on NVIDIA GPUs. The authors used a combination of source code analysis and trace simulation modelling approaches. The framework used static analysis to develop the

performance model without the need for profiling or executing the application. Then, the model's simulation process was performed. The framework was evaluated on four NVIDIA GPUs that belong to two generations of NVIDIA GPUs (Kepler and Maxwell); these GPUs were Kepler Quadro K600, Kepler GeForce GTX645, Maxwell Quadro K620 and Maxwell GeForce 940M. The model achieved an average of 17.04% on all four GPUs. The framework was also faster than *GPGPU-Sim* with an average speed-up of 164.39 times. The average error of the simulation results was 17.38%.

Another study was conducted by Wang and Chu [137] aimed to estimate the execution time of GPU applications using an analytical model under several memory and core frequency scaling configurations. The model also was aimed to determine the performance bottleneck on the GPU applications when several frequency configurations were applied. The analytical model considers some GPU hardware specifications and GPU applications performance counters as input parameters and eliminated the demand for analysing the source code to estimate the execution time. The experimental results showed that the proposed model could estimate the performance of the GPU applications on different frequency configurations in substantial accuracy with an average error of 3.85%, 8.6%, 8.82% and 8.83% applied on 20 GPU applications validated on various GPUs including Maxwell GTX 980, Pascal GTX 1080Ti, Pascal Tesla P100 and Volta Tesla V100, respectively.

Furthermore, Braun et al. [111] proposed an accurate, simple and portable GPU performance model using Random Forest (FR) algorithm. Independent source code instructions, such as memory instructions and kernels configurations like blocks number were used as the model's inputs. Several CUDA applications types and GPU architectures were used to evaluate the model. The model achieved the range of average error between 13.45% and 44.56% 15.59%, 13.27%, and 11.61% on all used NVIDIA GPUs: K20, GTX 1650, Titan Xp, P100 and V100, respectively. However, the model does not perform well in applications with a short execution time.

3.4.2 Workload Prediction

Energy consumption in Cloud computing environments is affected by several factors, such as the workload. To predict the energy consumption of applications and VMs in Cloud computing whether in deploy time or run time, the effect factors like the physical resources features and power consumption with variable workload utilisations need to be understood and considered. Thus, workload and resource prediction can be beneficial to predict the future of energy consumption in Cloud computing environments to minimise energy cost and reduce SLA violations [138].

Caron et al. proposed a resource utilisation prediction algorithm [139]. This algorithm was relied upon a set of historical data to recognise the pattern of the actual usage; this technique was called pattern matching. Experimental results conducted in a grid-like environment showed the prediction algorithm worked with a median error rate from 09% to 4.08%, and it could work better when increasing the size of historical data.

Gong et al. proposed PRedictive Elastic reSource Scaling (*PRESS*) framework to predict applications workload for allocating enough resources (CPU, network usage and memory) [140]. *PRESS* identified the repeating patterns called signatures by employing signal processing methods for prediction purpose. *PRESS* used a statistical state-driven technique to find the short terms pattern when no signatures were explored, and also utilised the discrete Markov chain for predicting the resource demand in the nearly future. *PRESS* aimed to avoid both over-estimation and under-estimation. However, it prioritised under-estimation since it causes more Service Level objectives (SLO) violations. *PRESS* prototype was implemented on Xen hypervisor. The results showed that *PRESS* achieved high prediction accuracy with both over or underestimation compared to other frameworks.

A resource allocation mechanism based on a workload prediction model for reducing the Cloud service provider and satisfying QoS requirement was developed by Roy et al.[141]. The authors apply the Autoregressive Moving Average Method (ARMA) technique for workload prediction. The results showed the benefit of the model to the end-user and the service provider in terms of reducing the operational cost.

Khan et al. developed a model to estimate repeatable VMs workload patterns over the time [142]. Hidden Markov Modelling (HMM) was the used technique to estimate the workload, and the authors were merely considered CPU utilisation workload. The authors used the co-clustering approach to assemble VMs that perform repeatable workload patterns in groups. Then, the authors used HMM technique to identify the correlation in the grouped VMs and predicted workload patterns variance. The model aimed to enhance the efficiency of Cloud computing resources provisioning.

Islam et al. utilised Artificial Neural Networks (ANNs) and Linear Regression (LR) to develop a module to predict the required resources to run applications within VMs and dynamic provisioning [143]. To train and test the module, a data set generated from the TPC-W benchmark [144] was used in the Amazon EC2 Cloud environment. The model was evaluated by statistical metrics, such as Mean Absolute Percentage Error (MAPE). Statistical metrics showed the prediction accuracy of the ANNs model was greater than the prediction accuracy of the LR model.

A prediction model was proposed by Zhang et al. [145] aimed to fit QoS requirements using Service Workload Patterns (SWP). The model was based on combining workload mining with the traditional filtering approach to enhance the accuracy of the model. The authors used a top-down approach and QoS-SWP matrix technique to allow the dynamic configuration and maintain the QoS requirements in Cloud computing environments. To predict SWPs, resource utilisation log files were used that contained CPU utilisation, storage data and network throughput data as well. To reduce the noise that came from the nature of Cloud computing, log smoothing and fuzzy logic approaches were added.

Yang et al. [146] applied the Linear Regression (LR) model to predict the VM workload. Then, this predicted workload was used to auto-scale the VM resources in Cloud computing environments at different resources level. The experimental results showed that the proposed auto-scaling approach can maintain the SLA requirements with low-cost scaling.

Moreover, other studies have been performed in [147], [148], [149] and [150] to predict the future required workload of VMs in Cloud computing environments using the Autoregressive Integrated Moving Average (ARIMA) technique. For

instance, Calheiros et al. [150] introduced a workload prediction model relied upon (ARIMA) method for dynamic provision purpose and meeting the QoS criteria, such as decreasing the rejection rate and increase the response time. The authors provided feedback to update the model. The accuracy of the model was evaluated by tracing the real request of the web server coming from the Wikimedia Foundation website[1]. The simulation results exhibited that the accuracy of the model could reach up to 91%, and thereby this led to efficient resource usage with reduced response time for end-users.

Huang et al. [151] presented a framework, called Prediction-based Dynamic Resource Scheduling (*PDRS*), to automate resource allocation in Cloud computing environments. ARIMA model was applied to predict the required CPU and memory resources for the VM reliant on historical data. Then, the predicted resources were used for scheduling VMs and VM live procedure to reduce the active PM numbers. Experimental results showed that *PDRS* was capable to understand automatic elastic resource management with adequate effect on the SLA requirements.

Zhang et al. [152] proposed a framework, called *PRMRAP*, for proactive resource management in Cloud computing environments. The authors predicted the required resource of the VM, and the considered resources were CPU, memory and network traffics. The authors used the ARIMA model to predict the aforementioned resources reliant on historical data. Then, these predicted resources were used to maintain the performance from any sudden changes that could be occurred by taking a proactive action (auto vertical or horizontal scaling actions).

Moreover, Farahnakian et al. [153] presented a VM consolidation framework, named *UP-VMC*, by considering the current and future resource utilisations to reduce the energy consumption and minimised the SLA violations in Cloud data centres. The authors used regression-based models (Linear and K-Nearest Neighbour) to predict the potential CPU and memory utilisations in both VM and PM levels. The experimental results showed that the presented approach *UP-VMC*

[1] http://www.wikimedia.org

performed well to reduce energy consumption, VM migrations number and SLA violations better than other approaches, such as heuristic and meta-heuristic.

In terms of performance prediction, Wagle et al. [154] presented a model to predict a set of performance patterns for several Cloud computing providers. In this work, the selected performance metrics patterns types were uptime, downtime, response time, latency, outage frequency and throughout. ARIMA and Exponential Smoothing (ETS) techniques were used to predict the potential Cloud performance patterns with a month of observed data on daily intervals. The results showed that the model could be applied for whether long or short terms of time.

Additionally, Ibrahim [155] proposed a framework, named *PRESEnCE*, to automatically evaluate the QoS of the deployed SaaS in Cloud computing. The author applied Gaussian Process Regression (GPR) model to predict the performance of the service, such as throughput and latency provided by the Cloud Service Providers (CSP).

3.4.3 Energy Modelling based on workload prediction

Also, workload and the required resources predictions participate to reduce the energy consumption in Cloud computing environments. Therefore, Wang et al. [156] presented a heuristic framework based on the greedy algorithm for VM resource allocation including deployment and live migration aiming to increase the resources usage (CPU and Memory) and reduce the energy consumption of the system. The authors applied the quadratic exponential smoothing technique for the workload prediction goal. Experimental results displayed a substantial enhancement in power saving, scalability and the balancing of workload in comparison with the single-purpose approach that merely relied upon just CPU utilisation.

Farahnakian et al. [157] presented a method, called *LiRCUP*, to maintain SLA and reduce the power consumption cost by taking the advantage of the workload prediction. The authors used the LR technique to predict CPU utilisation in the short term based on historical data. Then, the predicted CPU workload used for the live migration procedure to find the over-loaded and the under-loaded PM. Therefore, the VMs that hosted in the over-loaded host will be migrated to the

under-loaded host and then the over-loaded host will be in the sleeping mode to preserve the power consumption. The experimental results showed that *LiRCUP* can mitigate energy consumption and reduce SLA violations.

Subirats and Guitart [158] introduced a method to predict the energy consumption in different levels in Cloud computing environments (VMs, PMs and service levels) to support possible future management actions like VM deployment, VM placement or VM cancellation as well. To do so, a CPU utilisation estimation model was presented to support energy consumption. Moving Average (MA), Exponential Smoothing (ES), Linear Regression (LR), and Double Exponential Smoothing (DES) techniques were used to estimate CPU utilisation and to find the best prediction value among these techniques. The model merely considered the linear relation between CPU workload and power consumption. Their work was evaluated on heterogeneous infrastructure (heterogeneous CPU).

An end to end power consumption prediction model for the system during the run time level was produced by Kistowski et al. [159]. The model was based on the collected data of power consumption during running the application evaluated in heterogeneous environments (heterogeneous CPUs). Another approach was introduced to predict power consumption without collecting the power data in a single server depending on performance counters and load intensity. The authors applied regression techniques like Gradient Tree Boosting to predict power consumption. The authors claimed that their model can be applied with any type of distributed systems. The results showed that the percentage error for the model used in heterogeneous environments was 2.21%, and the remaining model had a 1.04% percentage error. However, these models are not applied in Cloud computing environments. Moreover, the authors do not consider the power consumption of GPU architectures.

Moreover, Aldossary et al. [160] proposed a novel architecture that consisted of several interactive components to support energy awareness in Cloud computing. First, the required resources (VCPUs, memory capacity, disk storage size and network) for running a workload in a VM are predicted using the ARIMA model. The authors considered static and periodic workload types for VM resources prediction. Second, the authors predicted the PM workload by using the predicted

VM resources hosted on this PM. Then, PM Power consumption was predicted reliant on the relationship between predicted PM and PM power consumption. After predicting the PM power consumption, the VM energy consumption was calculated, and then, the estimated VM cost was evaluated reliant on the predicted VM power and resources. Finally, energy-aware pricing models were presented to increase the pricing options strategies in Cloud computing environments. The experimental results on heterogeneous Cloud infrastructure showed that the pricing models based on energy awareness techniques could add economic values for Cloud providers.

3.5 GPU Scheduling in Cloud Computing Environments

The ubiquity of heterogeneous systems with GPUs stimulates the usage of high-performance computing applications, such as scientific, data-intensive and large data applications by accelerating the performance with efficient energy consumption. This usage has been moved to Cloud computing platforms by hosting these services. The applications' designers and developers can take advantages of these platforms like the accessibility by using these applications without the need for containing in-house servers and bear the costs of their maintenance. This section will cover the scheduling techniques of GPUs in Cloud computing environments.

3.5.1 Performance consideration

3.5.1.1 Integrated GPUs

Tian et al. [83] introduced *gVirt*, a framework that enables to fully virtualise GPU resources in Cloud computing environments using the mediated pass-through technique. To maintain the performance, a coarse grain QoS scheduling policy was proposed. The scheduling policy aimed to manage the performance of the context change in the GPU. Moreover, it was aimed to reduce the time delay of the submitted commands by the VM for execution within a certain time and enable fairness among VM commands. the Gang scheduling algorithm was used to schedule dependent engines simultaneously to eliminate the issue of synchronisation of the shared data. The experimental results conducted on 2D and 3D applications on integrated GPU demonstrated that the framework obtained

95% in the intensive workload of the performance compared to native environments. The scalability also was increased up to 7 VMs with acceptable performance.

A framework to enable GPU virtualisation in Cloud Computing environments named *gscale* was introduced by Xue et al. [161]. The *gscale* framework aimed to enhance the scalability issue in *gVirt* GPU virtualisation framework. To do this, the private shadow Graphics Translation Table (GTT), ladder mapping and slot sharing were developed. GTT was aimed to allow the global memory of the graphic card to be shared among virtual GPUs. Ladder mapping aimed to let the CPU accessing the physical memory. The goal of developing slot sharing was to enhance the performance of the virtual GPU under a large number of instances. The scheduler in slot sharing was optimised by dividing the memory graphics into many slots and every slot was dedicated to each virtual GPU. The experimental results conducted on Intel integrated GPU showed the scalability was improved 5x in Linux system and 4x in Windows system and produced a small performance overhead compared to gVirt.

FELIPE was another scheduling framework proposed by Zhao et al. [162] that consisted of several scheduling policies to enable efficient sharing and full utilisation of GPU resources among the existing VMs. *FELIPE* adopted a fine-grained scheduling technique to achieve the aims of the *FELIPE* framework. The mixed time event-based scheduling algorithm was designed to decrease the idle time during the process of VM switching. The seamless VM assignment process was developed to allow the seamless of switching VMs in several stages. Moreover, the hybrid per ring/VM scheduling algorithm was applied to place the workload to multiple GPU engines for simultaneous execution. *FELIPE* was implemented on Intel virtualisation technology for integrated Intel GPUs called *GVT-g* [163]. In the comparison with the default *GVT-g* configuration, *FELIPE* showed a performance improvement of 21.5% and 19.7% for the mixed time event-based and the seamless VM assignment process scheduling algorithms, respectively. Also, the performance was improved from 57.9% to 98.5% for the hybrid per ring/VM scheduling algorithm.

Moreover, Lu et al. [164] presented *gQoS*, an adaptive framework for virtualised GPUs in Cloud-based systems that supported resources sharing among virtual GPUs

instances with maintaining the QoS. The *gQoS* framework aimed to dynamically reconfigure the resources of active virtual GPUs to fit the allocated GPU workload and maintain the QoS. The *gQoS* framework contained a monitoring module, controlling module and scheduling module. The scheduling module altered the kernel driver to change the default setting of the time slot allocation and the context switch process for the existing virtual GPUs. The framework was conducted on Intel integrated GPU and compared with static scheduling policies: fair share allocation and proportional allocation. The framework was also compared with a dynamic scheduling policy: threshold-based control. The experimental results indicated that *gQoS* was more accurate and more stable, and the GPU utilisation was reduced to 25.85% compared with other scheduling policies.

3.5.1.2 Discrete GPUs

GVIM [20] scheduled VMs to access physical GPUs by using the concept of credits. Each guest was awarded credits and Xen credits as well. Credits represented the task execution time and Xen credits will be awarded by default during the booting time. Relying upon the value of each credit, the guest will be scheduled to the physical GPU. The scheduling policies used in *GVIM* were Round Robin (RR) and XenoCredit (XC). RR monitored and executed the guests' requests to access physical GPUs. Then, XC ensured to provide the basic requirements of QoS or fairness guarantee based on Xen credits to measure the needed time of the guest. The experimental results showed that *GVIM* was able to produce flexibility and efficiency with small performance degradation compared to non-virtualised environments.

Gupta et al. proposed *Pegasus*, a framework that enabled the scheduling coordination between CPU and GPU resources in a virtualised environment [165]. *Pegasus* allowed VMs to effectively share CPU and GPU resources by developing scheduling policies in the hypervisor to increase the throughput and reduced the latency. The developed policies were classified into Hypervisor Independent Policies, Hypervisor Controlled Policy and Hypervisor Coordinated Policies. In Hypervisor Independent Policies, the coordination between CPUs and GPUs was not supported and only support VMs scheduling to physical GPUs. Hypervisor Independent Policies contained Round Robin (RR) and AccCredit (AccC) scheduling

algorithms. RR played the baseline role for scheduling VMs to the accelerators. AccC proportionally shared the GPU resources to each VM based on the needed time to accomplish the task by giving accelerator credits to every VM. The aim of the Hypervisor Controlled Policy was to reduce the execution time of the applications by scheduling CPU and GPU tasks simultaneously, and this was achieved by developing Strict Co-Scheduling (CoSched) scheduling algorithms. CoSched mainly was dealing with latency-sensitive workload applications, such as financial applications [166]. Lastly, the aim of the Hypervisor Coordinated Policies was to enable the hypervisor to engage the CPU and GPU scheduling decisions rather than mentoring them. To achieve this aim, the Augmented Credit-based Scheme (AugC) and SLA Feedback (SLAF) scheduling algorithms were developed. AugC aimed to increase the throughput and enabling the coordinated scheduling by awarding additional temporary credits for next scheduled VM in the CPU for increasing the chance to be scheduled in the GPU. Then, SLAF aimed to maintain QoS requirements by adding feedback control to the AccC scheduling policy. The scheduling policies were implemented on Xen hypervisor. Experimental results exhibited acceptable performance penalties, and the performance of GPU resource sharing among 4VMs was improved compared to the basic GPU Driver Scheduling with the range from 18% to 140%.

A framework that enabled applications running within VMs with providing for sharing a single or multiple physical GPUs and runtime consolidation was proposed by Ravi et al. [19]. The framework was an extension of an open-source GPU virtualisation framework called *gVirtus* [167]. To enable the runtime consolidation, authors expanded *gVirtus* by developing the Dispatcher and Consolidation Decision Maker (DCDM) and the Virtual Context with Consolidator (VCC) components in the backend server. DCDM component was responsible for analysing the execution configuration of kernels that have been scheduled to the physical GPUs and decided whether it was valuable to consolidate or not. The execution configuration contained a set of details requested from the kernel to execute the task like the number of blocks and the threads within every block and the size of the shared memory. The consolidation decision in this work was based on two aims: increasing the throughput and keeping the performance of each application at an adequate

level. Subsequently, the VCC component executed the consolidation. To execute the consolidation during the runtime, three algorithms were developed. The first algorithm calculated the affinity score among the kernels to evaluate the benefit of the consolidation without the moulding technique. The second algorithm calculated the affinity score among the kernels with the moulding technique based on [168]. The third algorithm is to map the VMs to the physical GPUs based on the previous algorithms. The framework was evaluated by running applications that sharing two physical GPUs. The results showed a 50% average throughput improvement compared to sequential execution and the overhead of the framework was 4%.

Qi et al. [21] proposed *VGRIS* for resource scheduling in games in the VM. *VGRIS* was an API framework that manages the virtualised GPUs and isolates them by implementing scheduling algorithms without modifying the framework itself. The authors implemented three different scheduling policies and each policy had a certain task to evaluate *VGRIS*. The implemented scheduling policies were SLA-aware scheduling policy, proportional-share scheduling policy and hybrid scheduling policy. The SLA-aware policy aimed to just allocate enough GPU resources to every VM to meet the SLA requirements. Then, the aim of the proportional-share policy was to allocate all GPU resources in the system to each running VM proportionally based on the weight of each VM. Finally, the aim of the hybrid policy was to combine the previous policies. First, it allocated the minimum quantity of GPU resources to every VM in the system to meet the SLA requirements. Then, the excess of GPU resources was proportionally allocated to every running VM to increase GPU usage. The experimental results executed on Windows operating system exhibited that *VGRIS* was able to efficiently schedule GPU resources among different workload. Other studies that focus on Cloud gaming can be found in [169] and [170].

GPUvm was another study related to GPU full virtualisation in Cloud computing for discrete GPUs [75]. *GPUvm* is an open-source framework and its prototypes were implemented on Xen hypervisor to enable using GPU applications in Cloud computing environments. *GPUvm* reduced the GPU virtualisation overheads by utilising some optimisation techniques, such as decreasing the number of page

table scanning and decreasing the number of hypercalls in the hypervisor. Further, *GPUvm* supported full virtualisation, para-virtualisation, and high-performance para-virtualisation. The used scheduling policy in *GPUvm* to allocate VMs to Physical GPUs was GPU Fair-Share scheduling. With using this scheduling policy, each VM in the framework equally shared the amount of the physical GPU. The concept of this scheduling algorithm was inspired by the Bandwidth-Aware Non-preemptive (BAND) algorithm [171]. BAND was based on the CREDIT algorithm [39] that had some characteristics like the priority of scheduling the VM was based on reserved bandwidth and a certain waiting time after completion running the kernel for enabling a fair usage of the GPU resources time among the existing VMs. To evaluate *GPUvm*, Rodinia benchmark and some custom microbenchmarks were performed. Experimental results performed in different virtualisation types on Fermi Quadro 6000 NVIDIA GPU illustrated that the full virtualisation in *GPUvm* overheads was not decreased efficiently and the performance of para-virtualisation in GPUvm was slower than pass-through virtualisation technique between two or three times. The High-performance para-virtualisation technique produced a significant reduction in terms of overhead.

Hong et al [172] introduced *FairGV*, a framework to support GPU virtualisation to enable fairness of resource sharing and high performance for executing HPC applications in Cloud Computing. The authors claimed that *FairGV* can support both pre-emptive and non-preemptive GPUs architectures types. *FairGV* was implemented based on the *gVirtus* framework, an existing framework. To achieve the aforementioned goals and solving the non-preemptive scheduling technique supported by some GPU architectures, several scheduling policies were proposed and implemented. *FairGV* joined fair-queue policy, work conserving policy and GPU centric co-scheduling scheduling policies to obtain the fairness of GPU resources. Fair queue policy was used to reach fairness when resource sharing was limited. Work conserving policy was considered to increase the GPU utilisation when the framework was dealing with the non-intensive workload. Additionally, GPU centric co-scheduling policy was applied to deal with the interaction between GPU and CPU. To deal with non-preemptive GPU architectures, accounting mechanism and collaborative scheduling were applied. *FairGV* and its scheduling algorithms were

evaluated on Pascal TitanX NVIDIA GPU. The prototypes of *FairGV* implementation achieved close to the ideal fairness level (>= 0.97 with using min-max ratio), and it reached a small performance degradation (<= 1.02 of the total overhead) among the mix of evaluated GPU workloads.

Hu et al. [173] introduced *Olympian*, a framework aimed to fairly share the GPU resources when scheduling several Deep Learning (DL) workloads on a single GPU on TensorFlow (TF) [174], a serving middleware system in a Cloud-based System. *Olympian* firstly estimated the current time slice complete using linear regression model and it can be used for scheduling other jobs. Then, for scheduling jobs, the *Olympian's* scheduling algorithm merely needed small modifications on the TF processing loop scheduling algorithm. Also, *Olympian* implemented the fair share policy, weighted fair share and priority scheduling policies to achieve fairness among the jobs. The experimental results performed on Pascal GeForce GTX 1080 Ti NVIDIA GPU showed a fair GPU resource among several DL workloads, switched with these workloads within 2 milliseconds and produced a small overhead (less than 2%).

Ilager [175] extended the *Aneka* PaaS framework [176] to allow the use of GPUs in Cloud computing environments. The authors used the Round Robin scheduling algorithm to schedule GPU tasks. The Image edge detection application was used as a study case to evaluate the extended work, and it performed on QUADRO K620 NVIDIA GPU. The experimental results indicated that the GPU execution with a large size performed better than CPU execution. However, this work only supports Windows-based systems.

Yu [177] proposed scheduling policies to enhance the GPU resource utilisation and performance for GPUs dealing with GPU architectures that support Concurrent Kernel Execution (CKE) technique. CKE technique permits several GPU applications to be executed simultaneously in single or different GPUs. To achieve the objective of this work, an analytical model named *Moka* was created to predict the performance of CKE applications. *Moka* used GPU execution patterns, data transfer patterns stream running overhead and resource contentions for the model's inputs. Moka achieved an accuracy error within 12% compared to the actual execution. Then, a framework for interference–aware GPU workload scheduling

that relied on machine learning algorithms called *Magic* was introduced. *Magic* used an automated feature selection technique that relied on Principle Feature Analysis (PFA), and *Magic* only needed a small amount of profiling data of the application that is waiting in the queue to estimate the possible interference of the other applications. *Magic* supported single and heterogeneous GPUs. The framework was implemented by Python programming language on heterogeneous GPUs. The selected GPUs were Kepler GeForce GTX 760 Ti and Maxwell GeForce GTX 950 Ti GPUs. The developed scheduling policy within the Magic framework called InferBin performed 16% better performance than First Come First Serve (FCFS) policy in the single GPU. In a Heterogeneous GPU system, InferBin had 21% and 22% better performance than Least Loaded (LL) and Round Robin (RR) scheduling policies, respectively.

Moreover, Oh et al. [22] proposed a smart job placement algorithm for GPU workload using the DQN learning method based on Deep Reinforcement Learning. The aim of this algorithm was to maintain the performance of the application and to enhance the GPU utilisation when the GPU resources were shared. The history of the resources of the applications was used to train the model. HPC applications and Machine Learning workloads were used for experimental evaluation. Many placement technique types were compared like single, multiple and multiple placement techniques relied upon reinforcement learning. The proposed placement method was performed on a container GPU support and executed on a single Pascal GeForce Titan Xp NVIDIA GPU. The experimental results showed that the proposed placement method enhanced the GPU utilisation with small performance degradation.

3.5.2 Energy and QoS considerations

With considering GPUs as a part of physical resources in Cloud computing, *vGASA*, an adaptive framework for managing virtualised GPU resources in a Cloud gaming environment to preserve the performance in the run time, was proposed in [178]; *vGASA* has had three adaptive algorithms: SAL-Aware (SA), Fair SLA-Aware (FSA) and Enhanced SLA-Aware (ESA). SA algorithm aimed to achieve the SLA requirements for each VM. FSA algorithm aimed to provide gaming servers with a maximum level of GPU resource usage. ESA algorithm aimed to balance the

gaming performance and also to balance many users located on a single PM. The experimental results that performed on HD 6750 AMD GPU and Windows operating system on VMware Cloud platform indicated that *vGASA* can maintain the performance with a limited overhead from 5 to 12 percent. However, this work has merely considered the Gaming workload on GPUs which needs specific requirements and deals with a homogeneous AMD GPU.

Additionally, *EvGPU*, an adaptive framework that guarantees SLAs with energy efficiency consideration in virtualised GPUs in Cloud gaming, was proposed by Guan et al. [121]. *EvGPU* is composed of two-layer control components based on the feedback control method. The first control layer ensures SLA guarantees by implementing Proportional Integral (PI) and was measured by watching the Frames Per Second (FPS) number in every online game by assigning several threshold levels. Subsequently, the second control layer applied Dynamic Voltage Frequency Scaling DVFS technique to reduce the power consumption of the GPU based upon the existing FPS. Experimental results conducted on the same system setup of the *vGASA* framework showed that *EvGPU* was efficiently able to reduce the power consumption of GPUs by achieving SLA in online games run within VMs. Similar to the previous work, *EvGPU* deals with gaming workloads and only considers a single GPU.

Several scheduling policies with different goals were proposed and implemented in *GPUCloudSim* [23]. To allocate a virtual GPU into a physical GPU in a single server, three scheduling policies were implemented: first-fit, breadth-first and depth-first. In *GPUCloudSim*, the virtual GPU can be allocated into many available GPUs. First-Fit, breadth-first and depth-first scheduling policies were also used for allocating a virtual GPU to several GPUs. For VM allocation attached with a GPU, the First-Fit scheduling and the implemented proposed First-Fit Increasing (FFI) policies were introduced. The proposed FFI scheduling policy defined the resource bottleneck in each configurations pair between the VM and the physical host. Then, VMs were sorted in ascending order based on the required resources. Finally, VMs were allocated using the First-Fit scheduling policy. In terms of GPU provision, space-shared, time-shared and fair-shared scheduling policies were implemented. The virtual GPU used the physical GPU until completing the job in the space-shared

scheduling policy. Many virtual GPUs shared a physical GPU in the space-shared scheduling policy. All virtual GPUs shared a time slice equally on the physical GPU, and the virtual GPU will be scaled up when the processing capability was greater than the hosted GPU. Moreover, the GPU task scheduling policies were implemented. These policies aimed to provision the virtual GPU resources for running the task. The implemented scheduling policies for task scheduling were leftover and co-execution multitasking. The task will be released when it is completed. The tasks co-execution happened when enough resources existed. The simulation setup simulated VMware Horizon [179] and vSphere [180] hypervisor with NVIDIA GRID. The simulated results attested that the proposed First-Fit Increasing (FFI) VM placement performed better than the First-Fit scheduling policy developed by VMware Horizon in terms of acceptance rate, decreased makespan (the total length of the schedule) and energy saving with 59%, 25% and 21%, respectively. This work considers the use of NVIDIA GRID K1 and K2 [122] for *GPUCloudSim* evaluation; however, these cards belong to a homogeneous GPU architecture (Kepler architecture). Moreover, the QoS requirements have not been taken into account in this work, and the simulator cannot be used to simulate other GPU architectures.

Guleria et al. implemented *QuADD-SIM*, which is based on *CloudSim*, to simulate the GPUs deployment usage in disaggregated data centres architecture on a large scale [181]. The disaggregated data centre is built as a set of individual physical resources, and these resources are collected to be workload execution unit for on-demand usage [182]. *QuADD-SIM* aimed to show the benefit of using disaggregated architecture in terms of increasing resource utilisation and reducing power consumption compared to the traditional data centres architecture for GPUs deployment. *QuADD* framework consisted of QUADD-Broker and QUADD-Datacentre components. QUADD-Broker aimed to orchestrate the requests of GPU workload on behalf of the Cloud users. QUADD-Datacentre aimed to represent the Cloud data centre in both traditional and disaggregated configurations. For the traditional configuration, the server nodes contained fixed attached GPUs, whereas in the disaggregated configuration, the GPUs were regulated in GPUs resources pool. Moreover, QUADD-Datacentre was responsible to allocate the VM to the

available GPU, and the used scheduling policy for the VM allocation was best-fit in both configurations. *QuADD-SIM* classified the GPU deployment in the disaggregated configuration into the Traditional VM Size (TVS) and the Custom VM Size (CVS). *QuADD-SIM* used the available GPUs and their VM instances in Amazon AWS. The used GPUs were Kepler K80, Maxwell M60, and Volta V100 NVIDIA GPUs. The authors used Deep Learning training workloads to simulate the GPU workload in *QuADD-SIM*, and they used the VM failure request number and the GPU Watt/hour as evaluation metrics. Several simulation experiments showed that the GPU deployment in disaggregated configuration produced 5.14% and 7.90% additional failure request number. In addition, the GPU deployment in disaggregated configuration reduced the Watt/hour 10.9% and 3.3% less than the traditional deployment. However, this work has focused on the datacenters design impacts on the VM deployment failure and GPU Watt/hour rather than the impact of task scheduling on energy consumption and performance, and also the authors have not considered the energy modelling. Moreover, this work has used only Deep Learning workloads, and it has not considered the GPU applications workload and the QoS requirements.

Additionally, Ilager et al. introduced a data-driven with Dynamic Voltage Frequency Scaling (*D-DVFS*) scheduling policy for GPUs to reduce energy consumption [24]. First, prediction models for power and execution time considering several clock configurations were proposed. Different ML techniques were used for predicting power and execution time, such as linear regression, support vector machine, lasso-linear regression (Lasso), and gradient boosting algorithms like eXtreme Gradient Boosting (XGBoost) and CatBoost algorithms were also used. Then, since the problem of reducing the energy consumption with the deadline constraints is an NP-hard problem, a heuristic *D-DVFS* scheduling policy with the use of the prediction model was created. the experimental studies were conducted on Grid'5000 testbed [183] using a single NVIDIA GPU (Pascal P100). The experimental results attested that the prediction model predicted energy and execution time of GPU applications with average Root Mean Square Error (RMSE) error of 0.38 and 0.05, respectively. Moreover, the created algorithm reduced energy by 15.07% less than the default clock GPU and max clock GPU configurations. Yet, this work has

not considered heterogeneous GPUs for scheduling purpose, and the authors have not considered the impact of the cluster of applications including different size of applications on the scheduler.

3.6 Overall Discussion

Accurate measuring of power and energy consumption is not always achievable. Therefore, power and energy models are an alternative way to predict power and energy with good accuracy by using several modelling techniques, such as Machine Learning techniques. Power and energy modelling can be beneficial in several ways, such as optimising the power consumption of applications without harmfully affecting the performance. Power and energy modelling can also save time and financial expenses [106].

The usage of energy modelling in Cloud computing can support software developers and designers to develop energy-aware applications. Furthermore, power and energy modelling can help Cloud computing providers to instantiate efficient and energy-aware management techniques.

3.6.1 Power and Energy Modelling

In section 3.2, GPU power modelling techniques even in Cloud and non-virtualised environments have been reviewed. More specifically, section 3.2.1 has discussed the related work of GPU power modelling in non-virtualised environments. This has been divided into statistical and machine learning models, low-level analytical models, simulation-based models and power modelling with the DVFS technique.

As stated in section 3.2.1.1, studies in [104]-[108] have used statistical and machine learning techniques to predict GPU power consumption, but without the consideration of heterogeneous GPU architectures. Although the work showed in [109] has considered the heterogeneity of GPU architectures, yet the non-linear relationship between power consumption and performance to support new GPU architectures to produce a reasonable power prediction accuracy has not considered [10]. The work in [26] developed an online model to estimate the power consumption of GPUs, but the impact of GPU and memory utilisations on power consumption have not been investigated. Also, there is an overhead in some applications at runtime. Moreover, the presented work in [111] has created a GPU

power consumption model on several GPU architectures, yet the model has not performed efficiently in applications with short execution time. Also, the impacts of GPU and memory utilisation on power consumption have not been considered.

Studies presented in [29] and [112] have presented analytical models using the low-level micro-architectural components. This approach can produce acceptable estimation results. Yet, such kinds of models need a deep knowledge of the GPU micro-architecture [26], and it is hard to acquire information of heterogeneous GPU micro-architectures components when running several applications since the power consumption of each micro-architectural component needs to be measured and access rate when running every application needs to be calculated [126]. Moreover, these studies have not evaluated heterogeneous GPU architectures for models evaluation.

Moreover, *GPUWattch* [31] and *GPUSimPow* [115] have used low-level micro-architectural GPU power modelling, and are aimed to be used as GPU power simulators. Simulation tools are useful for cost reduction. However, these simulations produce significant overhead and require high knowledge for configurations setup. Additionally, simulation tools are not suitable for several generations of GPU architectures such as Kepler architecture [26].

Another approach has developed GPU power models integrated with several DVFS configurations. This approach can be found in [116], [117], [118] and [119]. In this thesis, the default setup is used, and the GPU power modelling integrated with DVFS configurations is out of scope. The studies that have presented in section 3.2.1 have been developed and evaluated in non-virtualised environments as a common factor.

Section 3.2.2 has reviewed the related work of GPU power modelling performed in Cloud computing environments. *GPUCloudSim* [23] is a standard tool that is used for GPU simulation in Cloud computing environments. However, its authors have merely considered the linear relationship which cannot represent the complexity of new GPU architectures [107], and also have not investigated heterogeneous GPUs.

A high-level GPU power modelling approach using hardware performance counters and machine learning is characterised by simplicity [107], [111] and the diversity of configurations [27]. Nevertheless, there is a debate about the benefit of using just performance counters for developing power models. Considering performance counters can produce overhead and increase complexity [184]. Considering hardware performance counters also can create insufficient outcomes because these counters do not cover all the hardware components [185].

Therefore, this work will investigate the feasibility of considering hardware performance counters for developing a power consumption model evaluated on heterogeneous GPU architectures. The Deep Neural Network technique will be used to develop The GPU power model in a Cloud computing environment. A further investigation will be performed when we find that performance counters usage is not feasible. Table 3.1 summarises the current work that is performed in GPU power modelling.

Table 3.1: Summary of the Work in GPU Power Modelling

Criteria / Reference	Heterogeneous GPU architecture	Modelling Technique Considered	Cloud Computing Environment
[104]-[108]	No	Statistical and Machine Learning	No
[109]- [111]	Yes	Statistical and Machine Learning	No
[29], [112]	No	Low-Level Analytical Models	No
[31], [115]	Yes	Simulation-based	No
[116]	No	Power modelling with DVFS Technique	No
[117]- [119]	Yes	Power modelling with DVFS Technique	No
[23]	No	Statistical and Machine Learning	Yes

In section 3.3, GPU energy modelling techniques in both Cloud and non-Cloud environments have been reviewed. The work performed in GPU energy modelling can be divided into indirect models and direct models to estimate GPU energy consumption. More specifically, section 3.3.1 has discussed the related work of GPU energy modelling in non-virtualised environments. Direct GPU energy models aim to directly estimate GPU energy consumption. Indirect models, including

analytical models, aim to predict power and the execution time individually, and then the energy consumption is calculated. This can be performed by developing several sub-models to estimate energy consumption.

[29], [123] and [124] have developed analytical models to predict GPU power and execution time separately to calculate the estimated energy, [107] have used a neural network to predict GPU power consumption and an analytical model to predict the execution time. The estimated GPU energy has then been calculated. [109] and [27] have used statistical and machine learning techniques to estimate GPU power and performance separately. However, the aforementioned studies in section 3.3.1 require more effort and time. These studies required several mathematical calculations to develop the necessary sub-models to estimate GPU energy.

Moreover, [106] in section 3.3.2 has developed a model to directly predict GPU energy consumption but considering only a single GPU device to evaluate the GPU energy model. In general, a common factor in the studies in sections 3.3.1 and 3.3.2 is that they all have been performed in non-virtualised environments.

In section 3.3.3, The work in [126] has developed an analytical model to estimate GPU energy consumption that can be used as a GPU simulation tools in Cloud computing environments. To estimate energy consumption, firstly, the GPU power consumption has been calculated. However, several calculation steps have been used to estimate GPU energy consumption. Moreover, there is no clear indication that the work has been performed in Cloud computing environments.

The presented work in [24] has developed GPU power and performance models; however, it has not considered heterogeneous GPU architectures.

Therefore, this work aims to develop an agnostic GPU energy model compatible with different type of GPU architectures containing different features in a Cloud computing environment. Additionally, this model aims to directly estimate GPU energy consumption without the demand for developing several sub-models. Table 3.2 illustrates a summary of the related studies conducted in GPU energy consumption modelling.

Table 3.2: Summary of the Current Studies in GPU Energy Modelling

Criteria / Reference	Heterogeneous GPU Architectures	Modelling Technique Considered	Cloud Computing Environment
[10], [29], [123], [109], [27]	No	Indirect	No
[124]	Yes	Indirect	No
[106]	No	Direct	No
[126]	Yes	Indirect	No
[24]	No	Indirect	Yes

To estimate the GPU energy consumption without the need for profiling and collecting the required parameters of the GPU applications running on a VM, it can be necessary to first predict the required resources such as the workload and performance. These can be used to estimate energy consumption. The studies in section 3.4.1 have aimed to develop models to predict the GPU execution time. However, these works have been conducted in non-virtualised environments and not used to estimate energy consumption. The work in [177] has developed an analytical model to predict the performance of GPU applications in Cloud computing, but it has not been used to estimate GPU energy consumption.

It is beneficial to establish a provisioning framework of resources to support executing Cloud applications for reducing energy cost. Even Studies in [138]-[143] and [145]-[155] discussed in section 3.4.2 have dealt with predicting the required resources to execute the applications in Cloud computing for reducing the operation cost, decreasing the energy consumption and providing a stable performance to the end-user. However, these studies have not considered GPU applications and the supplies of resources to run these applications in Cloud computing.

Studies in [156]-[160] in Section 3.4.3 have proposed frameworks to predict energy consumption by firstly estimating the required resources and workloads in Cloud computing. Some of these studies have used workload prediction models for developing managerial actions to reduce energy consumption, such as [157]. These studies have considered the resources of CPUs, the main memory and the network

traffics, but they have neglected the required resources to run GPU applications in Cloud computing environments, as shown in Table 3.3.

Thus, this work aims to develop an energy framework to predict GPU energy consumption to reduce the cost of energy estimation. The framework aims to eliminate the need for running the built-in profiling tools for data collection. This framework will be evaluated on heterogenous GPUs in a cloud computing environment.

Table 3.3: Summary of Energy Modelling based on Workload

Criteria / Reference	Consideration of GPU resources and workloads	Cloud Computing Environments
[156]	No	Yes
[157]	No	Yes
[158]	No	Yes
[159]	No	Yes
[160]	No	Yes

3.6.2 GPU Scheduling

The studies performed in [83]-[162] and [164] in section 3.5.1.1 have dealt with GPU scheduling in Cloud computing with considering integrated GPUs. Integrated GPUs have a different architectural design in comparison to discrete GPUs. Integrated GPUs share the main memory with the CPU while discrete GPUs have their private memory. Further, discrete GPUs have better computing capabilities than integrated GPUs.

Although the studies in [20], [165], [19], [21]-[75], [172], [173], [175], [177] and [22] in section 3.5.1.2 have dealt with discrete GPU types for GPU scheduling in the Cloud computing, they have merely considered the performance factor for allocating virtual GPUs to physical GPUs neglecting the energy consumption criterion.

Furthermore, GPU scheduling with energy and QoS considerations have been reviewed in section 3.5.1. The studies in [178] and [121] have proposed GPU energy-aware scheduling policies with the consideration of QoS in Cloud gaming, but these studies have only used a single homogeneous GPU architecture.

GPUCloudSim [23] has proposed scheduling policies aimed to reduce energy consumption. However, heterogeneous GPU architecture and QoS requirements have not been considered. Similarly, the work in [181] has not considered the application's deadline as a QoS requirement. Although the work in [24] has developed an energy aware scheduling policy taking into account QoS requirements (specifically the deadline), the work has considered a single homogeneous GPU architecture to evaluate the scheduling policy. Table 3.4 summarises the works performed on GPU scheduling in Cloud Computing.

Table 3.4: Summary of the work in GPU scheduling in Cloud Computing

Criteria / Reference	Scheduling Consideration			GPU Architecture Type	Heterogeneous GPU architecture
	Performance	Energy	QoS		
[83], [164]	Yes	No	Yes	Integrated	No
[161], [162]	Yes	No	No	Integrated	No
[20], [165], [21]	Yes	No	Yes	Discrete	No
[19], [169], [75], [172], [173], [175], [22]	Yes	No	No	Discrete	No
[170], [177]	Yes	No	Yes	Discrete	Yes
[178], [23], [121], [24]	Yes	Yes	Yes	Discrete	No
[181]	Yes	Yes	No	Discrete	Yes

CSPs always strive to reduce the cost of Cloud computing and increase the Return of Investment (ROI). Resource management techniques are beneficial for CSPs to increase ROI and profits [186]. As discussed in section 1.1, operational cost is a major issue for CSPs in Cloud computing, such as the energy consumption cost. Considering accelerator units like GPUs can help to reduce energy consumption because GPUs are more energy-efficient than CPUs [19]. Considering accelerators unit can maximise the ROI of CSPs to fit the Cloud users usages and reduce SLA penalties [187]. In terms of energy cost, considering GPUs in Cloud computing is helpful for CSPs. However, CPUs and GPUs resources cost like maintenance cost needs more investigation, and it will be a potential future work.

In Cloud computing, the heterogeneity level exists in the context of heterogeneous CPUs such as studies conducted in [158] and [159]**.** Moreover, the heterogeneity

level exists in the size of VMs including the capacity of the virtual CPUs, RAM, disk storage and network like the work performed in [160]. The context of heterogeneity in this research is the heterogeneity of GPU architectures in the Cloud computing infrastructure.

Therefore, after considering the related work in the literature, there is a lack of research regarding power and energy consumption modelling for heterogeneous GPU architectures in Cloud computing environments. Furthermore, applications supported by heterogeneous GPUs with the consideration of 1) the estimation of the required resources and the consumed energy prior to their scheduling in Cloud computing environments, and 2) key factors including performance and energy consumption, and 3) their QoS requirements, need to be addressed.

3.7 Summary

This chapter has presented the current literature review regarding GPU power modelling, GPU energy modelling and GPU scheduling in Cloud computing environments. Firstly, it has introduced several types of GPU power modelling including Statistical and Machine Learning models, low-level analytical models, simulation-based models and power modelling with the DVFS technique in non-virtualised and Cloud computing environments. Secondly, it has presented the types of GPU energy consumption modelling in non-virtualised and Cloud computing environments. It has then shown the GPU performance models, workload prediction and energy modelling based on workload prediction. After that, the chapter has shown the current works in GPU scheduling in Cloud computing environments. Finally, a summary of the closest related work to this research was presented and discussed.

The next chapter will present the proposed architecture to support and facilitate the energy-awareness of heterogeneous GPUs resource management in Cloud computing environments.

Chapter 4 Proposed Architecture

4.1 Overview

This chapter will provide a brief introduction to the development of Cloud Computing usage and the consequences of this development in section 4.2. Then, the proposed architecture, its components and their interactions will be illustrated in section 4.3. After that, the way of analysing heterogeneous GPUs in a Cloud environment will be introduced in section 4.4. Several experiments performed on a Cloud testbed for analysing heterogeneous GPUs and their results will be demonstrated in section 4.5. Finally, the overall discussion of the experiments' results will be presented in section 4.6.

4.2 Introduction

Recently, accelerator units, such as GPUs have been introduced in Cloud computing, and this introduction has shown new issues in Cloud infrastructure, as stated in section 1.1.

Cloud computing leverages the virtualisation of computing resources to allow end-users to provide them at an acceptable price. In increasing the computation demands, GPUs have been introduced in Cloud data centres because of their performance abilities and their suitability for some applications [188]. Moreover, GPU clusters will play an important role in Cloud computing data centres since some compute-intensive applications need to involve GPUs with CPUs [189].

The Cloud service provider has to continuously provide QoS for the end-user to avoid SLA violations by performing some actions. These actions incur a tremendous amount of energy consumption. Thus, energy-efficient solutions in Cloud data centres have become a major research concern. Standard Cloud physical infrastructure resources (i.e. CPU, memory and network) have been intensively studied by researchers in terms of performance and energy consumption. Therefore, it is important to establish a provisioning framework for heterogeneous GPU resources to achieve QoS requirements.

Therefore, this research proposes a holistic architecture to manage the GPU resources in Cloud computing environments for general purpose usage. The

architecture will focus on the application deployment time and ensure the applications QoS is fulfilled at the operation time. This proposed architecture also considers performance and energy consumption as key factors.

4.3 Proposed Architecture

To achieve the research objectives, an adaptive systematic architecture is proposed, as shown in Figure 4.1, to manage the GPU applications that run within VMs focusing on two parameters: energy consumption and performance in Cloud computing environments in two phases: deployment and operation times. The proposed architecture follows the standard Cloud computing architecture, as discussed in Chapter 2, which is composed of three basic layers: the SaaS layer where the service constructed, the PaaS layer where the deployment task deployed and IaaS where the operation control task executed. The proposed architecture mainly deals with PaaS and IaaS layers and abstract SaaS layer. In the PaaS layer, the application scheduler that allocates the GPU application to an appropriate VM is located. In IaaS, heterogeneous GPUs analyser, power and energy modeller and the self-adaptation manager are located.

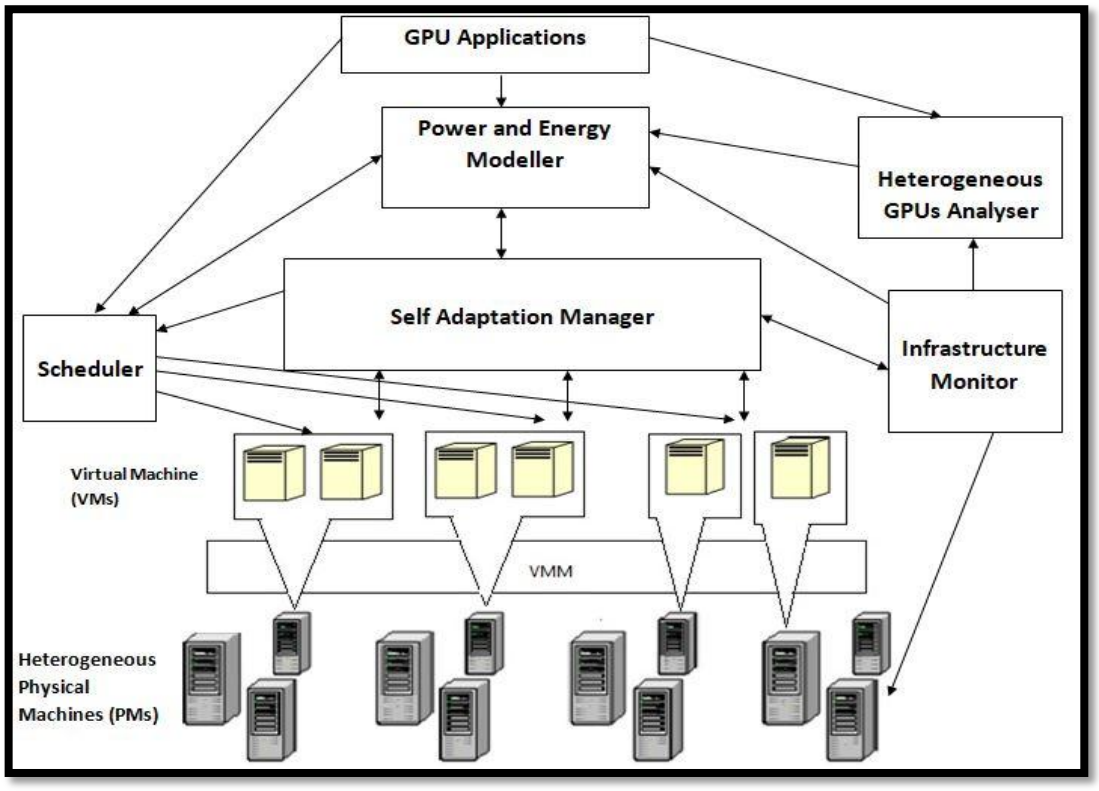


Figure 4.1: High Level Proposed Architecture

The proposed architecture considers the life cycle of GPU applications in Cloud computing starting from the deployment time to the possibility of a self-adaptation framework to maintain the application QoS at the runtime.

Before the service deployment initialised, the power and energy prediction modeller will estimate the power and energy consumed by the GPU applications. This will allow the application scheduler to allocate the GPU application to the most energy-efficient VM. In the operation time, the self-adaptation manager will continuously monitor the application's performance and will take proactive and corrective actions when the performance degrades like performing live VM migration.

4.3.1 Architecture Components and Interaction

The proposed architecture consists of interacting components to achieve the goals of this research, and each component has a certain role as shown next.

The Heterogeneous GPUs Analyser aims to analyse and compare heterogeneous GPU architectures, e.g. Fermi and Kepler, in terms of performance, power and energy consumption, see Section 4.4. The Heterogeneous GPUs Analyser interacts with the infrastructure monitor to obtain different GPU measurements, as discussed in section 4.5.

The Power and Energy Prediction Modeller is responsible for predicting the energy consumption of running GPU applications on Virtual Machines (VMs) taking into account the power consumption in the deployment and operation phases. The modeller interacts with the infrastructure monitor when the application is running on a certain VM.

The Application Scheduler allocates applications to the suitable VM in terms of energy and performance based on the output from the power and energy modeller and during the operation phase to interact with the self-adaptive manager.

The Infrastructure Monitor is responsible for observing the performance and power consumption of the physical infrastructure and sending the monitored data to the self-adaptation manager and power and energy modeller.

The Self-Adaptation Manager is a component to ensure that QoS is fulfilled during GPU applications operating within the VMs and implements the MAPE-K [190] (Monitor, Analyse, Plan, Execute and Knowledge) technique. The self-adaptation manager will need to invoke the scheduler and power and energy modeller to maintain the application's QoS. The purpose of invoking the modeller is to predict the future behaviour of the application. When the future behaviour of the application shows a performance degradation, the self-adaptation manager will invoke the VM scheduler to reschedule the VM to stabilise the application's performance. The self-adaptation manager is a part of the proposed architecture but not considered in this research. It is considered as future work to extend this research.

4.4 Heterogeneous GPU Benchmarking and Analysis

As a first step to implement the proposed architecture, it is important to analyse and compare the heterogeneity of the GPU architectures in the Cloud infrastructure to understand the different behaviours of heterogeneous GPUs architectures in accordance with adequate power, energy models and resource management development. Therefore, a comparative experimental study that reveals the architectural impact on the performance, power and energy consumption on heterogeneous GPUs is performed.

There are two generations of NVIDIA's GPUs architectures dealt with in this study: Fermi and Kepler. Kepler architecture is newer and more energy-efficient than Fermi. C2075 and K40c are examples of Fermi and Kepler architectures, respectively.

The architectural behaviour of GPUs is analysed in terms of three criteria: performance, power and energy consumption. The impact of the software side on the architecture side of the GPUs in the aforementioned criteria is investigated. The software side is defined as the number of blocks and the number of threads per block assigned by the developer to run the kernel which is the function that is executed by the GPU. This is performed by using a specific programming language that deals with GPUs. The selected programming language is CUDA which is supported by NVIDIA. Moreover, we study the factors that have an impact on

performance and power consumption. These factors are the hardware block scheduling, the GPU Occupancy and the memory hierarchy, such as the device memory DRAM.

The hardware block scheduling can be defined as the number of blocks which can be allocated in a Stream Multiprocessor (SM). The equations in the CUDA Occupancy calculator are used in [191] to find the number of blocks allocated in each SM and the GPU Occupancy.

To find the number of blocks per SM, we first calculate the number of warps per block in a given kernel, by using the following formula:

$$\#\ warps\ per\ block = \frac{\#allocated\ threads\ per\ block}{\#warp\ size} \quad (1)$$

Where warp size = 32 threads [192],
Then, we find the number of blocks per SM by using the following formula:

$$\#blocks\ per\ SM = \min\left(\#max\ blocks\ per\ SM, \frac{\#\ max\ warps\ per\ SM}{\#\ warps\ per\ block}\right) \quad (2)$$

Occupancy is an important metric to analyse the performance when dealing with GPUs for general purpose use. GPU Occupancy is defined as the ratio of the active number of threads to the maximum number of threads in the SM. The value of the GPU Occupancy is between 0 and 1.

To calculate the GPU Occupancy, we use the following formula:

$$GPU\ Occupancy = \frac{\#blocks\ per\ SM \times \#\ warps\ per\ block}{\#\max warps\ per\ SM} \quad (3)$$

The percentage of active threads per block is the allocated thread over the maximum number of threads per block. It is calculated by the following formula:

$$Active\ Threads\ per\ Block = \frac{\#allocated\ threads\ per\ block}{\#\max number\ of\ threads\ per\ block}\ x\ 100$$

(4)

We assume that the percentage of the active threads per block represents the GPU workload since it is an adequate representative for the GPU utilisation for both GPU architectures.

4.5 Performing Heterogeneous GPUs Benchmarking and Results

This section will explain the steps of the Heterogeneous GPUs Analysis on two VMs. Each VM contains different GPUs. These GPUs are NVIDIA Fermi C2075 and NVIDIA Kepler K40c.

4.5.1 Experimental Setup and Design

The experiments are performed in the School of Computing Cloud testbed at the University of Leeds. The experiments are performed on two different Virtual Machines (VMs) supported by two heterogeneous GPUs. These heterogeneous GPUs are NVIDIA Fermi C2075 and NVIDIA Kepler K40c. OpenNebula [42] is used as a Virtual Infrastructure Manager (VIM). The KVM hypervisor is used in this experiment. Additionally, the Operating System (OS) used is Linux CentOS. Table 4.1 shows the resources of each VM in the Cloud testbed, and Table 4.2 shows the details of Fermi C2075 and Kepler K40c GPUs.

Table 4.1: Two VMs Details in the Cloud Testbed

VM Characteristics	VM1	VM2
CPU	Intel Xeon E5-2630 v3 2.4GHz	Intel Xeon E5-2630 v3 2.4GHz
VCPU	8	8
RAM Size	32 GB	64 GB
GPU	NVIDIA Fermi C2075	NVIDIA Kepler K40c
Hypervisor	KVM	
CUDA Compiler Version	7.5	
OS	Linux CentOS	
VIM	OpenNebula	

Table 4.2: Fermi C2075 and Kepler K40c GPUs Characteristics

GPU Details	Fermi C2075	Kepler K40c
CUDA Cores	448	2880
SMs	14	15
Cores/SM	32	192
Core frequency(MHz)	1150	745
Memory Size (GB)	6	12
Max Power Consumption (W)	225	235
Max Threads/ Block	1024	1024
Max Warp/SM	48	64
Max Thread Blocks/SM	8	16
ECC Mode	Enabled	Enabled

A CUDA matrix multiplication application with o (n^3) complexity is used in these experiments since it is flexible to increase the size of the matrix. CUDA Compiler Version 7.5 is used to compile the matrix multiplication CUDA codes. Several tools supported by NVIDIA, as shown in Figure 4.2, are utilised. The NVIDIA CUDA Compiler (NVCC) to compile the different matrix multiplication application sizes is selected. The NVIDIA System Management Interface (nvidia-smi) [193] monitoring tool to profile the GPU power consumption and the temperature at the runtime is used. Additionally, the NVIDIA Profiler (nvprof) [194] is utilised to measure the hardware performance counters in the runtime.

The objectives of the experiments are to:

- Investigate the relationship between GPU workload and power consumption and influential factors as well,
- Explore the blocks and the threads block allocations' impact on energy consumption,
- Explore the temperature impact on power consumption.

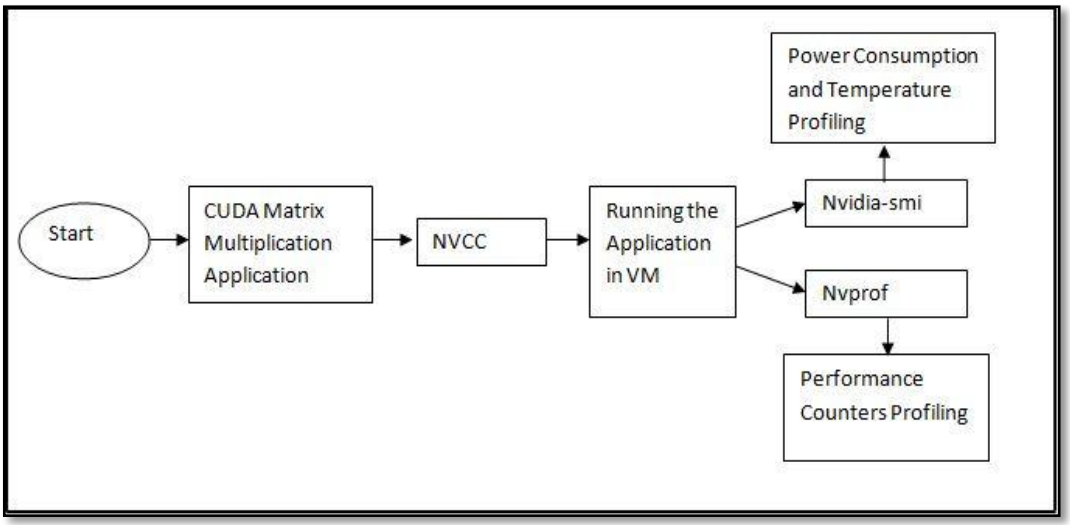


Figure 4.2: The Analysis Workflow and the Utilised Tools

4.5.2 Relationship between GPU Workload and Power Consumption and Influential Factors

The design of this experiment aims to find the relationship between the GPU workload and the GPU power consumption in both Fermi C2075 and Kepler K40c GPUs and the influential factors on performance and power consumption.

The number of threads per block up to the maximum number (1024 threads per block) is gradually increased, keeping the number of blocks constant. The number of blocks was 80 x 80 to ensure that SMs were working simultaneously. By increasing the number of threads per block, we increased the size of the memory as well. Then, we ran each matrix multiplication size five times and calculated the average power consumption and execution time. We profiled the GPU power consumption every 50 milliseconds.

4.5.2.1 Fermi C2075 Results

Table 4.3 shows the results of this experiment in the Fermi C2075 GPU. We applied the regression analysis (linear and nonlinear) to find the relationship between power consumption and the active threads per block. After applying this analysis, we found that the relationship tends to be more nonlinear by applying the quadratic regression since the R-square value in the quadratic regression (0.9528) is greater than the R-square value in the linear regression (0.4225). Figure 4.3 shows this relationship between the active threads per block and the power consumption in the C2075 Fermi GPU.

Table 4.3: The Results in Fermi C2075 GPU between Power Consumption and Workload

Matrix size	Threads Number	Active Threads per Block	Average Execution time (S)	Average Power Consumption (W)
480x480	36	4%	0.00828	87.79
800x800	100	10%	0.02717	94.48
1120x1120	196	19%	0.06493	126.27
1440x1440	324	32%	0.12215	137.53
1760x1760	484	47%	0.2238	149.11
2080x2080	676	66%	0.39725	160.05
2400x2400	900	88%	0.56202	136.03
2560x2560	1024	100%	0.65631	133.51

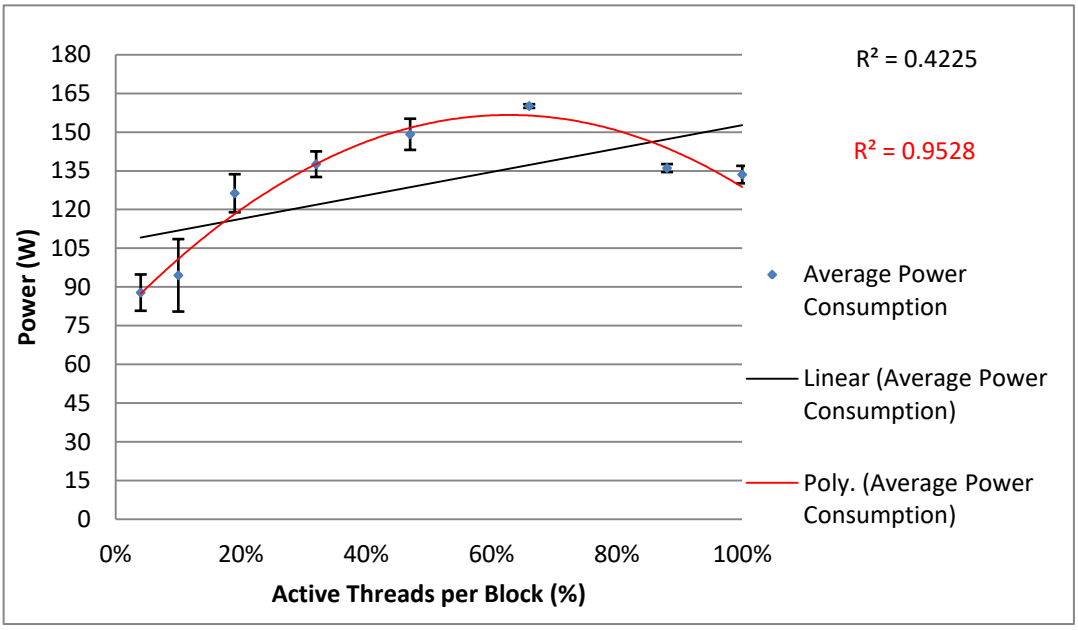


Figure 4.3: The Regression Analysis Power Consumption and the Active Threads per Block in Fermi C2075

4.5.2.2 Kepler K40c Results

Table 4.4 shows the results of this experiment. Similar to the previous experiment, the regression analysis (linear and nonlinear) was applied to find the relationship between power consumption and the active threads per block, as shown in Figure

4.4. After applying this analysis, we found that the relationship tends to be nonlinear, applying the quadratic regression, since the R-square value in the quadratic regression is greater than the R-square value in the linear regression. However, the difference between them is not so high, being 0.9875 and 0.8976, in Kepler and Fermi GPUs, respectively.

Table 4.4: The Results in Kepler K40c GPU between Power Consumption and Workload

Matrix Size	Threads Number	Active Threads per block	Average Execution time (s)	Average Power Consumption (W)
480x480	36	4%	0.00835	52.5
800x800	100	10%	0.02032	57.34
1120x1120	196	19%	0.0468	76.09
1440x1440	324	32%	0.08846	80.3
1760x1760	484	47%	0.15599	98.4
2080x2080	676	66%	0.24055	106.43
2400x2400	900	88%	0.38215	111
2560x2560	1024	100%	0.2745	111.57

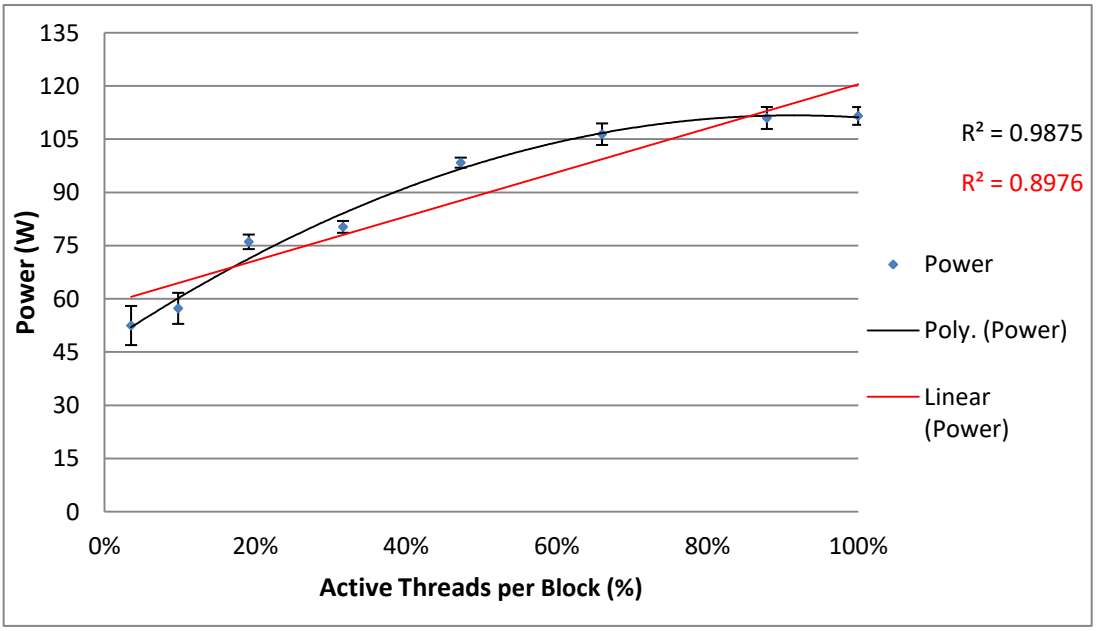


Figure 4.4: The Regression Analysis Power Consumption and the Active Threads per Block in Kepler K40c GPU

4.5.2.3 Results Analysis

Considering Fermi C2075 GPU, we found that there is a gradual increase in power consumption up to a certain level of the number of threads per block percentage,

when the active threads per block percentage is 66%. Then, power consumption significantly decreases to 136 Watts (W).

To explain the trend of the power consumption during an increase in the GPU workload, we need to compose a performance and architectural analysis for the applications running on the Fermi C2075 GPU at the runtime.

After analysing the GPU micro-architectures behaviour at the runtime by applying the hardware performance counters, we observe that the behaviours of some of these counters have unexpected values, specifically, the memories behaviour, such as the device memory, L2 and L1 cache memories. Table 4.5 shows the values of the performance counters related to specific memory types for the 2080 x 2080, 2400 x 2400 and 2560 x 2560 matrices because the drop of power consumption began when the size of the matrix was 2080 x 2080.

As shown in Table 4.5, we found that the specific performance counters values in the 2560 x 2560 matrix were smaller than the performance counters values in 2400 x 2400 and similarly in 2080 x 2080, even the memory sizes in (2080 x 2080 and 2400 x 2400) were smaller than the 2560 x 2560 matrix.

Table 4.5: Performance Counters values of the Memory types in Fermi C2075 GPU

Counter Name	Counter Description	Counter Value (2080x2080)	Counter Value (2400x2400)	Counter Value (2560x2560)
gst_transactions	Global Store Transactions	411362	520331	204960
dram_read_throughput	Device Memory Read Throughput	19.179GB/s	3.3041GB/s	2.8727GB/s
dram_write_throughput	Device Memory Write Throughput	94.811MB/s	61.674MB/s	47.643MB/s
l2_l1_read_hit_rate	L2 Hit Rate (L1 Reads)	91.85%	76.42%	67.85%
l2_read_transactions	L2 Read Transactions	1960588232	174374480	131120172

Then, we observed that when increasing the number of threads per block, the way of scheduling these blocks in each SM was not fixed, as shown in Figure 4.5. The number of blocks that were allocated to the SM decreased when the active threads per block percentage were increased.

To find the significance of the relationship between power consumption and the blocks number per SM, we use the following regression equation:

$Y = -7.2069x + 157.82$ (5)

Where y is power consumption in Fermi GPU and x is the number of blocks per SM. The p-value is 0.01, which confirms that the relationship between power consumption and the blocks number per SM is statistically significant.

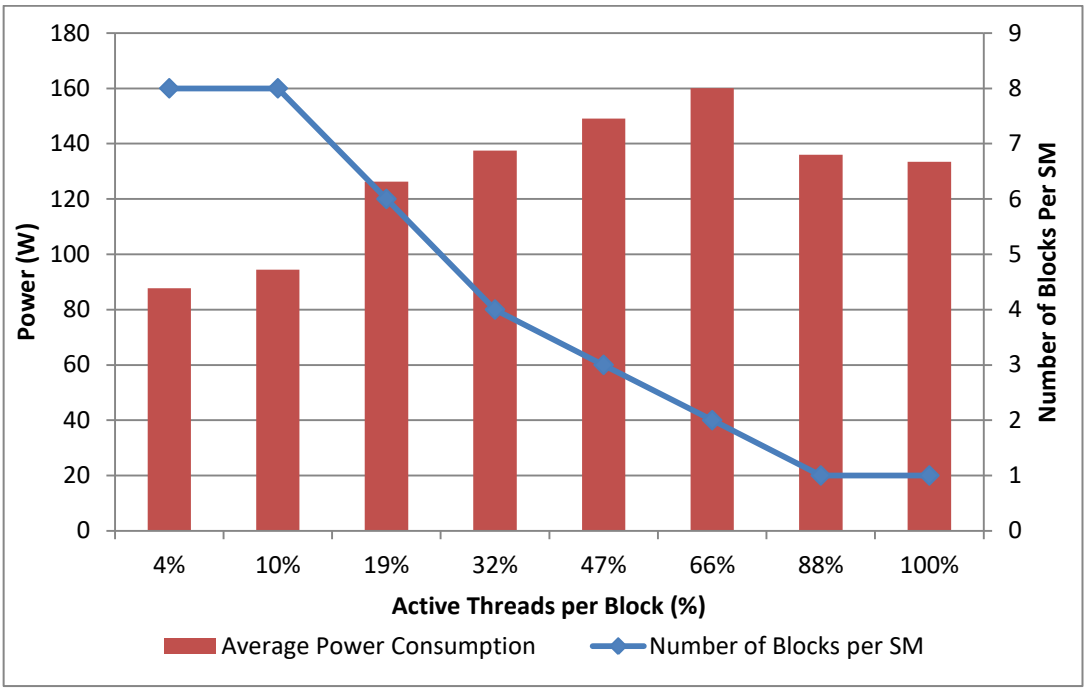


Figure 4.5: The Number of Blocks per SM in Fermi C2075 GPU

After calculating the GPU Occupancy for each workload (in Figure 4.6), we found that power consumption was affected by the GPU Occupancy. Even by increasing the size of memory and the number of threads per block in the Fermi C2075 GPU, power consumption had a relationship with the GPU Occupancy value.

To find the significance of the relationship between power consumption and GPU occupancy in Fermi GPU, we use the following equation:

$Y=63.6+86.1x$ (6)

Where y is power consumption in Fermi GPU and x is GPU occupancy. After using the previous equation, the p-value is 0.02, which confirms that the relationship between power consumption and GPU occupancy is statistically significant.

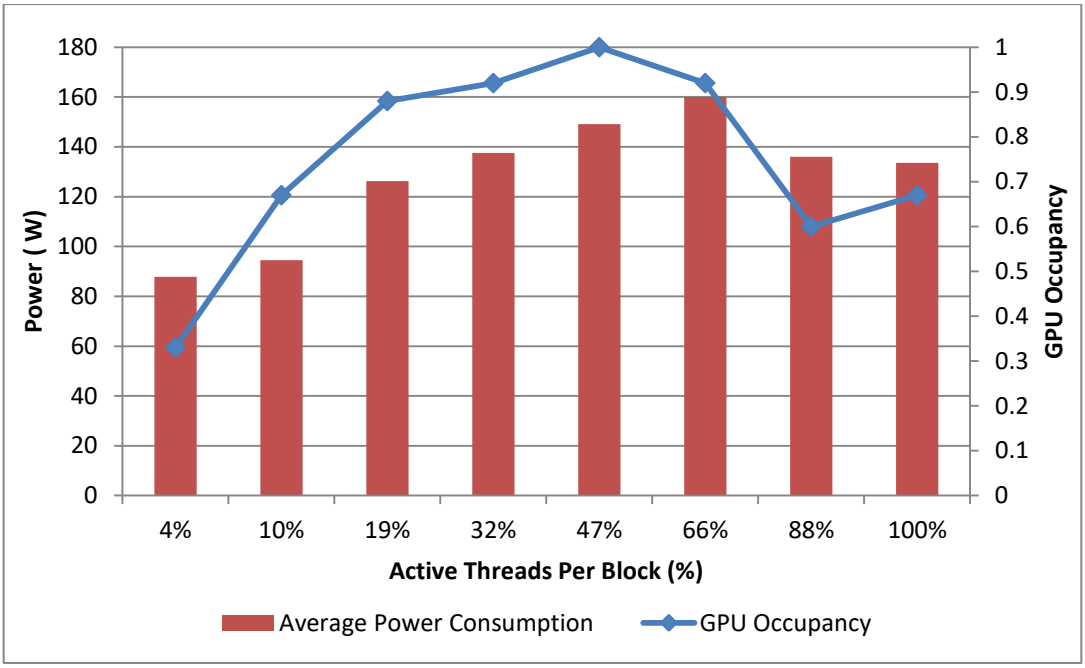


Figure 4.6: The Power Consumption and GPU Occupancy Values in Fermi C2075 GPU

For Kepler K40c GPU, we found that there was a gradual increase in power consumption up to a certain level of the active threads per block percentage, up to the active threads percentage per block was 88%. Then, power consumption values were equal when the percentage of the active threads per block was increased. However, GPU Occupancy has a remarkable impact on performance. Even the memory size of the 2560 x 2560 matrix is greater than the memory size of the 2400 x 2400 matrix, the execution time of 2560 x 2560 is lower than the execution time of 2400 x 2400, as shown in Table 4.4.

For Kepler K40c, after profiling the exact performance counters that were used in Fermi C2075 GPU especially with memory behaviour counters, we found that some of these counters values were correlated with the power consumption trend and the value of these performance counters were decreased when the workload was increased, more specifically in the maximum matrix size (2560 x 2560), as shown in Table 4.6. These counters were: gst_transactions, l2_l1_read_hit_rate and l2_read_transactions. Even the size of the 2560 x 2560 matrix was larger than the size of the 2400 x 2400 matrix, the values of the aforementioned counters in the 2560 x 2560 matrix were smaller than the values of the counters in the 2400 x 2400 matrix.

Table 4.6: Performance Counters values of the Memory types in Kepler K40c GPU

Counter Name	Counter Description	Counter Value (2080x2080)	Counter Value (2400x2400)	Counter Value (2560x2560)
gst_transactions	Global Store Transactions	411200	522000	204800
dram_read_throughput	Device Memory Read Throughput	9.8835GB/s	7.8928GB/s	9.4082GB/s
dram_write_throughput	Device Memory Write Throughput	130.00MB/s	95.954MB/s	113.87MB/s
l2_l1_read_hit_rate	L2 Hit Rate (L1 Reads)	97.29%	97.78%	97.41%
l2_read_transactions	L2 Read Transactions	1747268239	2830189274	2621517916

We then analysed the effectiveness of scheduling the blocks into SMs behaviour on the power consumption in Kepler K40c GPU, as shown in Figure 4.7. When increasing the number of threads per block, the way of scheduling these blocks was not fixed. The number of blocks allocated to the SM decreased when the number of threads per block was increased. We apply the following regression to find the significance of the relationship between power consumption and the blocks number per SM in Kepler GPU:

$$Y=-3.7X+131.1 \quad (7)$$

Where Y is power consumption, and X is the blocks number per SM in Kepler GPU. After the previous equation was applied, the p-value is 0.0001 which is statistically significant.

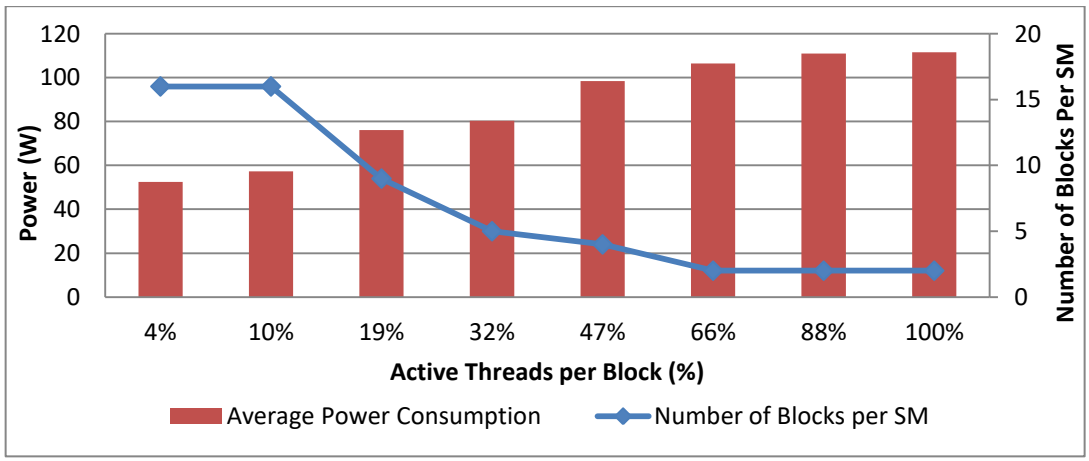


Figure 4.7: The Number of Blocks per SM in Kepler K40c GPU

For Kepler K40c, we found that the GPU Occupancy values in every workload were greater than or equal to 0.5. Therefore, GPU Occupancy was not a sufficient enough indicator to explain the power consumption trend, as shown in Figure 4.8. To measure the significance of this relationship, we apply the following regression equation:

$Y = 49.9+42.3X$ (8)

Where Y is power consumption and X is the GPU occupancy in Kepler GPU. the p-value in this relationship is 0.42 which is not significant.

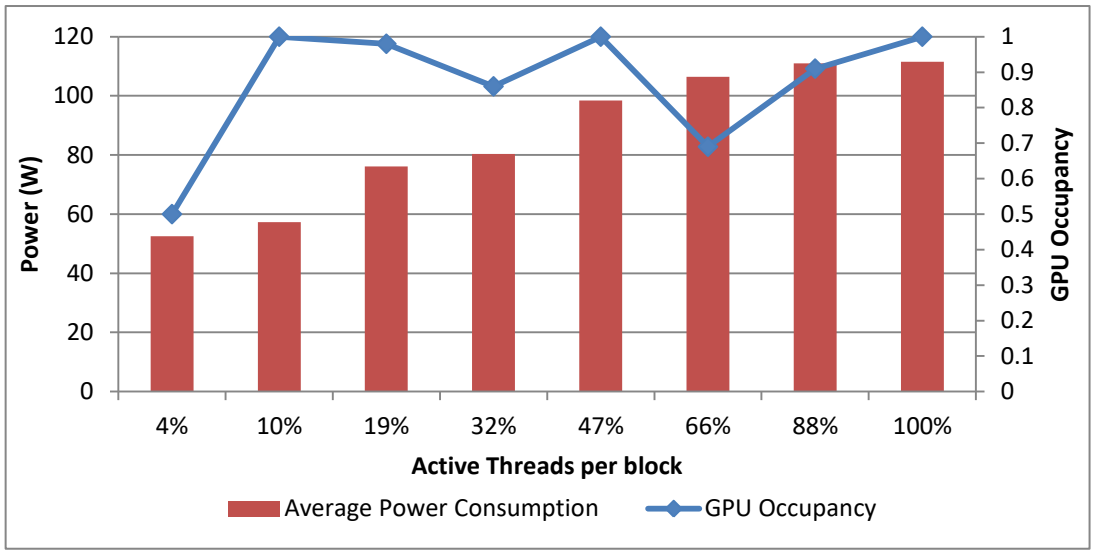


Figure 4.8: Power Consumption and GPU Occupancy Values in Kepler K40c GPU

4.5.3 The Blocks and the Threads per Block allocations Impact on the Energy Consumption

The aim of designing this experiment is to explore the impact of the blocks and the threads per block allocations on energy consumption in heterogeneous GPUs architectures.

The matrix multiplication size (2400 x 2400) after dropping the power consumption was selected to analyse the impact on the energy consumption in the experiment introduced in 4.5.2 since it can be implemented with different workload size. We implemented the same matrix multiplication size (2400 x 2400) with different workload allocations (a different number of blocks and threads per block). The first implementation had 100 x 100 blocks and 24 x 24 threads per block. The second implementation had 80 x 80 blocks and 30 x 30 threads per block. Then, we

calculated energy consumption (Joules) by multiplying the execution time in Seconds (S) and power consumption in Watts (W). Table 4.7 shows the execution time and the energy consumption of these matrices in Fermi C2075.

Table 4.7: The Execution time and the Energy Consumption of the Same Matrix size in Fermi C2075 GPU

Matrix Size	Number of Blocks	Number of Threads per Block	Average Power(W)	Average Execution Time (S)	Energy(J)
2400x2400	100x100	24x24	158.88	0.51102	81.19
2400x2400	80x80	30x30	136.03	0.56202	76.45

We found that there was an energy saving of 5.8% in the matrix that had a larger number of threads per block (30x30) and 9.1% in performance loss.

The second scenario checked the effectiveness of the execution time on the performance and energy consumption. The block size was increased to five times larger than the block size in the previous experiment for both the matrices to increase the execution time and to investigate the behaviour when the execution time was increased. The experiment was repeated five times for each implementation and calculated the average of the power consumption and the execution time as well, as shown in Table 4.8.

Table 4.8: The Execution Time and the Energy Consumption of the Same Matrix Size in Fermi C2075 by increasing the Number of Blocks

Matrix Size	Number of Blocks	Number of Threads per Blocks	Average Power(W)	Average Execution Time (S)	Energy(J)
12000x12000	500x500	24x24	179.439	68.355	12265.55
12000x12000	400x400	30x30	147.138	72.427	10656.76

Additionally, even by increasing the execution time in the matrix that had 24 x 24 number of threads per block, there was a 13.1% energy saving with the matrix that had 30 x 30 number of threads per block, which was similar to the previous experiment behaviour and had a lower execution time. In this case, by increasing the execution time, the performance loss decreased to 5.6% compared to the previous case.

Thus, in Fermi C2075 GPU, the energy consumption reduction moved towards the blocks and threads per block allocation which had a lower power consumption.

Therefore, there is an affordable trade-off between energy consumption and performance in Fermi C2075 GPU in this case. Moreover, the tradeoff between energy consumption and performance reduced when increasing the execution time.

Subsequently, the exact matrix multiplication size and the same way of calculating energy consumption in the previous GPU was implemented in Kepler K40c GPU, as shown in Table 4.9.

In Kepler K40c, we found the opposite situation: there was an energy consumption saving of 9.1% in the matrix that had a faster execution time and a larger number of blocks (100 x 100) since there was no substantial difference in power consumption between the first and the second workload allocation. The power consumption difference was merely 3.08 Watts between them.

Table 4.9: The Execution Time and the Energy Consumption of the Same Matrix in Kepler K40c GPU

Matrix Size	Number of Blocks	Number of Threads per Block	Average Power(W)	Average Execution Time (s)	Energy(J)
2400x2400	100x100	24x24	114.08	0.33763	38.51
2400x2400	80x80	30x30	111	0.38215	42.41

Then, the second scenario was to increase the block size five times larger than the block size in the previous experiment for both matrices to increase the execution time, as shown in Table 4.10. The matrix size that had 24 x 24 threads number per block was 11.2% more energy efficient.

Thus, in Kepler K40 GPU, the energy consumption reduction moved toward the blocks and threads per block allocation which had a fast execution time. Therefore, this experiment can make developers, when they develop GPU applications, aware of selecting energy-aware blocks and threads per block number allocation based on the GPU architecture.

In this experiment, it was observed that Kepler K40c GPU was 46.5% more energy-efficient than Fermi C2075 GPU.

Table 4.10: The Execution Time and the Energy Consumption of the Same Matrix size in Kepler K40c GPU by increasing the Number of Blocks

Matrix Size	Number of Blocks	Number of Threads per Blocks	Average Power(W)	Average Execution Time (s)	Energy(J)
12000x12000	500x500	24x24	141.213	40.366	5700.20
12000x12000	400x400	30x30	140.695	45.672	6425.82

4.5.4 Temperature Impact on Power Consumption

The design of this experiment aims to explore the temperature impact on the power consumption in Fermi C2075 and Kepler K40c.

We increased the size of the matrix and the size of the blocks to 1000 x 1000. Also, the threads number was configured with 20x20. We executed a 2000 x 2000 matrix multiplication application on both GPUs (Fermi C2075 and Kepler K40c), as shown in Figures 4.9 and 4.10, respectively. Power consumption and temperature were profiled every five seconds since a large execution time is used in this experiment.

We found that there was a linear increment in power when the temperature was increased in both GPUs. However, there was a decrease in power consumption in Fermi C2075 GPU at a certain level during the run time from 186.98 W to 179.7 W, but the power consumption was increasing after finishing this decrease, as shown in Figure 4.9.

Similar to Fermi GPU, power consumption in Kepler K40c GPU was decreasing after a continuous power increase at a certain level during the run time from 155.56 W to 152.84 W, but the power consumption was increasing after finishing this decrease, as shown in Figure 4.10.

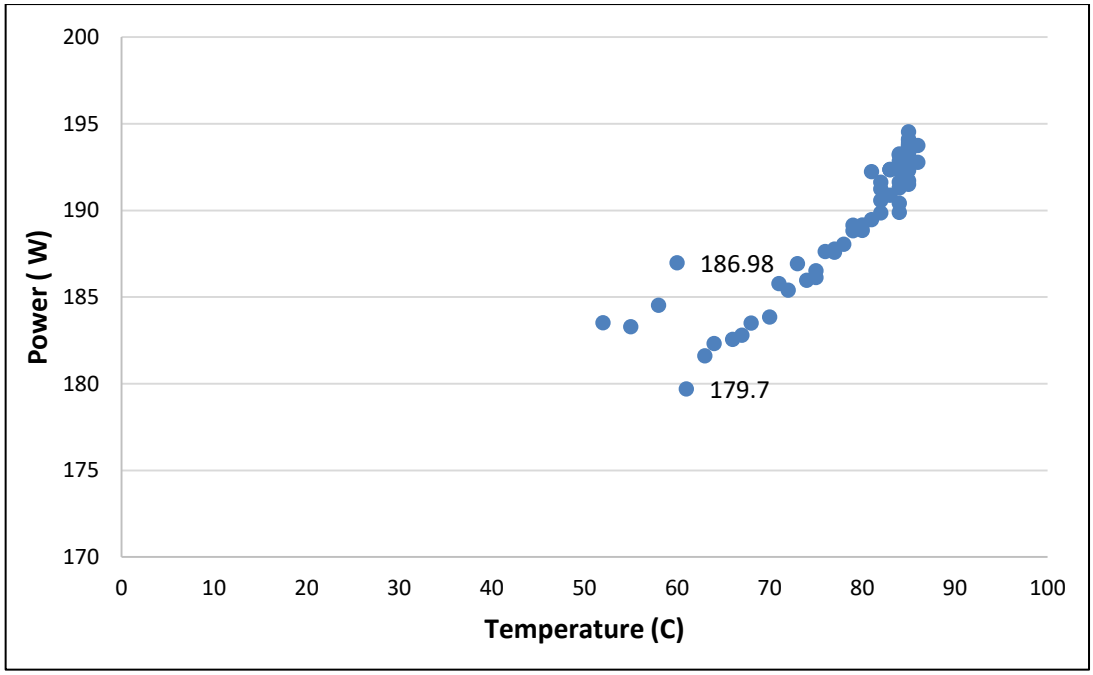


Figure 4.9: Power Consumption and Temperature in Fermi C2075 GPU

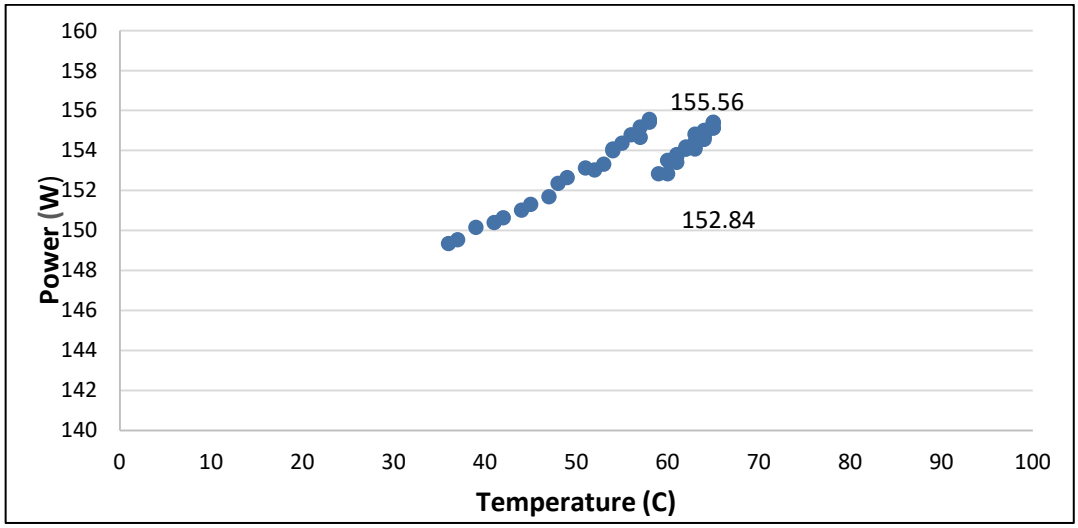


Figure 4.10: Power consumption and Temperature in Kepler K40c GPU

GPU memory utilisation which profiled alignment with power consumption by the nvidia-smi management tool could have influenced the occurrence of this resistance. Figure 4.11 depicts GPU memory utilisation behaviour with temperature and power consumption in Kepler K40c GPU.

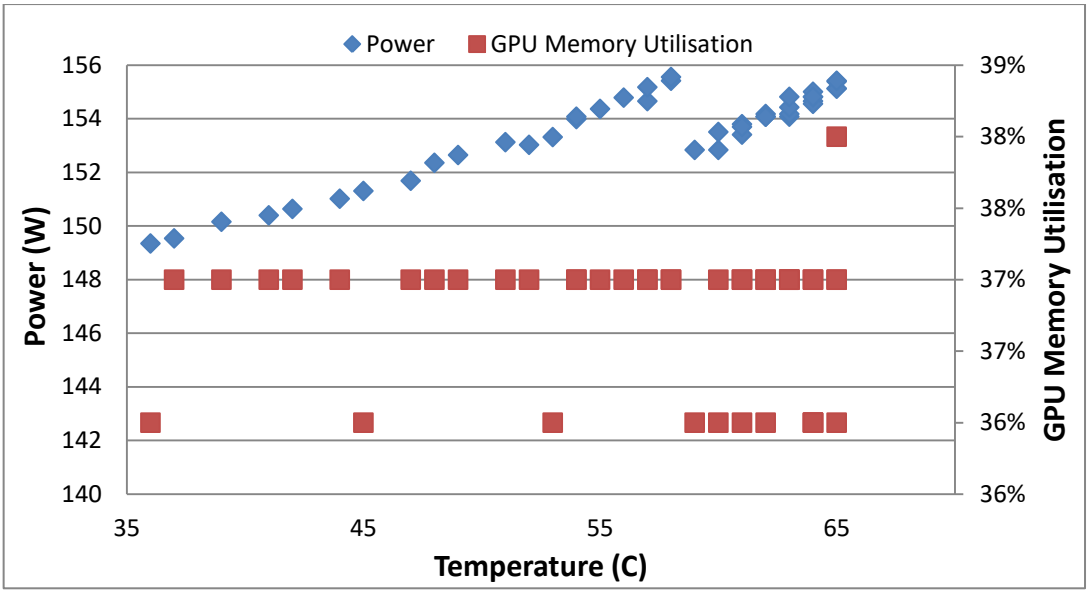


Figure 4.11: Power consumption, GPU Memory Utilisation and Temperature in Kepler K40c GPU

4.6 Overall Discussion

According to Figures 4.3 and 4.4, the non-linear relationship between the workload and power consumption in both GPUs (Fermi and Kepler) is more statistically significant than the linear relationship since the R^2values of quadratic regressions are greater than the linear ones. Moreover, the non-linear relationships between the workload and power consumption in both GPUs are more statistically significant. In Fermi GPU, The *p*-value of the non-linear relationship is 0.002, but the *p*-value in the linear regression is 0.082 which is not significant. In Kepler GPU, the *p*-value in the non-linear relationship is 0.000008 while the *p*-value in the linear one is 0.0001.

In Fermi C2075, raising the block numbers lead to an increase in hardware resource usage. Thus, the number of resident blocks' in the SM is decreased from 8 to 1. This reduction produces inefficient parallelism behaviour to cover the instruction pipeline and memory latency. Therefore, it leads to a performance decrease. This performance decrease affects power consumption. Similarly, GPU Occupancy decreases alongside the resident blocks in the SM leading to a performance and power decrease. However, in Kepler K40c, although the resident blocks' number in the SM is decreased from 16 to 2, the performance of the matrix with the size 2560 x 2560, which is the largest matrix size, is not affected by this reduction, and its execution time is faster than the previous matrix, 2400 x 2400. The reason lies in

the effectiveness of the GPU Occupancy on performance since the GPU Occupancy of 2560 x 2560 is greater than 2400 x 2400.

GPU memory types have an impact on the power consumption in Fermi GPU as sometimes affect the power consumption in the Kepler GPUs.

GPU Occupancy, GPU memory types and hardware block scheduling factors have a strong correlation with power consumption in Fermi C2075 GPU. However, the effectiveness of GPU Occupancy on power consumption in Kepler K40c GPU does not exist. However, it has a clear effect on performance. The remaining factors, some GPU memory types and block scheduling, can be considered in terms of effectiveness on power consumption.

Moreover, blocks and threads per block allocations affect energy consumption. The impact depends on the type of GPU architecture. In Fermi C2075 GPU, there is a trade-off between performance and energy consumption. Increasing the number of blocks will increase performance and also increase energy consumption. However, in Kepler K40c GPU, increasing the number of blocks will increase the performance and become more energy efficient.

Finally, the temperature has a strong impact on power consumption for both Fermi and Kepler GPU architectures when the execution time is increased.

To sum up, dealing with heterogeneous hardware GPUs are complex and not a straightforward process since each GPU has different characteristics and features; therefore, CSPs should consider these features carefully in resource management development.

4.7 Summary

In this chapter, a systematic architecture has been proposed in a Cloud computing environment. This architecture aims to manage heterogeneous GPUs resources for general purpose usage in Cloud computing environments. The architecture has considered the deployment and the runtime by focusing on performance and energy consumption factors. The architecture's components and their interactions have been outlined. The Heterogeneous GPUs Analyser has been introduced as the initial step to develop the aforementioned architecture. The Heterogeneous GPUs Analyser is aimed at analysing the architectural behaviour of heterogeneous GPUs

in terms of performance, power and energy consumption by conducting several experiments in a Cloud testbed. Additionally, Kepler architecture was 46.5% more energy-efficient than Fermi architecture.

The Power and energy consumption prediction modeller will be discussed in the next chapter. This modeller aims to estimate power and energy consumed by the GPU applications when they are executed in Cloud environments.

Chapter 5 Power and Energy Models

5.1 Overview

In this chapter, the power and energy modeller component will be deeply discussed. The modeller will be divided into three main parts. These parts are the power consumption model, energy consumption model and the GPU end-to-end energy consumption framework. An introduction of this modeller will be presented in section 5.2. Then, the way of designing power consumption, the GPU end-to-end energy consumption framework will be illustrated in sections 5.3, 5.4 and 5.5., respectively. Moreover, the implementation technique and models results will be shown in sections 5.6 and 5.7, respectively. Finally, the discussion of the results will be presented in section 5.8.

5.2 Introduction

The proposed architecture aimed to manage heterogeneous GPUs in Cloud computing environments considering energy and performance has been introduced in Chapter 4. In this chapter, power and energy modeller, highlighted in red, will be focused, as shown in Figure 5.1.

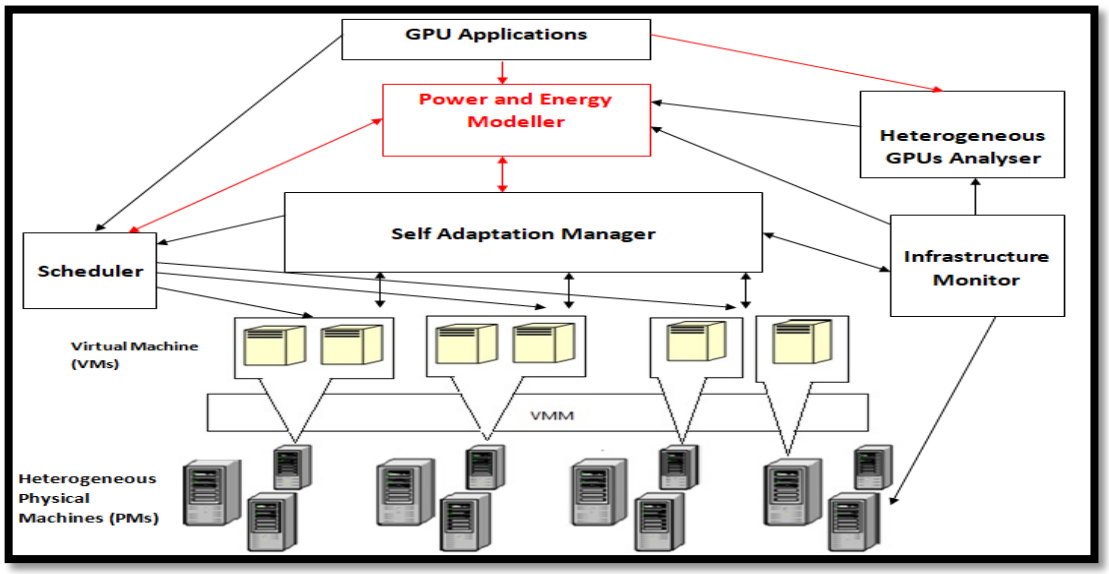


Figure 5.1: The Proposed Architecture with Power and Energy Modeller (Highlighted in Red)

The objective of the power and energy modeller is to predict power and energy consumption when GPU applications are executed on VMs in Cloud computing environments through the mathematical modelling technique.

We separately develop GPU power and energy consumptions for some reasons. GPU Power and energy consumption can be beneficially used for different purposes. GPU Power consumption models are used to develop and optimise GPU architectures for power consumption [115]. GPU energy models are utilised for energy-aware GPU resource management [24]. Unlike power consumption, energy consumption is affected by other factors, such as execution time and temperature for long execution time, as shown in section 4.5.4. Moreover, we develop GPU power consumption for applications with short execution time like [105] and develop GPU energy consumption for applications having long execution time for measurable energy consumption.

Cloud environments consist of heterogeneous infrastructures. The heterogeneity type in this research deals with heterogeneous GPUs, and each of which contains different characteristics and capabilities with different generations. The used GPUs for power and energy modeller are Nvidia Fermi C2075 and Nvidia Kepler K40c.

5.3 Power Model

This section illustrates the methodology of developing the power consumption model in details.

5.3.1 Applications

The power model involves the training phase that contains collected data of several GPU applications to develop, train and testing the model to evaluate the prediction accuracy of the developed model. Several GPU applications that contain different computational, communication characteristics and implementation techniques are selected to train, test the model and evaluate the strength of the model as well. Different GPU applications types including compute-intensive and memory-intensive applications are considered to develop power and energy models. N-body is an example of compute-intensive applications, and Spmv is an example of memory-intensive applications, chosen to harden the model. Selected applications are found in several fields, such as linear algebra, image processing, simulation and fluid dynamic. The main data set is divided into training and testing sets. The training phase contains 30 applications, and the testing phase contains 10 applications. The applications in the training and testing phases are selected from

well-known benchmarks in the field of GPU computing. The training set is larger than the testing for training the DNN model. These benchmarks are Rodinia [195], Parboil [196]and CUDA SDK applications [197]. Training and testing DNN models may need large data sets; but in this DNN model, we use a limited number of training and testing sets. This is because performing experiments in the Cloud testbed to collect data is significant time-consuming. We separately run two different profiling tools to collect data (nvprof and nvidia-smi), and nvprof has considerable overhead. Additionally, we consider two different GPUs to collects the required data for the model. Table 5.1 represents the application name, its source and the application's size in the training set. Table 5.2 similarly illustrates the details of the selected applications in the testing set.

Table 5.1: Training Set Characteristics in the Power Model

Application	Source	Description	Size
AlignedTypes	Cuda SDK	Memory aligned access implementation type	499,99,872 Bytes
Binomial Options	Cuda SDK	The European pricing options between seller and buyer	1024 Options
BlackScholes	Cuda SDK	The European pricing options using the Black-Scholes model	8,000,000 Options & 512 Iterations
Dxtc	Cuda SDK	DirectX Texture Compression Algorithm	512x512 pixels
ConvolutionSeparable	Cuda SDK	Convolution technique for image filtrations	3072x3072 Pixels
Histogram	Cuda SDK	Analysis tool for applications	64 & 256 bins
Transpose	Cuda SDK	Matrix Transpose implementation	1024x1024
FDTD3d	Cuda SDK	Three Dimensional Finite Difference Time Domain Model	376 x 376 x 376
MergeSort	Cuda SDK	Merge Sort implementation	4,194,304 elements
RadixSortThrust	Cuda SDK	Parallel Radix Sort implementation	1,048,576 elements
GuasirandomGenerator	Cuda SDK	Niederreiter Quasirandom Sequence implementation	31,148,576 elements

Application	Source	Description	Size
RecursiveGaussian	Cuda SDK	Gaussian blur implementation using Deriche's recursive way or Recursive Gaussian filter	512x512
HSOpticalFlow	Cuda SDK	a clear movement of objects estimation in a picture	640x480
Interval	Cuda SDK	Interval Newton Method Calculation	65,536 equations
SobolQRNG	Cuda SDK	Sobol Quasirandom Sequence implementation	1,000,000 vectors & 1000 dimensions
Reduction	Cuda SDK	Parallel reduction technique implementation	16,777,216 elements
ScalarProd	Cuda SDK	Scalar product implementation	2,048 vectors & 131,072 elements
StereoDisparity	Cuda SDK	Stereo Disparity computation implementation	1800x1800
ThreadFenceReduction	Cuda SDK	implementation of array reduction operation using Thread Fence instruction approach	1,048,576 elements; 128 threads; 64 blocks
Sgemm	Parboil	Single Matrix Multiplication implementation	AxB; A=2048x1984 , B= 1984x2112
Spmv	Parboil	Sparse Matrix Vector Multiplication implementation	146,689x146,689
Stencil	Parboil	3D Seven point Stencil implementation	128x128x32; 200 iterations
Heartwall	Rodinia	Ultrasound image for heart wall tracking	656x744 frame
Kmeans	Rodinia	Clustering algorithm	494,020 objects & 34 features
Gaussian	Rodinia	Gaussian elimination technique for solving equations	1024x1024
Leukocyte	Rodinia	Microscopy tracking of white blood cells	640x480 frames
Nw	Rodinia	Needleman-Wunsch method for optimised DNA alignment	6400x 6400
Hotspot	Rodinia	Simulation for estimating a processor temperature	4096 x 4096
Dwt2d	Rodinia	Two Dimensional Discrete Wavelet Transform Algorithm	3900x4200 Pixels
CFD	Rodinia	Computational Fluid Dynamic solver	97,000 Elements

Table 5.2: Testing Set Characteristics in the Power Model

Application	Source	Description	Size
ConvolutionTexture	Cuda SDK	Convolution technique using texture implementation	3072x1536 Pixels
FastWalshTransform	Cuda SDK	Fast Walsh Transform algorithm implementation	8,388,608 data length
MontCarlo	Cuda SDK	option pricing computation using Monte Carlo	8,192 options & 262,144 paths
Nbody	Cuda SDK	N-body simulation implementation	16,640 bodies
Cutcp	Parboil	Cutoff-limited Coulombic Potential computations implementation	96,603 atoms
Histo	Parboil	Gathering the number of occurrence for each element	256×8192
Mri-q	Parboil	Magnetic Resonance Imaging restoration in 3D	64x64x64
Bfs	Rodinia	Breadth First Search implementation	1,000,000 nodes
Streamcluster	Rodinia	Solving clustering problems	65,536 points
Srad_2	Rodinia	Speckle Reducing Anisotropic Diffusion algorithm implementation	8192x8192 data points

5.3.2 Data Collection

A built-in nvidia-smi [193] profiling tool to collect power consumption, with a 20 ms time interval for both Fermi and Kepler GPUs is used. A built-in nvprof [194] profiling tool is used to collect the hardware performance counters of the GPU architecture components by querying the counter name during executing the application in each GPU for 40 real applications in the training and testing sets. An appropriate applications size is selected for suitable power consumption measurements, as shown in Table 5.1 and 5.2 since Kepler GPU has a better performance than the Fermi GPU, as demonstrated in section 4.5.3**.** After measuring power consumption for each application, the average power consumption is calculated. Nvprof tool separately profiles performance counters of each kernel, and some applications have more than one kernel. If the application

has more than one kernel, the average value is calculated to unify all counters values.

5.3.3 Input Selection

The models' inputs mainly rely upon hardware performance counters since they have an adequate indication of power consumption. Selecting appropriate model inputs can increase the model's accuracy. Each generation of GPU architecture has a different number of hardware performance counters than others. Nvprof tool can profile 98 and 111 performance metrics in Fermi C2075 and Kepler K40c GPUs, respectively [198] [199]. Using all existing performance counters to feed the model can increase the model's accuracy. However, considering all existing performance counters as model inputs will increase the model's training complexity and the cost of the data collection because of the overhead incurred by the usage of the profiling tools. Therefore, we develop the candidacy algorithm to find the performance counters that have a reasonable impact on power consumption in each GPU (Fermi and Kepler), as shown in Algorithm 5.1. It is difficult to statistically compute the significance between each performance counter and power consumption differing from zero in two different GPUs because of the large number of performance counters in each GPU. Thus, we follow [200], [107] and [26] to develop the candidacy algorithm by using the correlation analysis to find the collection of hardware performance counters that have a reasonable correlation with power consumption in each GPU.

The candidacy algorithm consists of two main phases is developed. Phase 1 aims to find the main performance counters in each GPU that have an impact on power consumption. The experiment described in section 4.5.2 is used to extract essential counters. In the CUDA programming language, there are several implementations to handle the GPU memory types, such as global, shared and texture memories. The basic implementation for dealing with GPU memories is global memory, which is used in phase 1. In this phase, we aim to reduce the nvprof profiling tool overhead since we found some of the training set applications have a significant time delay when the profiling tool is executed. The output of phase 1 is used as an input for phase 2. Phase 2 aims to find the performance counters that have a great impact on power consumption on applications located in the training set.

The algorithm's inputs are all existed performance counters in each GPU (Fermi and Kepler) and all the applications considered in the training set. The algorithm's output is performance counters set that have a great impact on power consumption that will be used to train the model. In phase 1, the matrix size is gradually increased starting from 480x480 up to 2560x2560 with a regular increase, to analyse the number of the threads per block in terms of power, as described in section 4.5.2. This increase creates 8 different matrix sizes. Then, power consumption and the performance counters values of each matrix size are profiled. Once the profiling of power consumption and values of certain performance counters in all the matrices is completed, the level of strength between them is measured. Pearson correlation coefficient is applied to evaluate the strength of the relationship between power consumption and the performance counter values. Pearson correlation coefficient aims to measure the linear correlation between two variables sets *X* and *Y* between -1 and 1. When the absolute value of the coefficient is 1 or close to 1, it means that there is a linear high correlation. In [200], the authors selected the top 10 performance counters that have a strong correlation with GPU power consumption. However, since we consider two heterogeneous GPUs containing different features, we initially selected the hardware performance counters that are greater or equal than 0.5 as a predefined threshold that represents the middle of the value between 0 and 1 [201], [202]. After applying the Pearson correlation coefficient, a threshold is set to select the Pearson coefficient Pt_c (line 14). The outcome of phase 1 is FC set which includes the essential performance counters metrics that greater than the assigned 0.5, and the FC set will be used as an input for the second phase.

Similar to phase 1, power consumption and the performance counter value of each application in the training set that contains 30 applications are profiled. After profiling power consumption and performance counters values, another Pearson correlation coefficient $Pt2_c$ is applied between the counter value and profiled power consumption in all applications in the training set. After applying the correlation analysis in phase 2, we found that there is a low overlapping in the correlation relationship between the selected performance counters and power consumption, as shown in Tables 5.3 and 5.4. The reason behind this low

overlapping in phase 2 is that we applied the Pearson correlation coefficient on 30 applications with different size and different characteristics to find the strength of the relationship between each selected performance counter and power consumption. Applying this correlation analysis on 30 applications containing different features and sizes will show us how strong the correlation is, and this correlation analysis with 30 applications will give us a clear and more confident view about the relationship between each selected performance counter and power consumption, unlike applying the correlation analysis on only one application with a regular increase, such as phase 1 or only using 10 applications like [26]. Therefore, Ω is calculated to find the average of the correlation analysis values of all selected counters in each GPU and used as an appropriate threshold like [203] and [204] using the following equation:

$$\Omega = \sum_{c=1}^{n} \left| \frac{\mathrm{p}}{\mathrm{n}} \right| \qquad (1)$$

Where P is the value of every performance counter and n is the total number of performance counters in phase 2 in every used GPU. So Ω aims to calculate the average of all performance counters in phase 2 for every GPU after using the absolute value. The value of Ω is 0.232 and 0.087 for Fermi C2075 and Kepler K40c, respectively. Then, Ω is assigned to be a threshold value to camper it with each selected counter in phase 2. If the Pearson correlation coefficient $Pt2_c$ is greater than or equal to the threshold value Ω (line 32), the performance counter metric will be added to the PC set. Otherwise, the metric will be rejected. The candidacy algorithm applies to the performance counters twice on Fermi and Kepler GPUs performance counters.

Algorithm 5.1: Candidacy algorithm

Input: C [ ], A [ ]

Output: HC [ ]

1: set HC [ ], P1 [ ], P2 [ ], FC [ ] = null

2: m [ ] = 480x480

3: **begin phase 1**

4: **for each** c in C [ ] **do**

5: **for** (i=1; i<=8; i++) **do**

6: set PC1 [ ] = null
7: increase m (the matrix size) gradually
8: profile average p_i (power consumption) in every matrix size
9: profile c_i (performance counter) in every matrix size
10: Enqueue p_i into P1 [ ]
11: Enqueue c_i into PC1 [ ]
12: **end for**
13: Apply the Pearson Correlation test Pt_c between P1 and $PC1_c$
14: **if** |Pt_c | >= 0.5 **then**
15: Enqueue c_i into FC [ ]
16: **else**
17: Reject c_i
18: **end if**
19: **end for**
20: **end phase 1**
21: **begin phase 2**
22: **for each** c in FC [ ] **do**
23: **for** (i=1; i<=30; i++) **do**
24: set PC2 [ ] = null
25: profile average p_i (power consumption) in every application
26: profile c_i (performance counter) in every application
27: Enqueue p_i into P2 [ ]
28: Enqueue c_i into PC2 [ ]
29: **end for**
30: Apply Pearson Correlation test $Pt2_c$ between P2 and $PC2_c$
31: calculate Ω
32: **if** |$Pt2_c$ | >= Ω **then**
33: Enqueue c_i into HC [ ]
34: **else**
35: Reject c_i
36: **end if**
37: **end for**
38: **end phase 2**

According to the Candidacy algorithm's outcomes, there are 9 and 6 selected counters metrics for the model inputs for Fermi C2075 and Kepler K40c, respectively. Performance counters metrics names, their description and the Pearson correlation coefficient in phase 2 $\boldsymbol{Pt2_c}$ are shown in Tables 5.3 and 5.4 for Fermi and Kepler GPUs, respectively.

Table 5.3: Selected Performance Counters for Fermi C2075

Metric Name	Description	$\boldsymbol{Pt2_c}$ Value
Gst_transactions	transactions of Global memory store number	0.281
Ecc_transactions	Error-Correcting Code memory transactions sent between L2 cache memory and DRAM memory number	0.421
L2_read_transactions	L2 cache memory read transactions number	0.236
L2_write_transactions	L2 cache memory write transactions number	0.281
L2_L1_read_transactions	Memory read transactions requested by L1 cache memory and seen in L2 cache memory number	0.310
L2_L1_write_transactions	Memory write transactions requested by L1 cache memory and seen in L2 cache memory number	0.310
Eligible_warps_per_cycle	The average number of eligible wraps to be issued in every cycle	0.291
DRAM_read_transactions	The GPU Device RAM read transactions number	0.293
DRAM_write_transactions	The GPU Device RAM write transactions number	0.284

Table 5.4: Selected Performance Counters for Kepler K40c

Metric Name	Description	$\boldsymbol{Pt2_c}$ Value
Gld_transactions_per_request	The average number of transactions of the global memory load conducted for every global memory request	-0.187
Gld_transactions	transactions of Global memory load number	0.102
Inst_per_warp	The average number of instructions run by every warp	0.165
DRAM _read_transactions	The GPU Device RAM read transactions number	-0.090
IPC	Instructions performed per cycle number	0.275
Issued_ IPC	Issued number of Instructions for every cycle	0.214

5.3.4 Model Design

Some studies such as [105] have confirmed that the linear regression method is appropriate for old GPU architectures in non-Cloud environments. Nevertheless, another study in [107] conducted in non-Cloud environments has argued that the linear models can fit the complexity of modern GPUs. Moreover, according to the conducted experiment in section 4.5.2, the relationship between the workload and power consumption in both GPUs tends to be non-linear. Therefore, DNN is selected to predict power consumption for heterogeneous GPUs connected with VMs since DNN is a non-linear model and shows a better performance than other machine learning algorithms [205], [97].

Since every model's input has a different range of value scale, the collected data in each input attribute has been pre-processed by using the normalisation technique. The selected normalisation technique is the Z score since it is the most popular normalisation technique [206]. The aim of using the normalisation technique is to simplify the procedure of model training, to adjust all inputs' attributes scale in a unified scale and reduce the impact of variation in the model training phase. Z score is represented by the following equation:

$$Z = \frac{X-\mu}{\sigma} \qquad (2)$$

Where X is the element value; in this case, it will be the counter value for every application, μ is the mean of all applications values for a certain selected counter in the training set and σ is the standard deviation of a single selected counter in Tables 5.3 and 5.4. We similarly use the process of μ to calculate σ. Then, we calculate Z for the remaining applications on others selected counters. After the inputs are pre-processed, they will be feed for the model's training stage.

In the feed-forward networks, the input layer is responsible to receive the model's inputs, in this case, the selected performance counters of each GPU. Then, the input layer passes received inputs to the hidden layer for processing. Finally, the processed data will be transferred to the output layer to accomplish the model task, as shown in Figure 5.2.

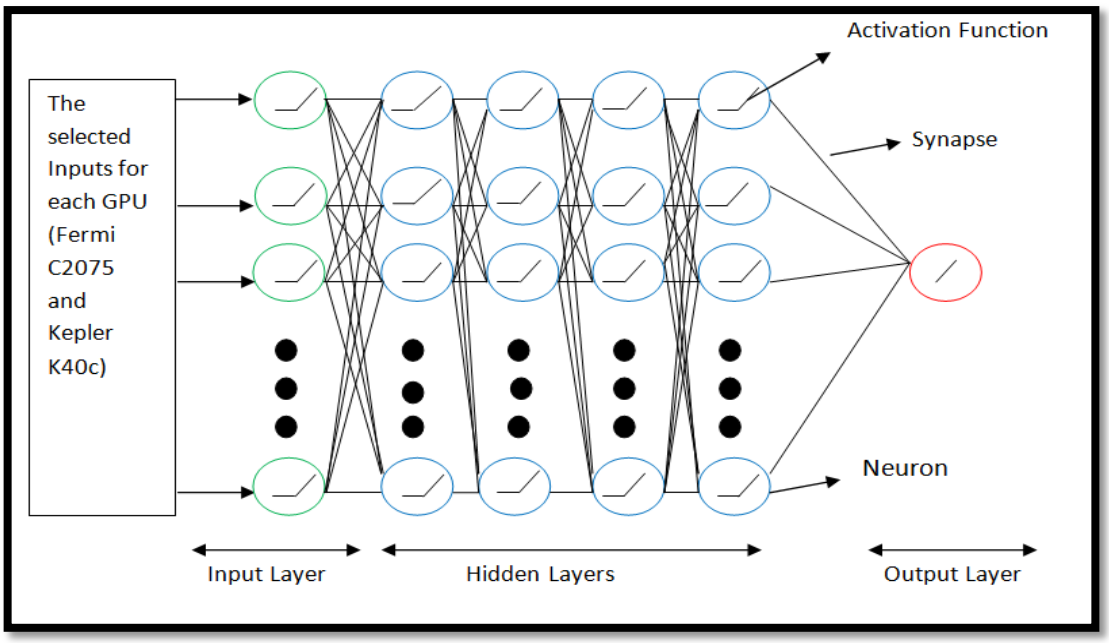


Figure 5.2: The DNN Structure of Power Consumption Models

The model aims to develop a function f that maps X inputs set to Y outputs set $f:(X \rightarrow Y)$ to estimate the power consumption P of a GPU (Fermi and Kepler) connected with a VM. Two power models are developed for Fermi and Kepler GPUs. For the Fermi power model, the input layer has 9 neurons representing every selected input, as shown in Table 5.3. After performing some experiments to find the adequate number for hidden layers and neuron numbers in every layer, we found 4 hidden layers and 8 neurons in each hidden layer is an appropriate number. Therefore, the model contains 4 hidden layers to enable layers to extract features from data, and each hidden layer contains 8 neurons; moreover, the output layer contains 1 neuron representing the predicted power consumption.

For the Kepler GPU power model, the input layer consists of 6 neurons indicating the selected inputs for Kepler GPU, as shown in Table 5.4. Similar to the Fermi Power model, the Kepler model has 4 hidden layers and each hidden layer has 8 neurons. Finally, the output layer contains 1 neuron to represent the predicted power consumption.

Suppose $x \in X = \{ x_1, x_2, x_3, \dots, x_n \}$ is a selected input variable and w_{ij} represents a certain Synapse weight S in each layer j which contains i neurons. Every S represents the summation of the product of all w_{ij} with x to every neuron by using the following equation:

$$s_i = \sum_j w_{ij} x_i \qquad (3)$$

After obtaining s for every neurone, the value will be passed to the activation function f(s_i) by using the appropriate non-linear function. In this model, the Rectified Linear Unit (ReLU) function is used as an activation function. ReLU activation function has a better performance than other activation functions such as Sigmoid and Tanh [207], and it has better behaviour for DNNs training. ReLU is represented in the following formula:

$$\mathrm{f}(s_i) = \begin{cases} s_i & s_i > 0 \\ 0 & \mathrm{otherwise} \end{cases} \quad (4)$$

This means that if the value of s_i is greater than zero, the s_i value will be assigned as the function's output in the neuron. Otherwise, the output of the neuron will be 0. Since ReLU cannot be an activation function for the output layer, the activation function in the output layer s_o is the linear activation function representing by the following equation:

$$\mathrm{f}(s_o) = s_o \quad (5)$$

In DNN, to find the output of every layer l and map them together, the following formula can be used to estimate power consumption P_d:

$$P_d = \mathrm{f}(W, X, b) \quad (6)$$

Where *W* is the weight matrix in every layer, *X* is the input features matrix in every layer, *b* is the bias in every layer. [208] and [97] are useful for more information about DNNs mathematical formulations.

In DNN, it is beneficial to randomly initialise weights by small values to avoid the similarity of the values among layers and to rapidly train the model [96]. More specifically, the weights of the synapses in the input layer are randomly initialised under the uniform distribution. The synapses weights in the hidden and output layers are randomly initialised under normal distribution, and bias values are initialised by zero. Moreover, the performance of machine learning algorithms such as DNNs models is usually affected by their hyper-parameters values [209]. Hyper-parameters are the parameters that their values establish before starting training the model. Some DNN models can have a range of hyper-parameters between ten to fifty based on the model's developer who set certain hyper-parameters and keep others as defaults, and hyper-parameters should be tuned carefully by experiments since it is a challenging task [210]. Some hyper-parameters are

adjusted in this model, such as learning rater, number of epoch and batch size. Learning rate is considered as the most substantial type of hyper-parameter [211]. Learning rate is the step size of every iteration to reduce the loss function during the training phase[212] , and the suggested value is 0.01 number of Epochs which is defined as the iteration number for the training algorithm. The assigned number of epochs is 1000. Batch size is the samples number in the training set used in every step for a faster training process, and it is assigned by 30. Table 5.5 shows the values of the assigned hyper-parameters.

Table 5.5: Hyper-parameters Values in Power Model

hyper-parameters	**Value**
Hidden Layer numbers	4
Activation Function	ReLU
Activation Function of the output layer	Linear
Learning Rate	0.01
Epochs	1000
Batch size	30

5.3.4.1 Model Training

After designing the model, the model training step is established. The training data set will be used to feed the training step. In general, the training process in neural networks is called back-propagation. The back-propagation process aims to adjust the biases and the weights of the model's layers to minimise the difference between the actual power P_{ac} and the output produced by the model P_d in each training step by using gradient descent [213], whereas the difference should be close to zero. The loss function L is used to measure the difference between P_d and P_{ac} in each training step by the following equation:

$$L\left(P_d^i, P_{ac}^i\right) = \frac{1}{2}\left(P_d^i - P_{ac}^i\right)^2 \qquad (7)$$

Where i is a certain training step. The DNN Training process is time consuming and hard task [214]. The standard gradient descent faces difficulties to train DNN layers

with random initialisation weights, however, gradient descent can work better if the weights are carefully selected [214]. This is because the difference of updated weights in every layer in DNN can be high in each training step [96]. Additionally, due to the flexibility of DNN models of creating many parameters in every layer, it will create complex nonlinear relationships among the DNN layers. Therefore, the Adam optimisation algorithm [215] is the selected algorithm for the model's training stage. Adam stands for Adaptive moment estimation. Adam is an extension of the classic stochastic gradient descent to update the edge weights. Adam has a better ability to train the model than other training algorithms, such as stochastic gradient descent and AdaGrad [216].

Next, to validate DNN models, a shallow neural network that contains only one hidden layer is developed for Fermi C2075 and Kepler K40c GPUs connected with VMs is to compare the performance of shallow and deep neural networks. The hyper-parameters for this model are identical to the DNN model except for the hidden layer which contains only one layer.

5.3.4.2 DNN with Hybrid inputs

Since the relationship between the selected performance counters and power consumption in both GPUs is not very correlated, we need another investigation to find other input factors that contain a better correlation. Therefore, another DNN model that contains hybrid inputs is developed. These inputs are a mix of input parameters. These inputs are the selected performance counters, GPU and memory utilisations, measured by percentage, to give a comprehensive overview of the GPU behaviour during running the applications for each GPU. Nvidia-smi tool is used to collect GPU and memory utilisations. When the Pearson correlation coefficient has been applied on GPU and memory utilisations, $Pt2_c$ values in Fermi GPU are 0.368 and 0.479, respectively. $Pt2_c$ values for GPU and memory utilisations in Kepler GPU are 0.653 and 0.872, respectively. After the Candidancy algorithm has been applied on GPU and memory utilisations, it can be observed that GPU and memory utilisations have an acceptable correlation with power consumption in both GPUs.

In terms of the model design stage, the number of hidden layers and the output layer is similar to the previous DNNs model that only uses performance counters, unless the number of neurons in the input layer is changed in both GPUs since input parameters are increased. For Fermi GPU, the number of neurons in the input layer is equivalent to the number of inputs. Therefore, the number of neurons in the input layer is 11 neurons. For the Kepler GPU, the number of neurons in the input layer is 8. The hyper-parameters for this model is similar to the previous DNN only performance counters model except the number of epochs is 300, as illustrated in Table 5.6.

Table 5.6: Hyper-parameters Values for Hybrid Inputs Power Model

hyper-parameters	**Value**
Hidden Layer numbers	4
Activstion Function	ReLU
Activation Function of the output layer	Linear
Learning Rate	0.01
Epochs	300
Batch size	30

5.4 Energy Model

This section illustrates the methodology of developing the energy consumption model in detail.

5.4.1 Applications

The applications that are selected for the energy consumption models are taken from Tables 5.1 and 5.2. The selected applications are capable to increase their sizes and being executed for longer periods of time to analyse the increase of the application's execution time trend with energy consumption. Each application has different patterns and characteristics. The applications' sizes are regularly and gradually increased until the VM cannot execute the application's size based on the GPU resources computing capabilities that connected to the VM or when the application's execution time reaches 1 hour and 40 minutes. The range of the

execution time is between 5 seconds up to 1 hour and 40 minutes. Each application has a different execution time, as shown in Figures 5.3 and 5.4. The selected applications in the energy model are:

Gaussian elimination is used for solving linear equations in the linear algebra field. Gaussian elimination application is a part of Rodinia benchmark applications. The application's size is increased from 4000 x 4000 to 20000 x 20000.

Hotspot is a simulation application that aims to estimate the temperature of the processor. Hotspot simulates the temperature of the processor based on the processor's architecture floor plan and simulating power consumption. It is a part of Rodinia benchmark applications. The range of the application's sizes is between 1024x1024 and 16384x16384.

Nbody is an application that continuously simulates the interaction of a certain body with others. An example of this application is an astrophysical simulation that contains several bodies which each body represents a star. Nnody is a part of CUDA SDK applications. The application's size is increased from 665600 to 3328000 bodies.

Srad is an application that aims to reduce the noise in the image without affecting the image content. Srad relies upon differential equations and is mainly used in radar and ultrasonic applications. Srad is a part of Rodinia benchmark applications. The application's size is increased from 3200x3200 to 14400x14400.

Streamcluster deals with data that continuously appear and cluster them in groups. Each data point is selected to a certain cluster based on the close distance between the point and the cluster. Streamcluster is a part of Rodinia benchmark applications. The range of the application's sizes is between 65536 and 4194304 points.

The main data set is divided into the training set and the testing set. The applications with their sizes are selected randomly. One application's size is selected from each application group to evaluate the model. Tables 5.7 and 5.8 show the applications in training and testing sets in the energy model, respectively.

Table 5.7: Training Set in the Energy Model

Application	Size
Gaussian	4000x4000
Gaussian	6000x6000
Gaussian	8000x8000
Gaussian	10000x10000
Gaussian	12000x12000
Gaussian	14000x14000
Gaussian	18000x18000
Gaussian	20000x20000
Hotspot	1024x1024
Hotspot	2048x2048
Hotspot	4096x4096
Hotspot	16384x16384
Nbody	665600
Nbody	998400
Nbody	1331200
Nbody	1664000
Nbody	1996800
Nbody	2329600
Nbody	2995200
Nbody	3328000
Srad	3200x3200
Srad	4800x4800
Srad	6400x6400
Srad	8000x8000
Srad	9600x9600
Srad	11200x11200
Srad	14400x14400
Streamcluster	65536
Streamcluster	131072
Streamcluster	262144
Streamcluster	524288
Streamcluster	2097152
Streamcluster	4194304

Table 5.8: Testing Set in the Energy Model

Application	Size	Abbreviation
Gaussian	16000x16000	G16000
Hotspot	8192x8192	H8192
Nbody	2662400	N2662400
Srad	12800x12800	S12800
Streamcluster	1048576	SC1048576

5.4.2 Data Collection and Input Selection

To obtain energy consumption, the application's execution time should be considered. Moreover, as seen in section 4.5.4, according to the experiments between power and temperature, it has been found that device temperature has a great impact on the power consumption when the execution time is increased. Thus, the device temperature also should be considered to enhance energy consumption estimation.

Nvidia-smi is used to profile the temperature with power consumption, GPU and memory utilisations with 1 second time interval. Then, the average temperature, power consumption, GPU and memory utilisations are calculated after profiling them for every application. Nvprof is used to profile the selected performance counters in the power model. However, nvprof is unable to profile some counters when the application's size and execution time are increased because an overflow happened when profiling these counters. Therefore, the remaining performance counters that nvprof can profile are used as the model's inputs. The remaining counters represent the behaviour of GPU memory types.

An agnostic energy model is also developed by selecting the inputs that share both GPUs (Fermi and Kepler). This agnostic energy model aims to normalise the difference and complexity of heterogeneous GPU architectures by selecting the common inputs between the used GPUs. Table 5.9 illustrates the model's inputs for every VM connected with a different GPU and the common inputs between them.

Table 5.9: The Models Inputs for Each VM and the Common Inputs

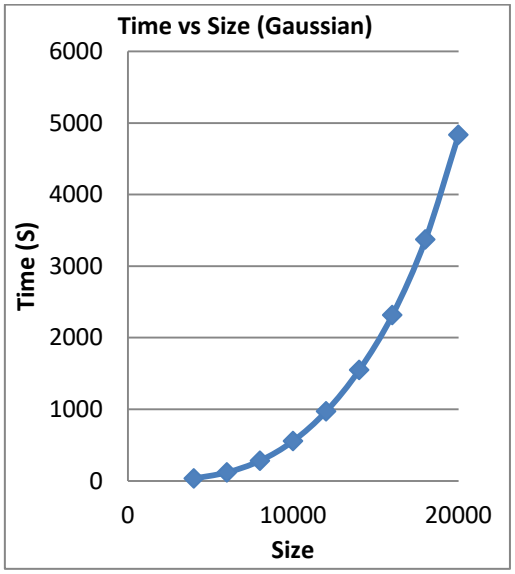
Model Input	VM with C2075	VM with K40c	Common inputs (Agnostic Model)
Gst_transactions	X		
Ecc_transactions	X		
L2_read_transactions	X		
L2_write_transactions	X		
L2_l1_read_transactions	X		
L2_l1_write_transactions	X		
Dram_read_transactions	X	X	X
Dram_write_transactions	X		
Gld_transactions_per_request		X	
Gld_transactions		X	
Execution Time	X	X	X
Temperature	X	X	X
Power Consumption	X	X	X
GPU Utilisation	X	X	X
Memory Utilisation	X	X	X

5.4.3 Model Design

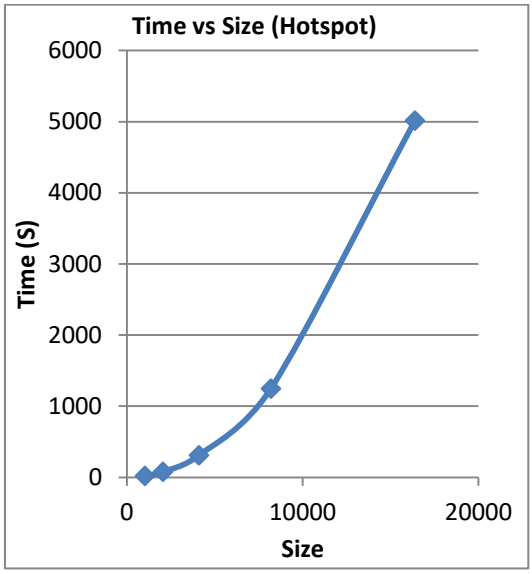
This model aims to directly and automatically calculate the CUDA applications energy consumption and eliminates human intervention when the applications are executed in a VM connected with a GPU. Two different energy models are designed for two VMs that are connected with heterogeneous GPUs (Fermi C2075 and Kepler K40c), and each GPU has different characteristics. The model will calculate the application's energy consumption by the following equation: E= P_{avg}x T, where $P_{avg}$the average of the application's power consumption and T is the application's execution time. However, as has been discussed earlier, other factors affect energy consumption, such as device temperature, GPU utilisation and memory utilisation. Therefore, these factors will be considered to predict energy consumption. For every VM connected with a GPU, there are two groups of models. The first group is to develop the energy model with individual inputs representing every GPU

architecture. The second model is to develop the model with common inputs parameters (the agnostic model) to reduce the model complexity.

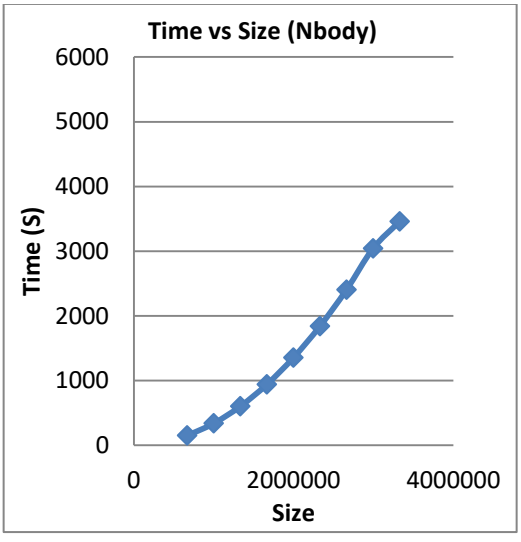
When the relationship between the size of the applications in section 5.4.1 and their execution time has been analysed, we found that most of the relationships are not linear even the size has been regularly increased. Similarly, the relationship between the size of the applications and power consumption is also not linear. This analysis is performed in both VMs that contain heterogeneous GPUs. Figure 5.3 shows the relationship between execution time and size of the selected applications when they run in the VM connected with Fermi GPU.



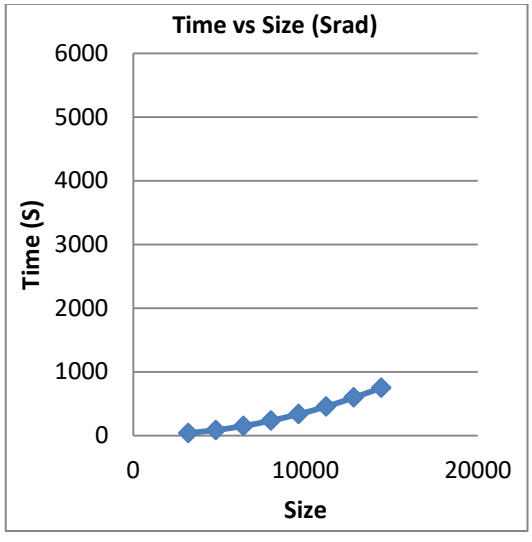
(A)



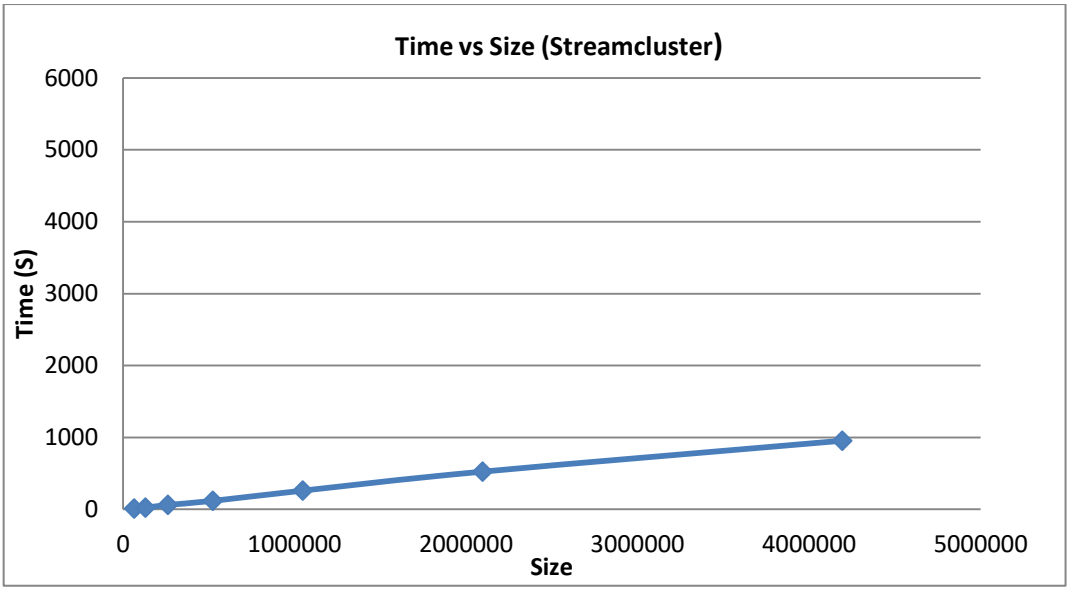
(B)



(C)



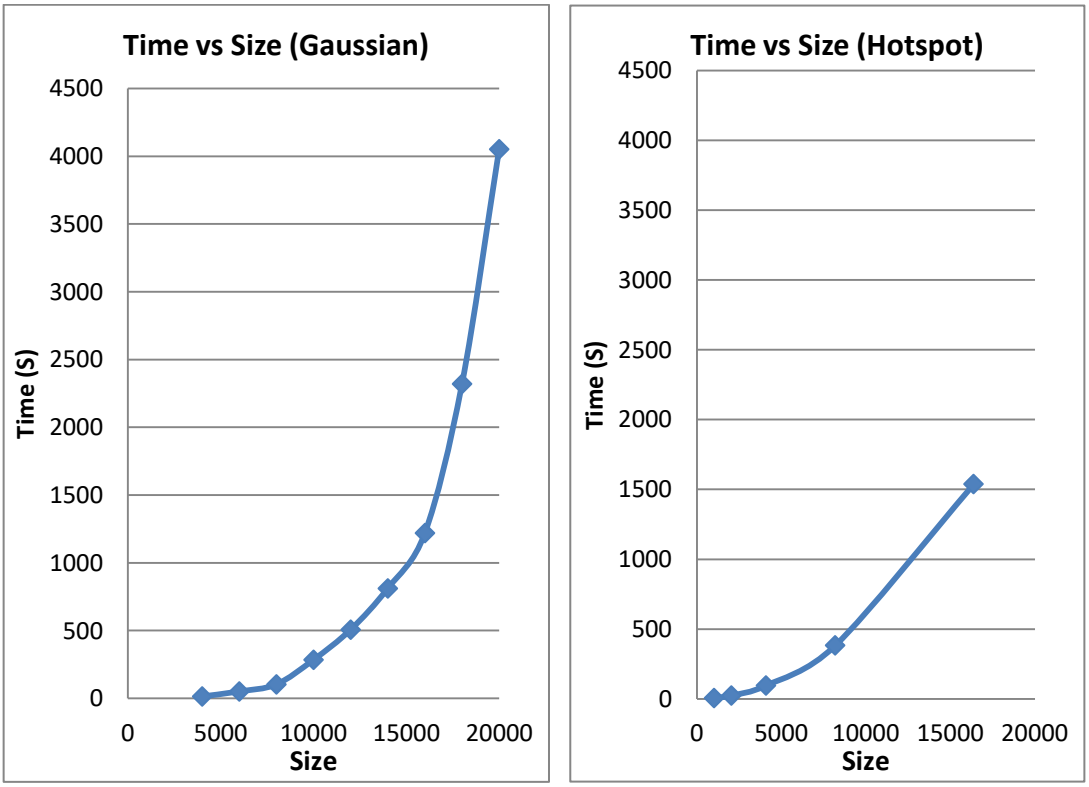
(D)



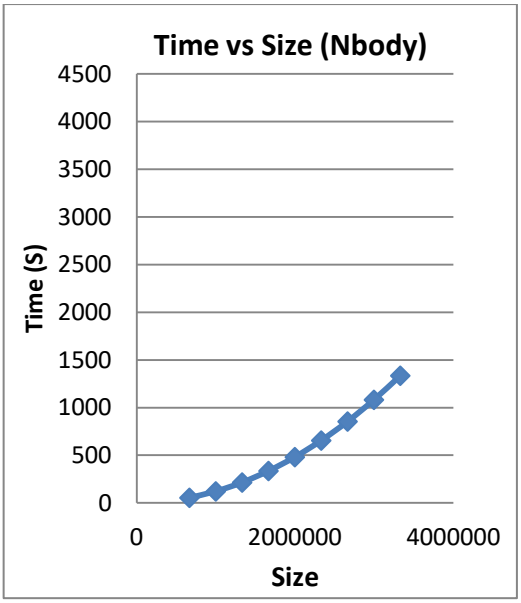
(E)

Figure 5.3: Relationship between Execution Time and Size of Applications on VM connected with Fermi GPU

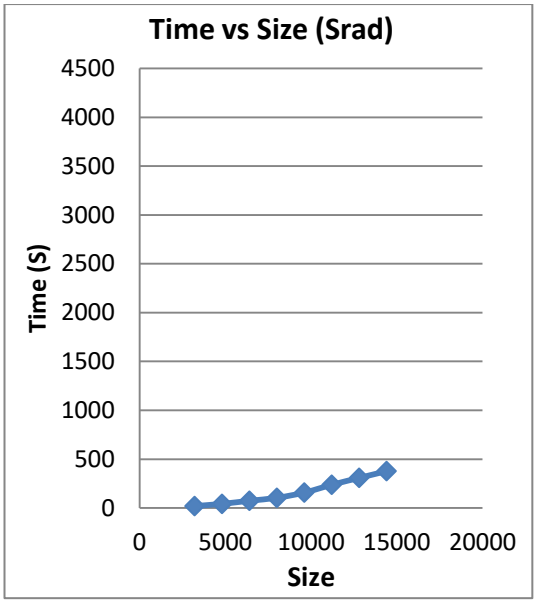
Figure 5.4 shows the relationship between execution time and size of the selected applications when they run in the VM connected with Kepler GPU.



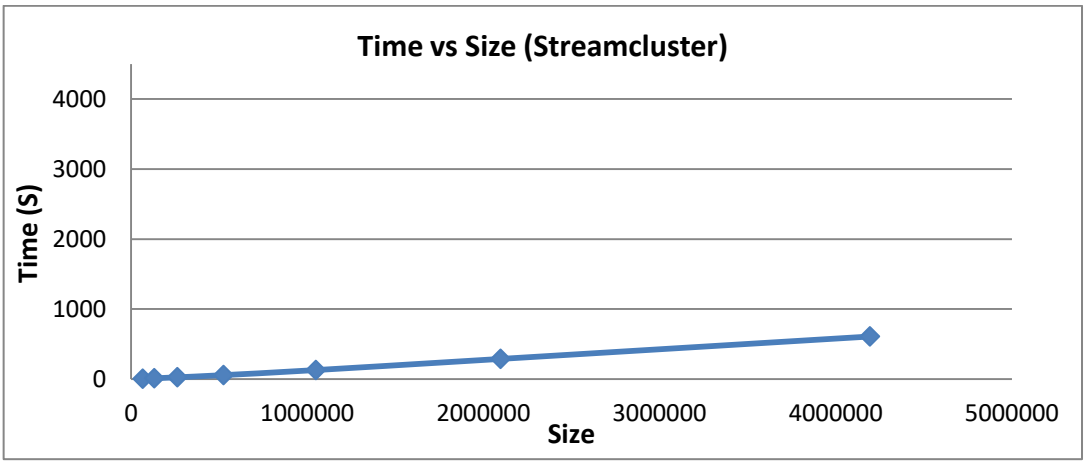
(A) **(B)**



(C)



(D)



(E)

Figure 5.4: Relationship between Execution Time and Size of Applications on VM Connected with Kepler GPU

We found that the relationships of the application conducted in different VMs connected with different GPUs are mostly similar. The calculated energy consumption when the application is increased is not linear. Thus, predicting the application's energy consumption will be complex. Therefore, DNN is an appropriate candidate to automatically calculate energy consumption.

The procedures for designing the energy consumption model are similar to the power consumption model, as illustrated in section 5.3.4. Firstly, the model's inputs in the training set have been pre-processed using z-score normalisation to unify the scale range. Then, the model has been designed using DNN to estimate energy consumption E_d by the following equation:

$$E_d = \mathrm{f}(W, X, b) \qquad (9)$$

Where *W* is the weight matrix in every layer, *X* is the input features matrix in every layer, *b* is the bias value in every layer. For the VM connected with Fermi GPU, for the DNN model that has the standard input, the input layer contains 13 neurons equivalent to the number of inputs attributes. Similar to the power model, the number of hidden layers is 4 layers, and each layer contains 8 neurons. The output layer has 1 neuron to represent the predicted energy consumption. Another energy model for the VM connected with Fermi GPU is the common inputs (the agnostic model). The features of this model are similar to the standard model except the number of neurons in the input layer is 6 neurons and the number of neuron in every hidden layer is 6 neurons.

For the VM connected with Kepler GPU, for the structure of the DNN model that has the standard inputs, the input layer contains 6 neurons equivalent to the number of inputs attributes, the number of hidden layers is 4 layers, and each layer contains 6 neurons. The output layer has 1 neuron to represent the predicted energy consumption.

Another energy model for the VM connected with Kepler GPU is with the common inputs (the agnostic model). The input layer has 6 neurons, and other layers are identical to the DNN model with standard input for VM connected with Kepler. Another aim of designing the agnostic energy model is to compare the accuracy with the standard inputs model. The number of hidden layers in all models is fixed.

Table 5.10 explains the DNN energy model structure for every VM with a different type of inputs.

Table 5.10: The DNN Features of all Energy Models in Fermi and Kepler GPUs

VMs / Features	VM with Fermi GPU (Standard inputs)	VM with Fermi GPU(Common inputs)	VM with Kepler GPU (Standard inputs)	VM with Kepler GPU (Common inputs)
Inputs Layer Neurons	13	6	8	6
Hidden Layers	4	4	4	4
Hidden Layer Neurons	8	6	6	6

The selected activation function for all DNN layers except the output layer is the ReLU activation function. The activation function of the output layer is the linear function, as illustrated in section 5.3.4.

Similar to power models, the weights of the synapses in the input layer are randomly initialised under the uniform distribution, and the weights of the synapses in hidden and output layers are randomly initialised under normal distribution, and bias values are initialised by zero.

Other hyper-parameters for all energy models are identical, and the tuned hyper-parameters are the learning rate, epochs and batch size. Table 5.11 illustrates the tuned hyper-parameters.

Table 5.11: Hyper-Parameters of all Energy Consumption Models in Fermi and Kepler GPUs

hyper-parameters	Value
Learning Rate	0.01
Epochs	1000
Batch size	33

Then, the energy models are trained using the Adam algorithm to adjust the weights of the edges in the DNN layers, similar to power models. The difference between the actual energy and the predicted energy is evaluated by the loss function in every training step.

5.5 GPU End-to-End Energy Consumption Framework

Energy modelling using merely performance counters is resulting in a significant overhead [184], [217]. In this energy model, data collection produces a critical overhead since there are two profiling tools are used. These tools are nvidia-smi and nvprof. Nvidia-smi profiles the application's power consumption, temperature, GPU utilisation and memory utilisation, and nvprof profiles the selected performance counters. Moreover, each profiling tool profiles the aforementioned factors individually for a clear measurement. This individual profiling will increase the cost of the data collection. Energy consumption modelling will be more complex when the Cloud infrastructure has several VMs contain different heterogeneous GPU architectures since each GPU has different performance counters and features. Therefore, a GPU end-to-end energy consumption

framework is developed to estimate the required application's resources when it is executed in a VM connected with a GPU and then used to predict energy consumption. The aims of developing this framework are to eliminate the overhead of the data collection because the data will take a long time when will be collected from the system, to reduce the cost of the data collection and to enable the model to predict the energy consumption of any application's size without collecting the data. So the framework just needs the application's name and size to predict energy consumption without collecting data. Thus, the framework will automatically predict the application's energy consumption with minimum cost of data collection. Figure 5.5 shows the structure of the GPU end-to-end energy consumption framework.

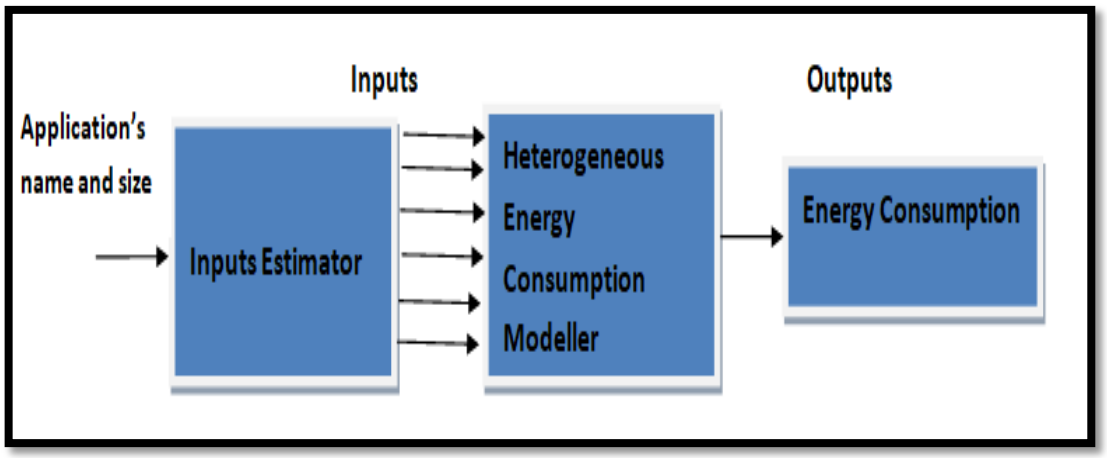


Figure 5.5: The GPU End-to-End Energy Consumption Framework

To predict energy consumption, the framework merely needs to get the application's name and its size as inputs. Then, these given data will be passed to the resource estimator components to estimate the required application's resources. These resources are the agnostic model's common inputs for heterogeneous GPU architectures complexity reduction, as discussed in section 5.4.2. The common inputs are Dram_read_transactions, Execution time, Temperature, Power Consumption, GPU Utilisation and Memory utilisation. Finally, the estimated resources will be passed to the heterogeneous energy modeller to predict the application's energy consumption. The model will be evaluated on two VMs, and each VM contains a GPU having different characteristics.

The selected applications used in these models are the following: Gaussian, Nbody, Srad and Streamcluster to predict their required resources. Hotspot application has

been eliminated because it does not contain a sufficient amount of data for validation. Table 5.12 shows the training set of the GPU end-to-end energy framework. The testing set in the previous model (as shown in Table 5.8) has been combined with the new training set to increase the training set and evaluate the model with a new data set. This training set will be used twice. First, it will be used to predict each application's resources by dividing each application's data into separate groups. Then, the total training set will be used to train the energy model. For resource estimation, each application data will be analysed and dealt with individually to estimate each resource. Table 5.13 shows the testing set of the GPU end-to-end energy framework.

Table 5.12: The Training Set of the GPU End-to-End Energy Framework

Application Name	Size
Gaussian	4000x4000
Gaussian	6000x6000
Gaussian	8000x8000
Gaussian	10000x10000
Gaussian	12000x12000
Gaussian	14000x14000
Gaussian	16000x16000
Gaussian	18000x18000
Gaussian	20000x20000
Nbody	665600
Nbody	998400
Nbody	1331200
Nbody	1664000
Nbody	1996800
Nbody	2329600
Nbody	2662400
Nbody	2995200
Nbody	3328000
Srad	3200x3200
Srad	4800x4800
Srad	6400x6400
Srad	8000x8000
Srad	9600x9600
Srad	11200x11200
Srad	12800x12800

Application Name	Size
Srad	14400x14400
Streamcluster	65536
Streamcluster	131072
Streamcluster	262144
Streamcluster	524288
Streamcluster	1048576
Streamcluster	2097152
Streamcluster	4194304

Table 5.13: The Testing Set of The GPU End-to-End Energy Framework

Application	Size	Abbreviation
Gaussian	9000x9000	G9000
Gaussian	13000x13000	G13000
Nbody	1000192	N1000192
Nbody	2500096	N2500096
Srad	5600x5600	S5600
Srad	7200x7200	S7200
Streamcluster	90000	SC90000
Streamcluster	3200000	SC3200000

5.5.1 Framework Design

In this section, each framework's component design will be discussed in a separate subsection.

5.5.1.1 Resource Estimator

To predict the application's energy consumption, the required application's resources should be estimated. Each application is analysed individually when it is run in every VM connected with a GPU (Fermi or Kepler). Appropriate single regression analysis types are used to estimate each application's resources when it is executed on a VM separately. Several single regression techniques are investigated, such as linear, power, exponential, Logarithmic and polynomial

regression types to fit the trends between the required application's resource and its sizes.

The model aims to estimate every application's input resource separately using an appropriate regression type to fit the trend in the training set for each application's group. As stated in section 2.6.1, the dependent variable in the resource estimator is a single resource for a certain application and the independent variable is the application's size.

Each application's resource has a specific and appropriate regression model to estimate its value, and each application's data group will be analysed individually. Table 5.14 explains the used regression type to estimate the required resources for each application when these applications are executed on the VM connected with Fermi GPU.

Table 5.14: Regression Types for Applications Resources Run on VM with Fermi GPU

Resource / Application	Dram_read_transactions	Temperature	GPU Utilisation	Execution Time	Memory Utilisation	Power
Gaussian	Poly. with order 2	Poly. with order 2	Constant =100	Power	Linear	Poly. with order 2
Nbody	Power	Power	Constant =100	Power	Power	Power
Srad	Power	Poly. with order 2	Constant =100	Power	Poly. with Order 2	Linear
Streamclust-er	Linear	Logarithmic	Poly. with order 2	Linear	Power	Poly. with order 3

Table 5.15 explains the used regression type to estimate the required resources for each application when these applications are executed on the VM connected with Kepler GPU. The GPU utilisation factor in some applications like Gaussian, Nbody and Srad run in Fermi GPU have a static value of GPU usage (100%) in all sizes in the training set. Gaussian and Nbody also have a static value of GPU usage in Kepler GPU execution. Therefore, a constant value is set in these applications for this input factor, and these constant inputs do not need estimation.

Table 5.15: Regression Types for Applications Resources Run on VM with Kepler GPU

Resource / Application	Dram_read_transactio-ns	Temperat-ure	GPU utilisation	Execution Time	Memory Utilisation	Power
Gaussian	Poly. with order 2	Power	Constant =100	Power	Poly. with order 2	Poly. with order 2
Nbody	Power	Power	Constant =100	Power	Power	Power
Srad	Power	Poly. with order 2	Linear	Poly. with order 2	Poly. with order 2	Linear
Streamclu-ster	Linear	Logarithmic	Poly. with order 2	Power	Power	Poly. with order 3

Table 5.16 illustrates regression equations, intercepts, slopes values and R^2 for estimating resources for Gaussian, Nbody, Srad and Streamcluster applications when they are executed on the VM connected with Fermi GPU. Moreover, Table 5.17 illustrates the previous factors when the applications are executed on the VM connected with Kepler GPU.

Table 5.16: Regressions and R^2 for Estimating Applications Resources Run on VM with Fermi GPU

VM with Fermi C2075		
Gaussian		
	Regression Equation	R^2
Dram_Read_Transactions	$0.2833x^2 - 1094x + 1E+06$	1
Temperature	$-2E-07x^2 + 0.0052x + 38.53$	0.9858
Memory Utilisation	$-5E-05x + 24.822$	0.0045
Execution Time	$3E-10x^{3.0675}$	0.9999
Power Consumption	$-9E-08x^2 + 0.0029x + 112.34$	0.995
Nbody		
Dram_Read_Transactions	$3E-09x^{2.8291}$	0.9898
Temperature	$43.225x^{0.0483}$	0.6439
Memory Utilisation	$0.0007x^{0.5725}$	0.6733
Execution Time	$5E-10x^{1.9757}$	0.9995
Power Consumption	$149.36x^{0.0173}$	0.7581
Srad		
Dram_Read_Transactions	$0.6588x^{2.0002}$	1
Temperature	$-1E-07x^2 + 0.0037x + 45.562$	0.9991
Memory Utilisation	$7E-08x^2 - 0.0011x + 23.798$	0.2776
Execution Time	$4E-06x^{2.0003}$	1
Power Consumption	$0.0002x + 108.53$	0.774
Streamcluster		
Dram_Read_Transactions	$51.868x - 82900$	1
Temperature	$4.3291\ln(x) + 6.9075$	0.9431
GPU Utilisation	$2E-14x^2 - 1E-07x + 0.4838$	0.4449
Memory Utilisation	$57.117x^{-0.122}$	0.5852
Execution Time	$0.0002x + 4.527$	0.9971
Power Consumption	$-8E-19x^3 + 5E-12x^2 - 7E-06x + 100.22$	0.566

Table 5.17: Regressions and R^2 for Estimating Applications Resources Run on VM with Kepler GPU

VM with Kepler K40c		
Gaussian		
	Regression Equation	R^2
Dram_Read_Transactions	$0.2355x^2 - 924.4x + 309886$	0.9997
Temperature	$2.781x^{0.3171}$	0.9542
Memory Utilisation	$-2E-07x^2 + 0.0036x + 6.981$	0.8576
Execution Time	$6E-12x^{3.4119}$	0.9917
Power Consumption	$-5E-08x^2 + 0.0017x + 95.403$	0.9697
Nbody		
Dram_Read_Transactions	$1E-11x^{3.2232}$	0.9966
Temperature	$3.115x^{0.2127}$	0.839
Memory Utilisation	$0.0007x^{0.5725}$	0.6733
Execution Time	$1E-10x^{1.9967}$	1
Power Consumption	$106.58x^{0.0299}$	0.8999
Srad		
Dram_Read_Transactions	$0.6051x^{2.0114}$	0.9999
Temperature	$-7E-08x^2 + 0.0027x + 29.171$	0.9971
GPU Utilisation	$0.0018x + 70.024$	0.3233
Memory Utilisation	$7E-08x^2 - 0.0011x + 23.798$	0.2776
Execution Time	$2E-06x^2 - 0.0037x + 9.958$	0.997
Power Consumption	$0.0004x + 89.972$	0.4684
Streamcluster		
Dram_Read_Transactions	$42.49x - 74786$	1
Temperature	$3.8203\ln(x) - 5.0118$	0.9368
GPU Utilisation	$2E-12x^2 - 9E-06x + 28.356$	0.2916
Memory Utilisation	$88.583x^{-0.204}$	0.7106
Execution Time	$1E-05x^{1.1643}$	0.9996
Power Consumption	$-9E-19x^3 + 6E-12x^2 - 8E-06x + 75.807$	0.3009

As shown in Table 5.17, some resource estimation models are containing low $\boldsymbol{R^2}$ values in the Kepler GPU, such as the memory utilisation model for the streamcluster application. The reason to keep using them in the framework is to preserve the aim and the consistency of predicting the dynamic required resources on both GPUs using regression analysis types. Then, the outcomes of these models will be pre-processed using the Z score normalization type and transferred to the heterogeneous GPUs energy modeller.

5.5.1.2 Heterogeneous GPUs Energy Modeller

After resources estimation models for each application in both VMs have been designed, the next step in the framework is to design DNN models for energy prediction. First, the training set is normalised to unify the data range. The agnostic model's numbers of inputs layer, hidden layer and the output layer of DNN models are similar for both energy models in both VMs with Fermi and Kepler GPUs in the framework, as stated in section 5.4.3. The number of neurons in the input layer is 6 neurons that are identical to the number of inputs attributes. The number of hidden layers is 4 layers. Each hidden layer consists of 6 neurons. The output layer contains 1 neuron to show the predicted energy consumption. The selected activation function is ReLU for each neuron in the input and hidden layers, and the selected activation function for the output layer is the linear function. Then, the equation that was used in section 5.4.3 is used to predict energy consumption. Similar to previous DNN models, the weights of the synapses in the input layer are randomly initialised under the uniform distribution. Additionally, the weights of the synapses in the hidden and output layer are randomly initialised under normal distribution, and bias values are initialised by zero. The learning rate is adjusted with 0.001, the batch size is adjusted with 33 and the epoch value is set with 1000 as another hyper-parameter. Table 5.18 shows the hyper-parameters of energy models with resource estimation for both VMs.

Table 5.18: Hyper-Parameters of Heterogeneous Energy Modeller in the GPU End-to-End Energy Framework

Hyper-parameters	Value
Neurons Number in the Inputs Layer	6
Hidden Layer	4
Neurons Number in each Hidden Layer	6
Learning Rate	0.001
Epochs	1000
Batch Size	33

The DNN model for each VM is trained individually with features of the applications using standard back-propagation technique alongside with ReLU activation function to adjust the edge weights. The selected algorithm for DNN training is the Adam algorithm, similar to previous DNN models, and each training step is evaluated by the loss function.

5.6 Implementation

Several models have been introduced in this research to predict a specific goal. First, a DNN model has been developed to estimate GPU power consumption connected with a VM. Another DNN model has been introduced to automatically calculate GPU energy consumption connected with a VM. Finally, a GPU energy consumption framework has been designed to lightly estimate energy consumption by firstly estimating the application's required resources and then pass them to the heterogeneous energy modeller. To evaluate the aforementioned models and the framework, several experiments have been performed in Leeds University Cloud testbed to generate historical data for models training and testing. Several applications are selected for well-known benchmarks and CUDA SDKs containing different features and characteristics to evaluate the models. DNNs models for power and energy predictions are implemented using the Keras platform [218] that is based on the Python programming language.

5.7 Results

5.7.1 Experimental Set-Up

The experiments are performed in the School of Computing Cloud testbed at the University of Leeds. The experiments are performed on two different Virtual Machines (VMs) supported by two heterogeneous GPUs. These heterogeneous GPUs are NVIDIA Fermi C2075 and NVIDIA Kepler K40c. OpenNebula [42] is used as a Virtual Infrastructure Manager (VIM), and the KVM is the used hypervisor. Additionally, the Operating System (OS) used is Linux CentOS in the VM. Moreover, the used CUDA compiler version is 7.5. The details of the resources of each VM and the details of all used GPUs have been previously shown in Tables 4.1 and 4.2 in section 4.5.1 within Chapter 4.

As shown in Tables 4.1 and 4.2 in Chapter 4, GPUs have different features and resources, and also the VMs connected with GPUs were set with different RAM sizes. For both VMs, The physical RAM was allocated to the virtual RAM. In all models, we selected applications sizes to be compatible with both GPUs resources since Kepler GPU and the connected VM have better resources and performance and to enable the fairness between GPUs as well. Additionally, the nvprof profiling tool measured the time of data transfer between the RAM and the GPU which is included in the execution time.

To measure the model accuracy with the actual value for each application in the testing set, the absolute error percentage is utilised as used in [26], and it is calculated by the following formula:

$$Error\ (\%) = \frac{Predicted - Actual}{Actual} \text{ X } 100 \qquad (11)$$

The best accuracy value is zero or close to zero.

5.7.2 Power Models

5.7.2.1 Performance Counters Inputs Models

Figure 5.6 summarises the difference between the actual and estimated applications power consumption of shallow and deep neural networks when they run on a VM connected with Fermi C2075 GPU.

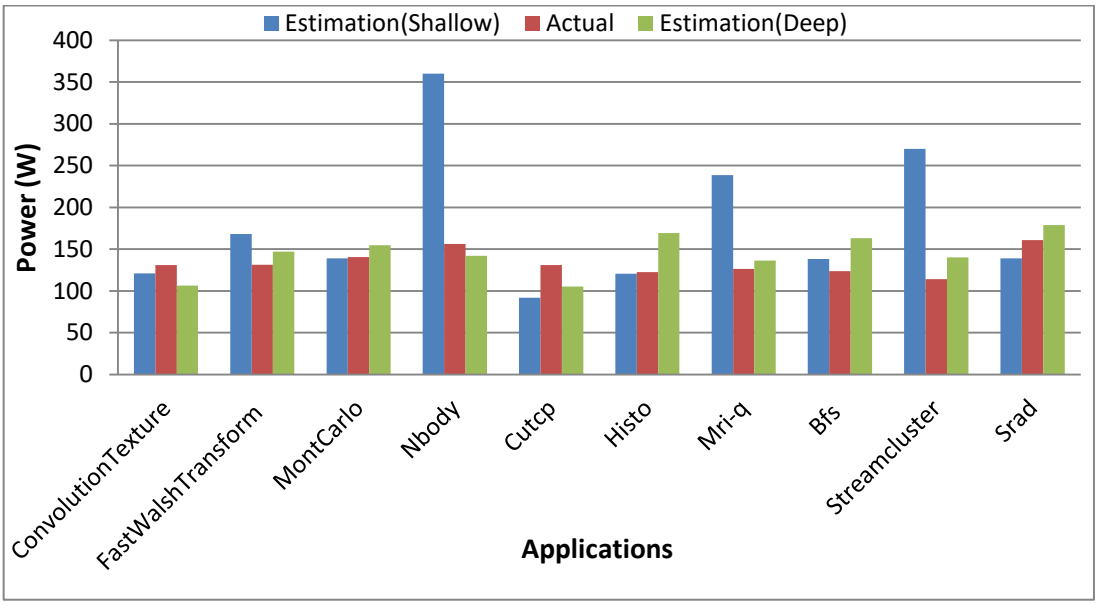


Figure 5.6: The Difference between Actual and Estimated Power Consumption in VM Connected with Fermi C2075 GPU

Figure 5.7 shows the absolute percentage error difference between shallow and deep neural networks models for predicting applications' power consumption when they executed on a VM connected with Fermi C2075 GPU.

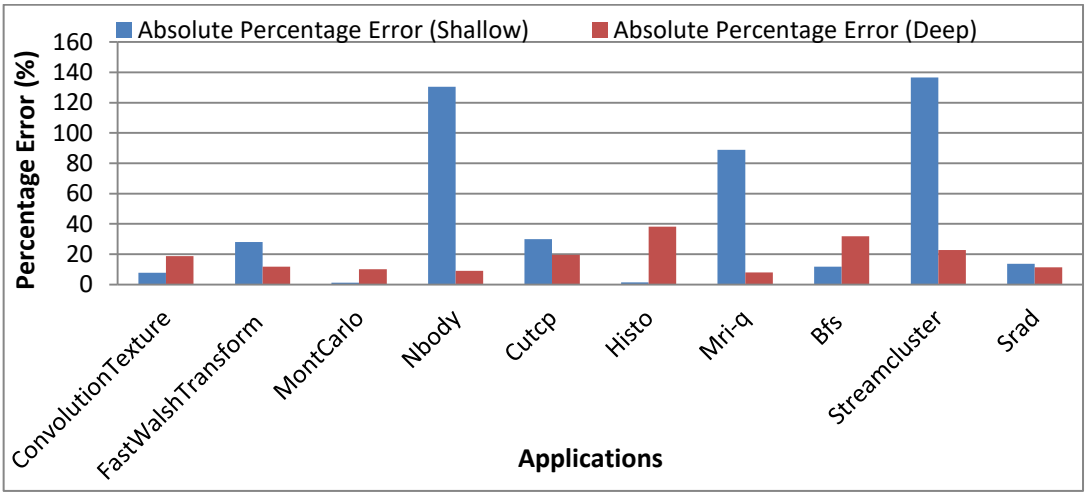


Figure 5.7: The Absolute Percentage Error between Shallow and Deep Neural Network Power Models in VM Connected with Fermi C2075 GPU

Figure 5.8 summarises the difference between the actual and estimated applications' power consumption of shallow and deep neural networks when they run on a VM connected with Kepler GPU.

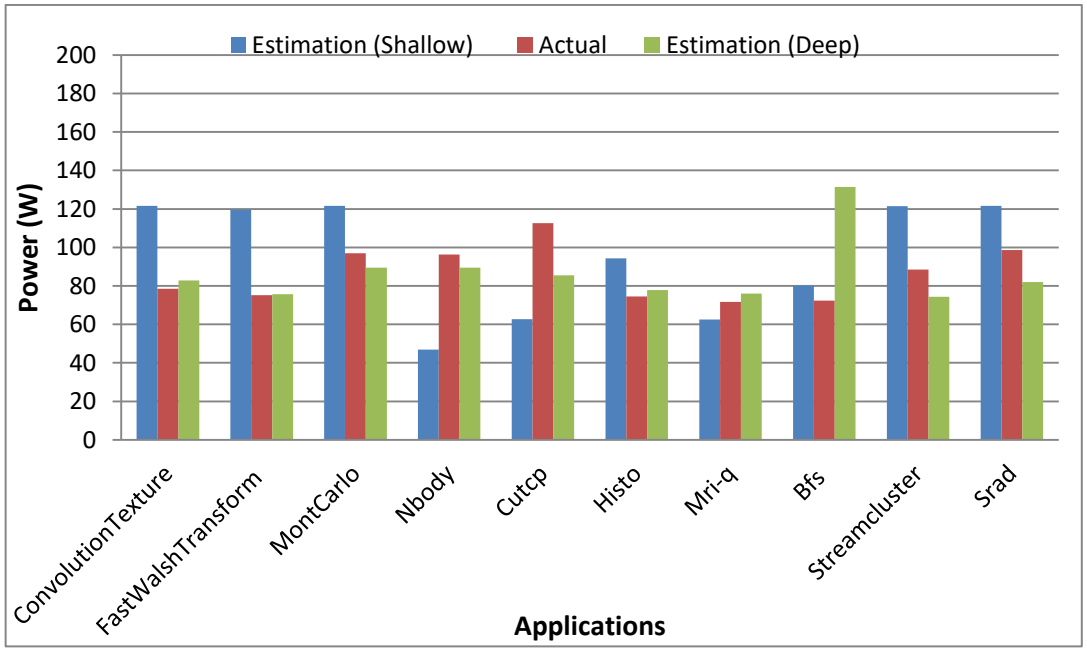


Figure 5.8: The Difference between Actual and Estimated Power Consumption in VM Connected with Kepler K40c GPU

Figure 5.9 shows the absolute percentage error difference between shallow and deep neural networks models for predicting applications' power consumption when they executed on a VM connected with Kepler K40c GPU.

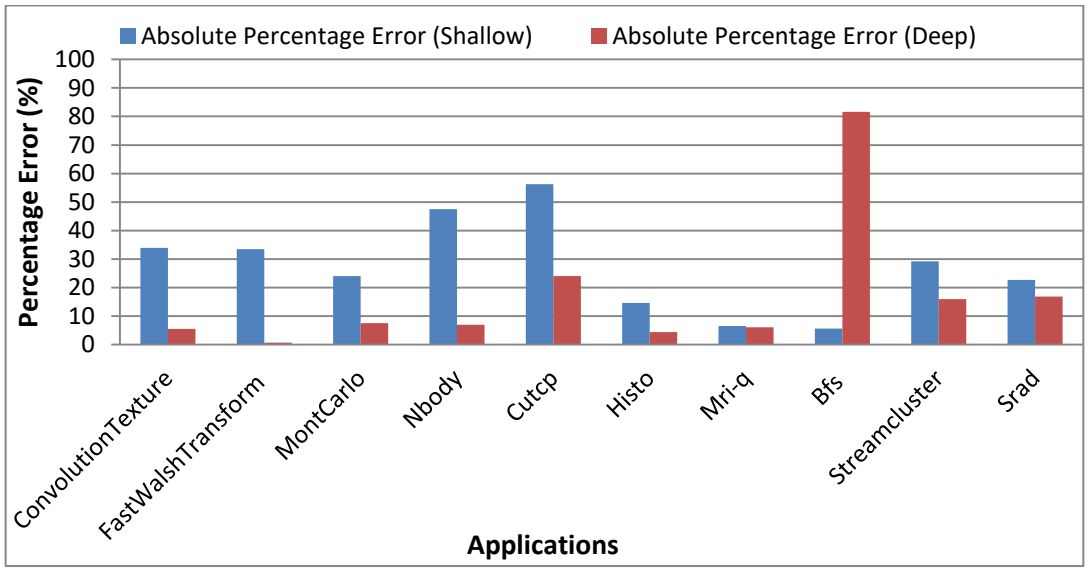


Figure 5.9: The Absolute Percentage Error between Shallow and Deep Neural Networks Power Models in VM Connected with Kepler K40c GPU

5.7.2.2 Hybrid Inputs Models

Figures 5.10 and 5.11 summarise the absolute percentage error difference between shallow and deep neural networks models with hybrid inputs for predicting applications power consumption when they executed on a VM connected with Fermi C2075 GPU and a VM connected with Kepler K40c, respectively.

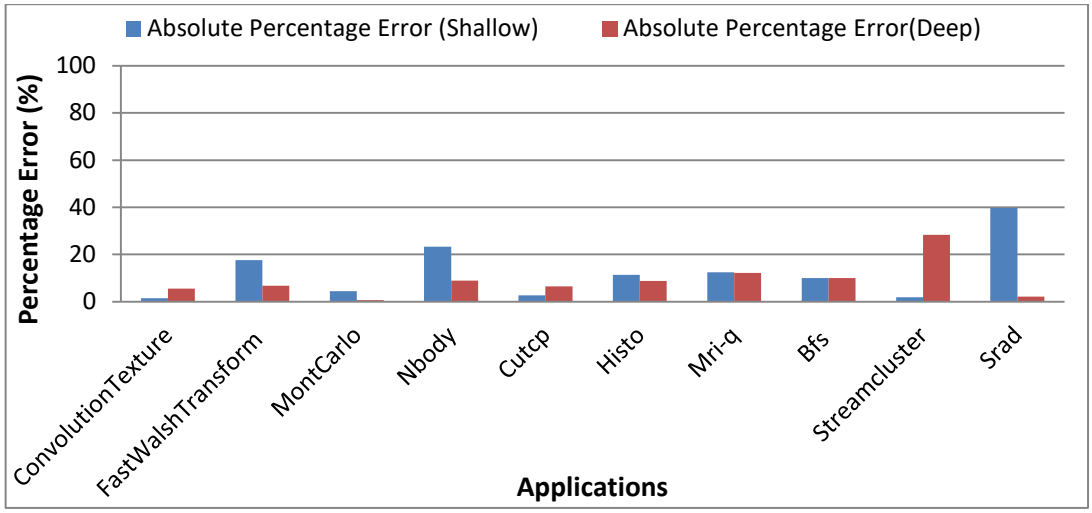


Figure 5.10: The Absolute Percentage Error between Shallow and Deep Neural Networks Power Models of hybrid inputs in VM Connected with Fermi C2075 GPU

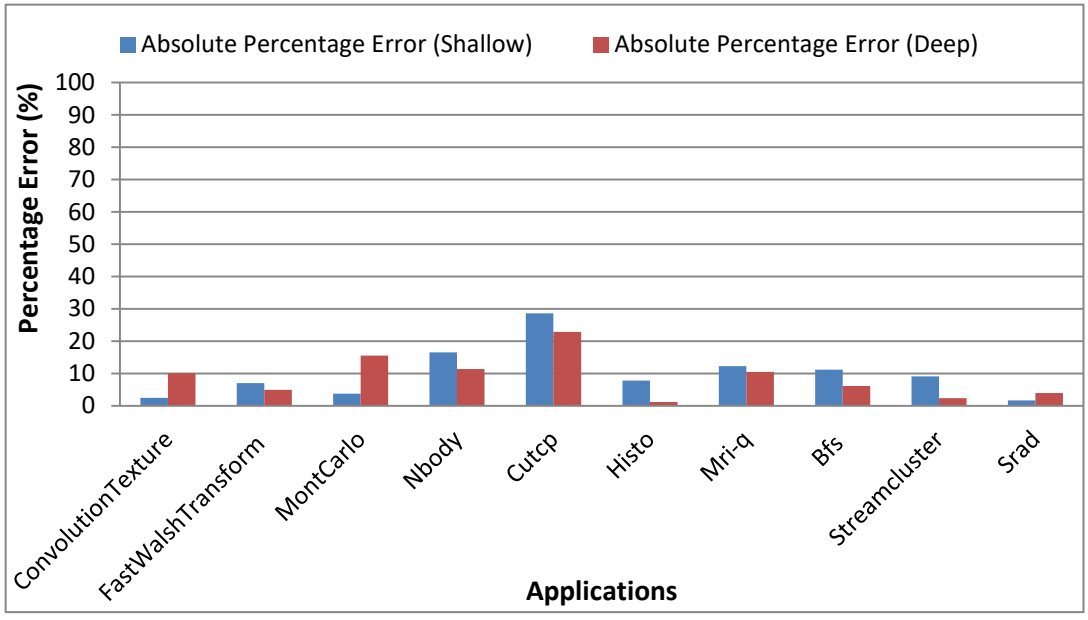


Figure 5.11: The Absolute Percentage Error between Shallow and Deep Neural Networks Power Models of hybrid inputs in a VM Connected with Kepler K40c GPU

5.7.3 Energy Models

Figure 5.12 depicts the energy consumption differences among the estimated values by the model with standard inputs, actual values and the estimated values by the model with common inputs (the agnostic model) of the applications in the testing set when these applications are executed on a VM connected with Fermi C2075 GPU.

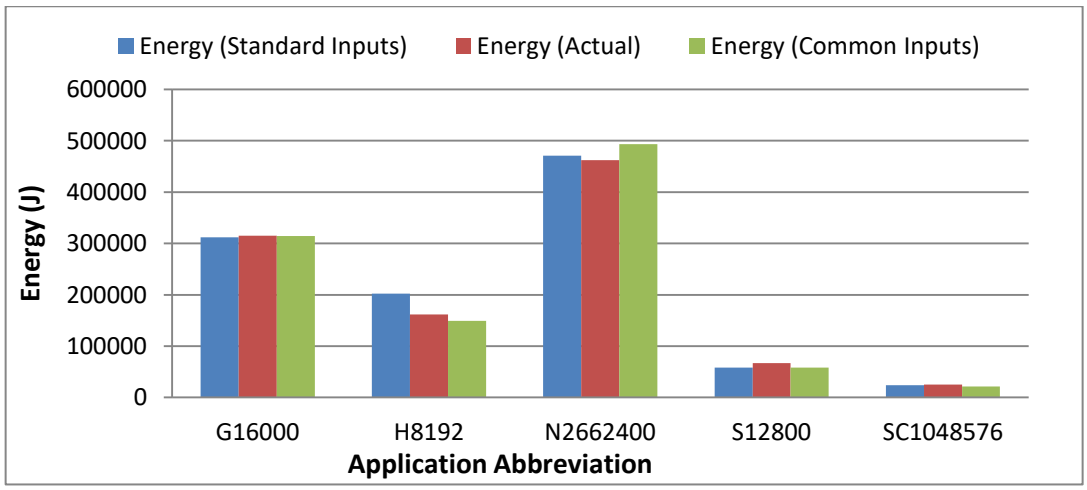


Figure 5.12: The Energy difference between Actual and Estimated Values of Standard and Common Inputs Models on a VM Connected with Fermi C2075 GPU

Figure 5.13 shows the difference of absolute percentage error values of the energy model with standard inputs and the agnostic energy model with the common inputs on the applications in the testing set when these applications are executed on a VM connected with Fermi C2075 GPU.

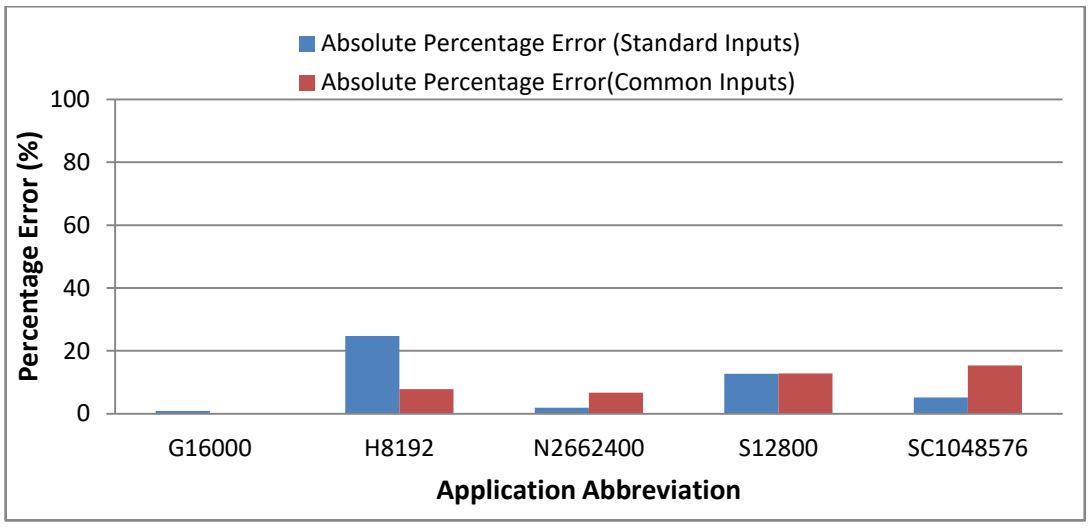


Figure 5.13: Absolute Percentage Error of Energy Models (Standard and Common Inputs) on a VM Connected with Fermi C2075 GPU

Figure 5.14 depicts the energy consumption differences among the actual and the estimated values of the developed models (standard and agnostic with common inputs) for the applications in the testing set when these applications are executed on a VM connected with Kepler K40c GPU.

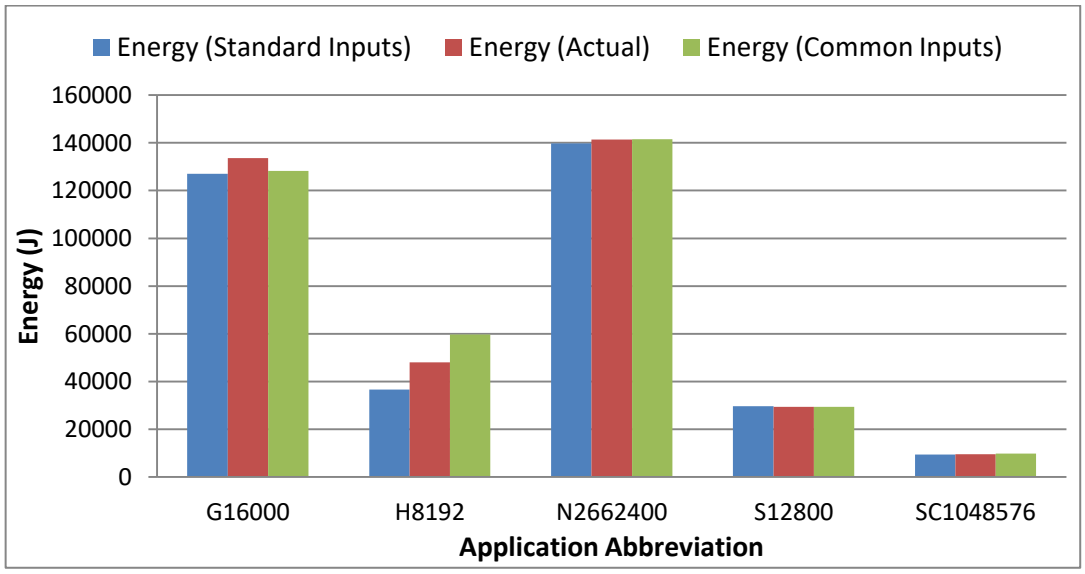


Figure 5.14: Energy difference between Actual and Estimated Values (Standard and Common Inputs) on a VM Connected with Kepler K40c GPU

Figure 5.15 shows the absolute percentage error values of the energy models (standard and agnostic with common inputs) on the applications in the testing set when these applications are executed on a VM connected with Kepler K40c GPU.

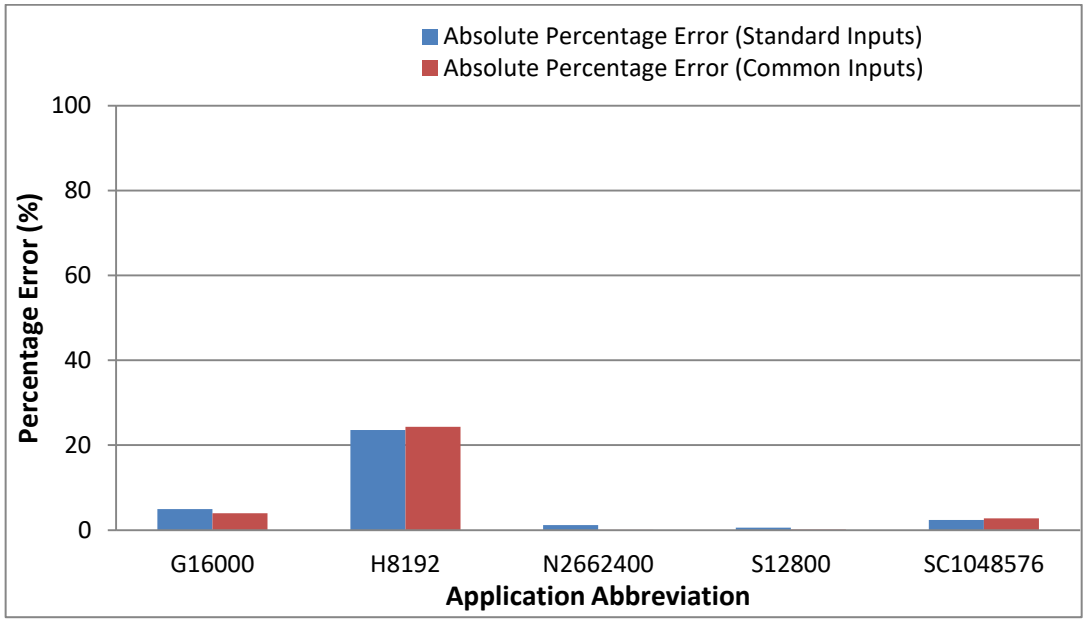
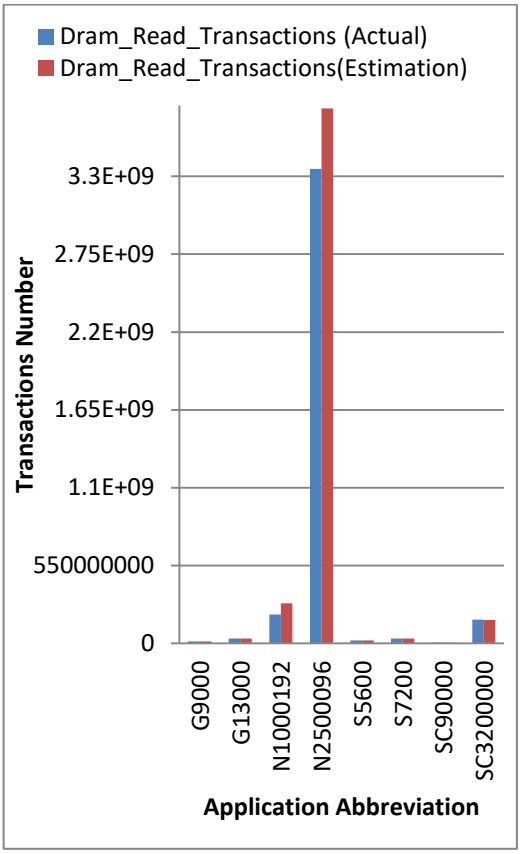


Figure 5.15: The Absolute Percentage Error of Energy Models (Standard and Common Inputs) on VM Connected with Kepler K40c GPU

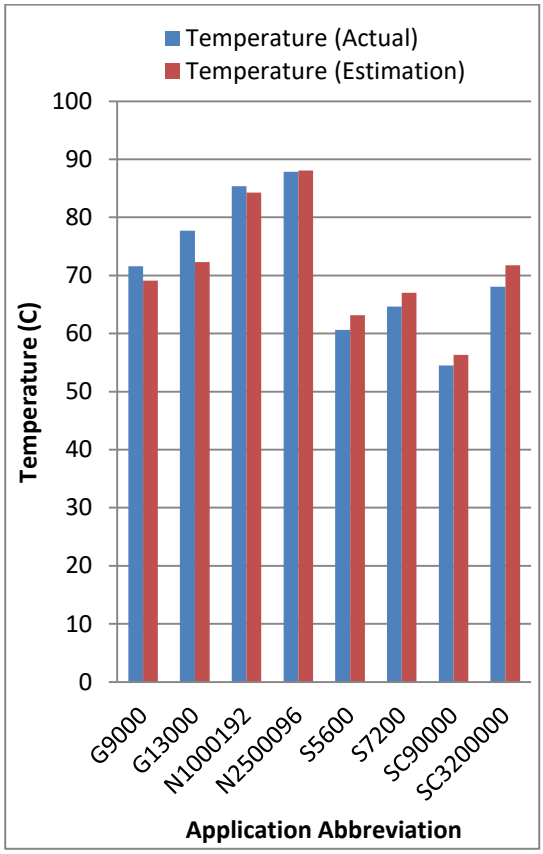
5.7.4 GPU End-to-End Energy Consumption Framework

5.7.4.1 Resource Estimator

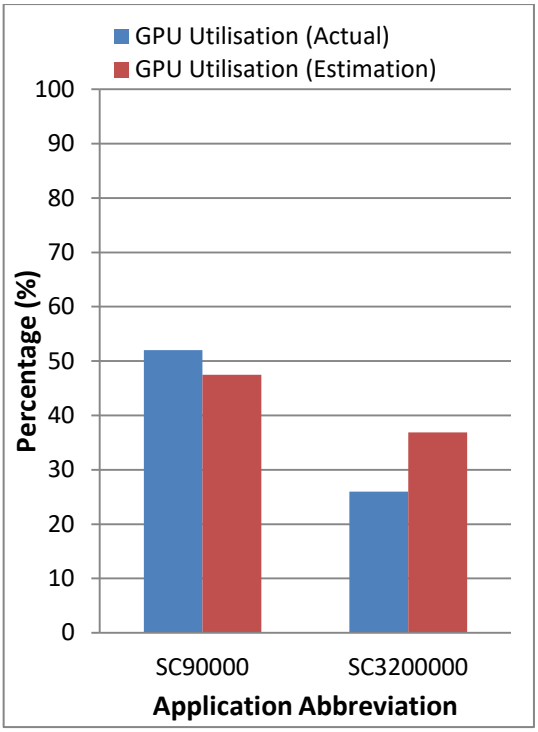
The difference between the actual and estimated resources including Dram_Read_Transactions (A), temperature (B), GPU utilisation (C), memory utilisation (D), power consumption (E) and execution time (F) for applications in the testing set when they are executed on a VM connected with Fermi C2075 GPU are shown in Figure 5.16. In terms of GPU utilisation results, the estimated values are only for Streamcluster application, with size 90000 and 3200000 because other applications in the testing set have constant values, which were assigned by a constant value. The constant value was 100.



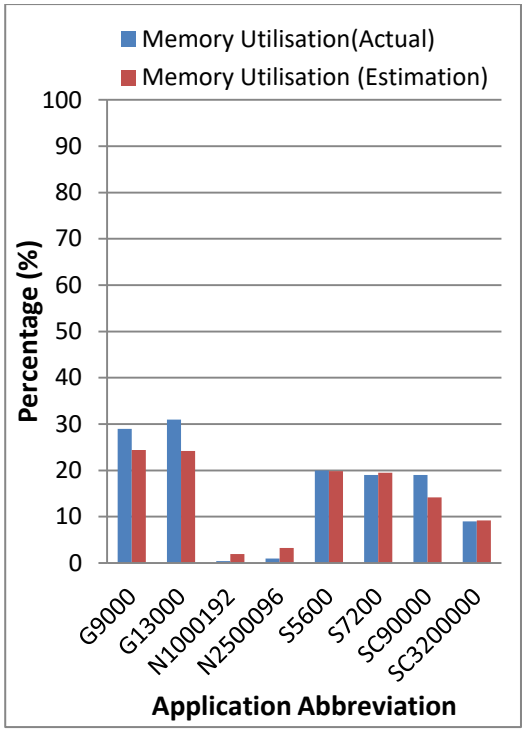
(A)



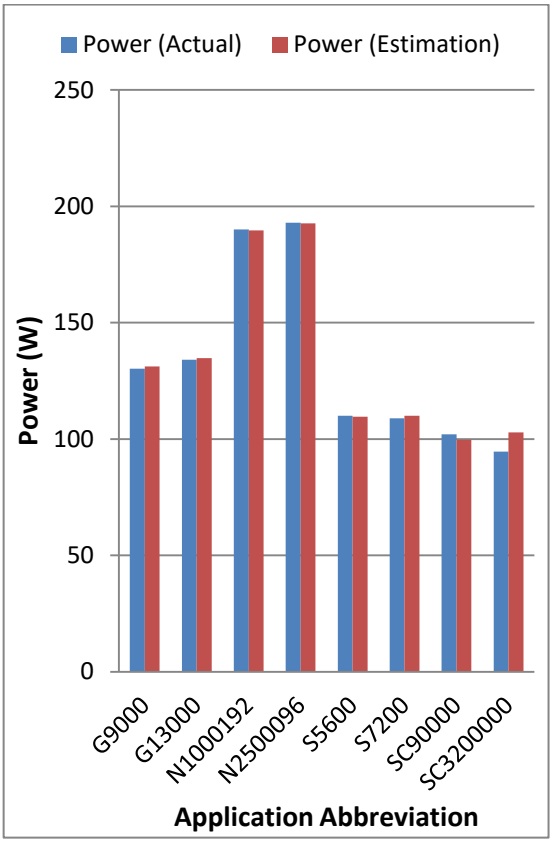
(B)



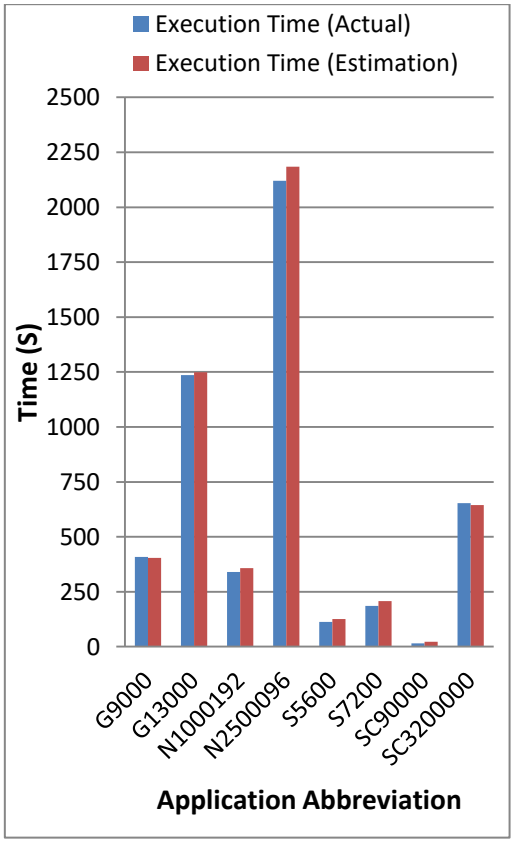
(C)



(D)



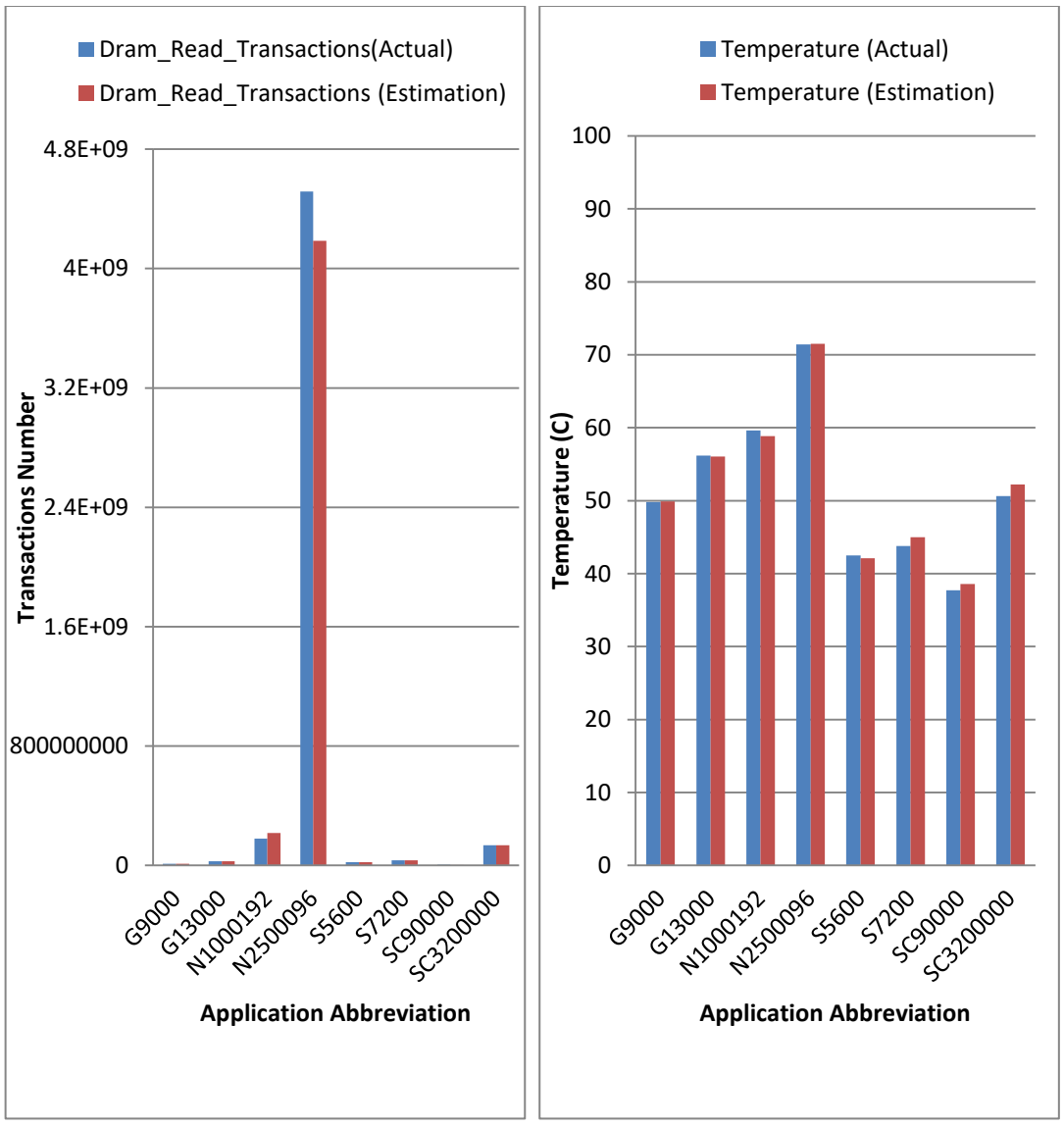
(E)



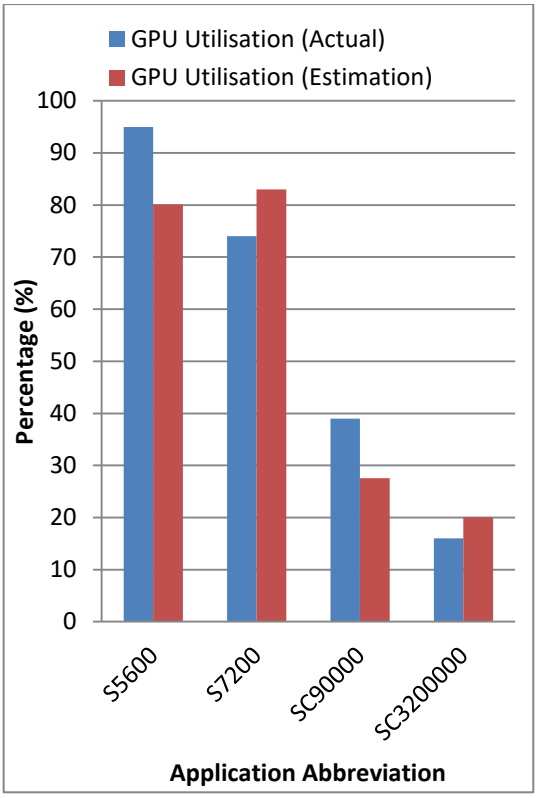
(F)

Figure 5.16: The Difference between Actual and Estimated Resources of Applications in the Testing Set in a VM with Fermi C2075 GPU

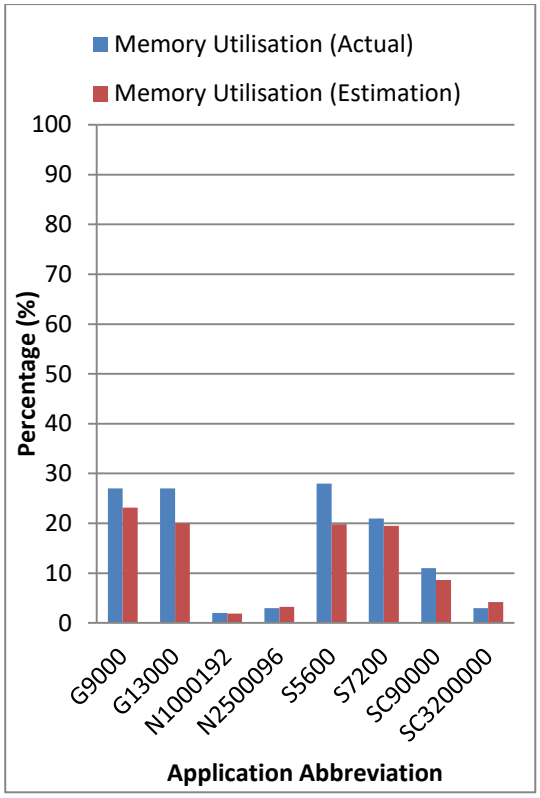
The difference between the actual and estimated Dram_Read_Transactions (A), temperature (B), GPU utilisation (C), memory utilisation (D), power consumption (E) and execution time (F) for applications in the testing set when they are executed on a VM connected with Kepler K40 GPU are illustrated in Figure 5.17. In terms of GPU utilisation results, the estimated values are just for Srad, with the size of 5600 and 7200, and Streamcluster applications, with the size of 90000 and 3200000, because the remaining applications in the testing set have constant values, which were assigned by a constant value. The constant value was 100.



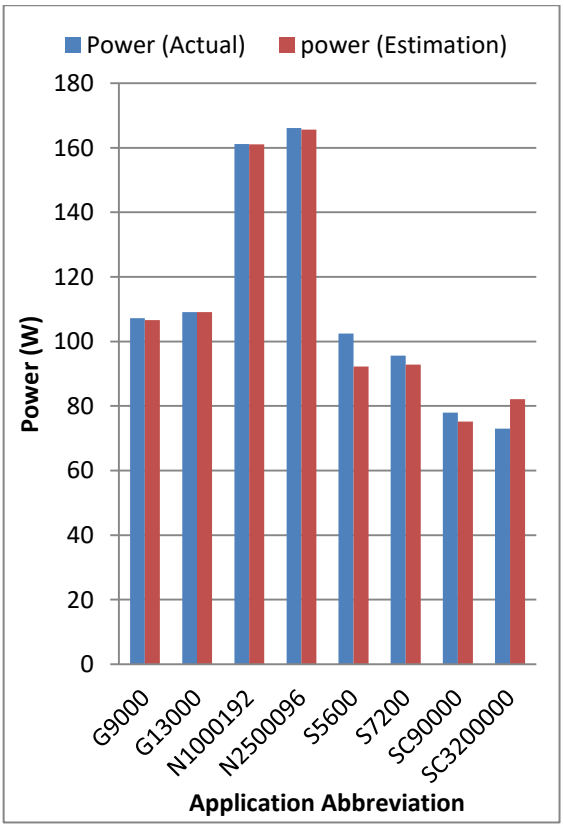
(A) **(B)**



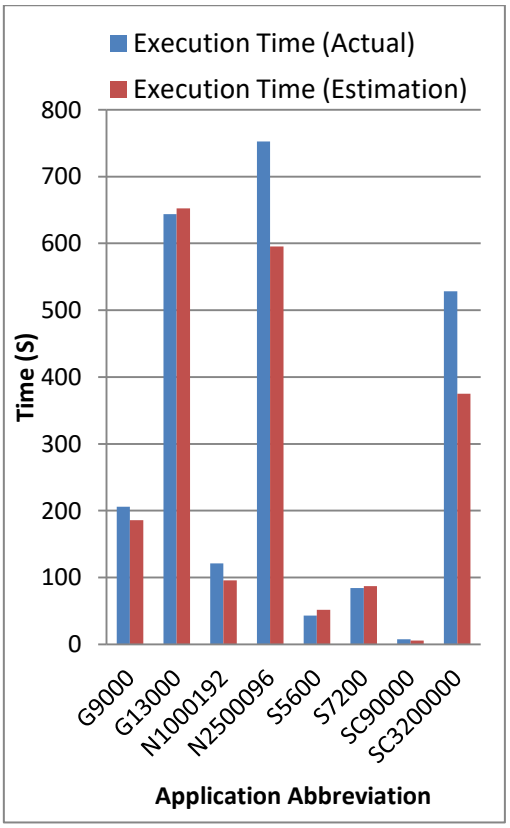
(C)



(D)



(E)



(F)

Figure 5.17: The Difference between Actual and Estimated Resources of Applications in the Testing Set in a VM with Kepler K40c GPU

Then, the outcomes of the resource estimator which are the required resources for every GPU application located in the testing set for both VMs will be normalised using the Z score and feed the heterogeneous GPU energy modeller as inputs.

5.7.4.2 Heterogeneous GPU Energy Modeller

The difference between the actual energy and the predicted energy produced by the heterogeneous GPU model and the absolute percentage error values for the applications in the testing set when they are executed in the VM connected with Fermi C2075 GPU for the model's evaluation are shown in Figures 5.18 and 5.19, respectively.

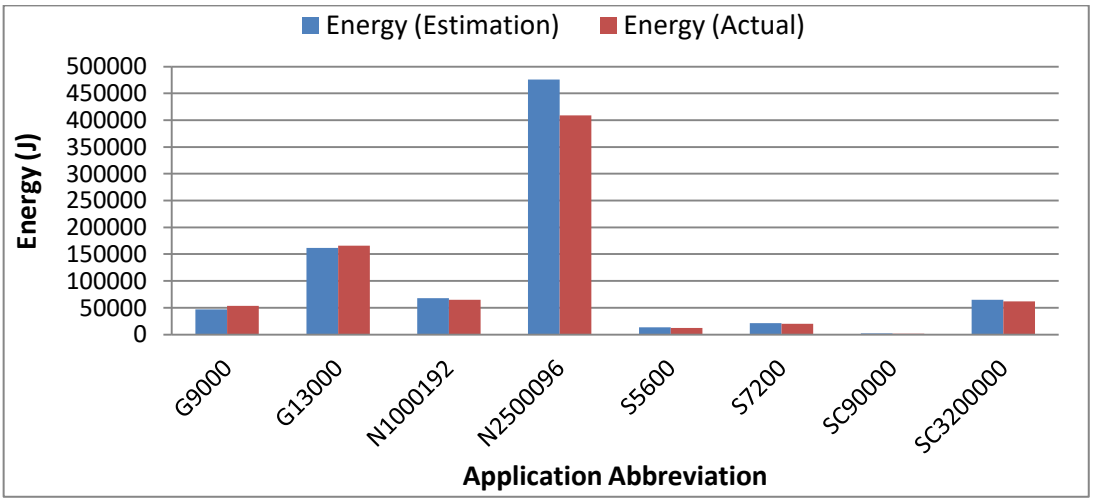


Figure 5.18: The Energy difference between Actual and Estimated Values by the model with Resource Estimation Inputs in a VM Connected with Fermi C2075

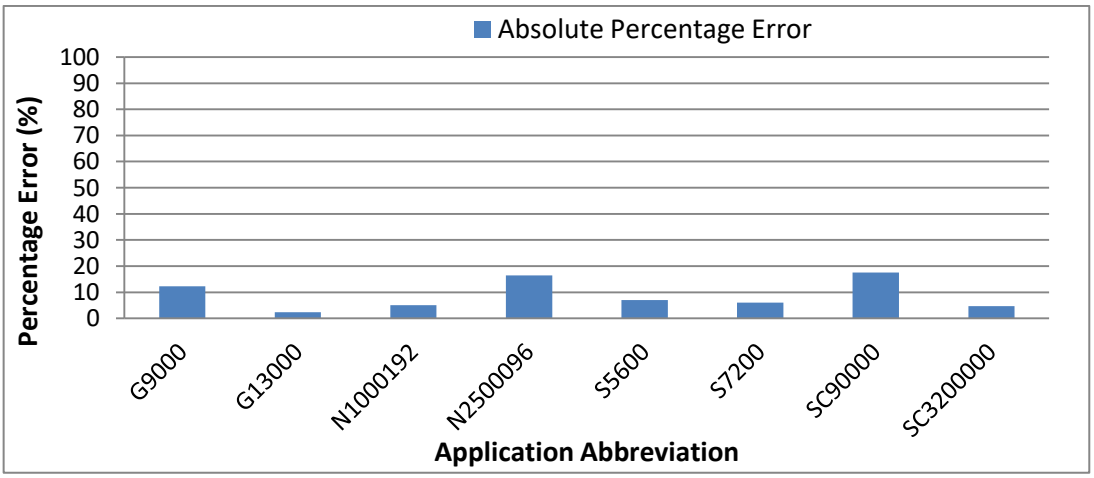


Figure 5.19: The Absolute Percentage Error of the Energy Model with resource Estimation Inputs on a VM Connected with Fermi C2075 GPU

The energy difference between the actual and the predicted energy produced by the heterogeneous energy modeller and the absolute percentage error for the applications in the testing set when they are executed in the VM connected with Kepler K40c GPU are depicted in Figures 5.20 and 5.21, respectively.

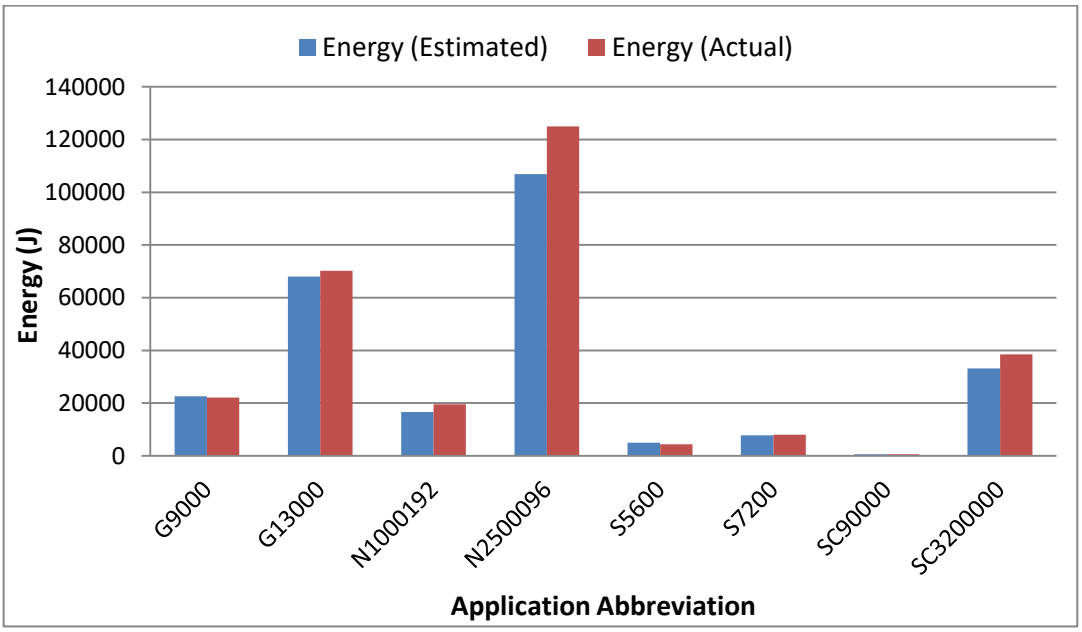


Figure 5.20: The Energy difference between Actual and Estimated Values by the model with Resource Estimation Inputs in a VM Connected with Kepler K40c

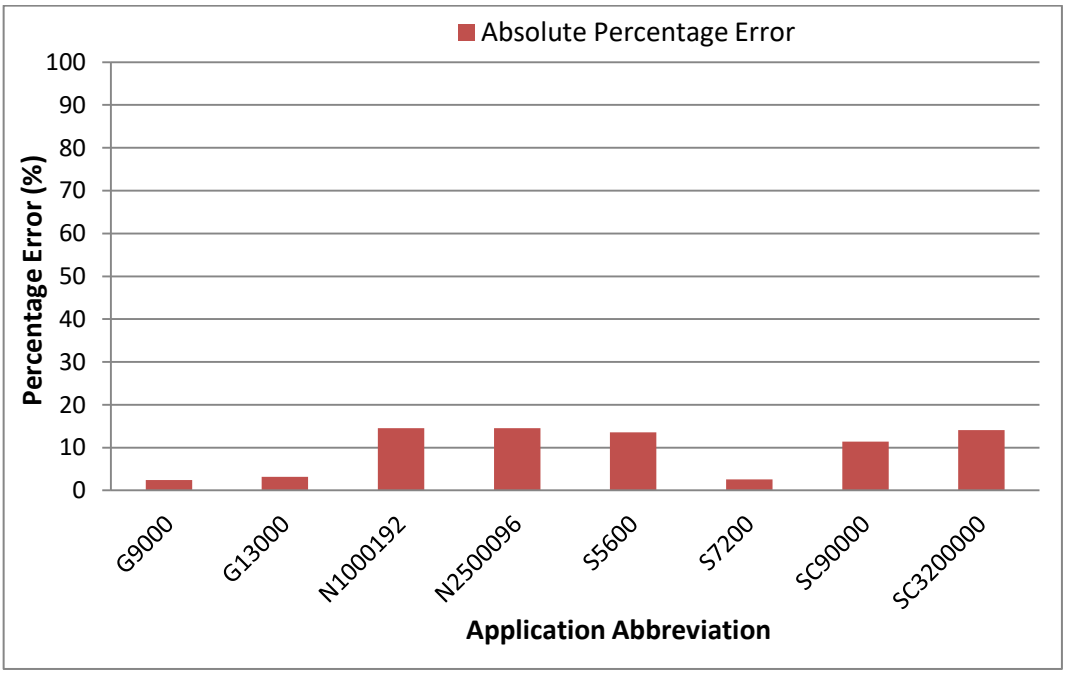


Figure 5.21: The Absolute Percentage Error of the Energy Model with resource Estimation Inputs on a VM Connected with Kepler K40c GPU

5.8 Discussion

A number of models have been developed to predict power and energy consumption on heterogeneous GPUs connected with VMs in a Cloud environment; each of which has different characterises. The absolute percentage error has been calculated to measure the overall accuracy in the testing set.

5.8.1 Power Consumption Models

In power consumption models, a comparison between shallow neural network and DNN to validate DNN models has been performed.

Acceptable accuracy differences between actual and predicted power values have been conducted using neural networks (Shallow and deep) with merely using performance counters as models' inputs. According to the power models' results on heterogeneous VMs, as shown in section 5.7.2.1, DNN models systematically produce a better accuracy compared with shallow networks in both VMs connected with heterogeneous GPUs.

In power consumption models' results on a VM with Fermi C2075 GPU, the outliers of power consumption values in Nbody, Mri-q, and Streamcluster applications produced by shallow networks have been dramatically decreased in the DNN model, as shown in Figure 5.6. Additionally, the overall accuracy in the DNN GPU power model has exhibited better accuracy than shallow networks with a mean error of 18.1% compared with 45% in shallow networks, as shown in Figure 5.7. In power consumption models results on a VM with Kepler K40c GPU, many power consumption outliers have been produced by the shallow network in ConvolutionTexture, FastWalshTransform, Nbody and Cutcp applications, while these outliers have been mitigated in the DNN power model, as shown in Figure 5.8. The overall accuracy of the DNN model is also greater than the shallow network with 17% in the comparison with 27.4% mean error in the shallow network, as shown in Figure 5.9.

A significant accuracy improvement has occurred when GPU and memory utilisations have been added with the selected performance counters, hybrid

inputs, for power consumption prediction in both heterogeneous GPUs connected with two different VMs, as illustrated in section 5.7.2.2.

In the power consumption model's results on a VM with Fermi C2075 GPU, the accuracy of shallow networks has been improved with a mean error of 12.5% compared with 45% in the performance counters only model. Similarly, the DNN power model's accuracy has been improved from 18.1 % in the performance counters only model to 9% in the hybrid inputs model, as shown in Figure 5.10. In terms of individual applications accuracy, the greatest absolute percentage error has been dropped from 38.1% (in Histo application in Figure 5.7) in performance counters only inputs to 28.3 (Streamcluster application in Figure 5.10) in hybrid inputs for the DNN power model. While in the shallow networks model, the greatest absolute percentage error has been reduced from 137% (in Streamclucter application in Figure 5.7) to 40% (Srad application in Figure 5.10) on the VM connected with Fermi C2075. However, the accuracy of DNN has performed better than shallow networks with a 9% mean error and 12.5% for shallow networks.

In power consumption results on a VM with Kepler K40c GPU, an improvement of the overall accuracy has been occurred in shallow networks from 27.4% mean error in only performance counters inputs to 10% mean error in hybrid inputs model. Moreover, the overall accuracy in DNN has been enhanced from 17% mean error in only performance counters inputs to 9% mean error in hybrid inputs, as shown in Figure 5.11. Although the accuracy of the shallow networks has been improved in hybrid inputs, the DNN model has performed better than the shallow networks model with 9% compared with 10% mean error in the shallow network model. For individual evaluation, the greatest absolute percentage error has been decreased from 82% (Bfs application in Figure 5.9) to 23% (Cutcp application in Figure 5.11) in the DNN model with hybrid inputs. Similarly, the greatest absolute percentage error value in shallow networks has been dropped from 56.3% (Cutcp application in Figure 5.9) to 29% (Cutcp application in Figure 5.11) in the hybrid inputs model. Table 5.19 summarises the mean errors and the greatest errors percentages of GPUs power consumption models on both VMs.

Table 5.19: The Mean errors and Greatest Errors Values Summary of Power Consumption Models on Both VMs

VMs / Power Models	VM with Fermi C2075 GPU		VM with Kepler K40c	
	Mean Error	**Greatest Error**	**Mean Error**	**Greatest Error**
Performance Counters Inputs (Shallow)	45	137	27.4	56.3
Performance Counters Inputs (Deep)	18.1	38.1	17	82
Hybrid Inputs (Shallow)	12.5	40	10	29
Hybrid Inputs (Deep)	**9**	**28.3**	**9**	**23**

According to Table 5.19, the minimum mean error percentage of all power models on both VMs were DNN models with hybrid inputs in both VMs with 9%, and the minimum greatest error percentage of all power models was the DNN model with hybrid inputs in the VM connected with Kepler K40c with 23%.

5.8.2 Energy Consumption Models

The outliers in energy DNN models results have not been existed comparing with actual energy values in both VMs, as shown in section 5.7.3.

In comparison energy models having standard inputs and the agnostic model that has the common inputs in the VM connected with Fermi GPU, the overall accuracy in the agnostic energy model with common inputs has performed slightly better than the energy model with standard inputs with a mean error of 8.6% compared with 9.1% mean error in the standard energy model. In terms of individual accuracy, the maximum percentage error in the DNN energy model with standard inputs was 25% (in H8192), while the greatest error in the agnostic DNN energy model with the common inputs was 15.4% (in SC 1048576), as shown in Figure 5.13.

In a VM connected with Kepler GPU, the agnostic energy model with common inputs has performed slightly better than the energy model with standard inputs with a mean error of 6.3% compared with 6.5% in the energy model with standard inputs. Moreover, the maximum individual percentage error in the DNN energy model with standard inputs was 23.6% (in H8192), while the highest individual percentage error in the DNN energy model with common inputs was 24.3% (in

H8192). Therefore, the agnostic DNN energy model with common inputs has exhibited a better performance in terms of the mean error and the maximum individual error percentage, as shown in Figure 5.15. Table 5.20 summarises the mean errors and the greatest errors percentages of energy models in both VMs.

Table 5.20: The Mean errors and Greatest Errors Values Summary of Energy Models for both VMs

VMs / Energy Models	VM with Fermi C2075 GPU		VM with Kepler K40c	
	Mean Error	**Greatest Error**	**Mean Error**	**Greatest Error**
DNN with Standard Inputs	9.1	25	6.5	24.3
DNN with Common Inputs (agnostic model)	8.6	**15.4**	**6.3**	23.6

According to Table 5.20, for both VMs comparison, the best mean percentage error for the energy prediction model was the VM connected with Kepler K40c GPU (6.3%), while the VM that connected with Fermi C2075 GPU had the minimum greatest individual error (15.4%).

Thus, the energy model with common inputs has performed a better performance and provided to reduce the inputs parameters, which contributes to reduce the cost of data collection.

Considering DNNs models to predict energy consumption is a critical approach. Training DNNs models consume a large amount of energy consumption for both GPUs. DNNs provide a reasonable energy consumption estimation, but with a high demand for energy consumption for training purposes. Thus, there is a trade-off and overhead between using Deep Neural Networks for energy consumption estimation and training these models.

5.8.3 GPU End-to-End Energy Consumption Framework

Although the required inputs resources for energy consumption prediction models have been estimated, the predicted energy consumption values produced by DNN models were extremely acceptable. The variation between actual and predicted

energy values in the testing set was not extreme for both VMs, as shown in section 5.7.4.2.

The DNN model in the VM connected with Fermi GPU has performed well with a mean error of 9%, and the range of individual percentage errors has varied between 2.4% and 17.6%, as shown in Figure 5.19.

Furthermore, the DNN model in the VM connected with Kepler GPU has distinguishably performed with a 9.5% of mean error, and the range of individual percentage errors has varied between 2.4% and 14.6%, as shown in Figure 5.21.
For overall comparison between VMs, the DNN energy model in the VM connected with Fermi C2075 GPU has had the smallest mean percentage error (9%), and the DNN energy model in the VM connected with Kepler K40c has had the smallest maximum individual percentage error (14.6%).

In conclusion, selecting the appropriate input parameters and adequate model type can improve the prediction's accuracy, reduce the model's cost and overheads as well.

5.9 Summary

This chapter has developed, evaluated and discussed power and energy consumption models in several ways. These models have been developed for two VMs, and each of them contained heterogeneous GPUs (Fermi C2075 and K40c) with different generations. Each GPU has consisted of different characteristics and features. Models have been evaluated by experiments through a Cloud testbed between actual and predicted values produced by models.

DNN power models have been validated with shallow neural networks using performance counters as inputs. Then, the results have been significantly enhanced when using hybrid inputs (performance counters, GPU and memory utilisation).

Moreover, DNN energy models have been presented for two VMs. A comparison between standard and agnostic energy models containing common inputs have been conducted in each VM. Agnostic energy models with common inputs for both VMs have shown a slight enhancement of accuracy with selected inputs reduction.

Finally, energy and resource estimation models in the GPU end-to-end energy framework using common input for two VMs have been developed and evaluated. Appropriate regression types techniques have been used to estimate the required application resources, and DNNs have used to predict energy consumption by using the estimated resources as inputs. The reason for developing these models was to eliminate the cost of data collection when using third party profilers. The outcomes of the resource estimation models have been used for energy models' inputs and then predicting energy consumption by knowing the application's name and size with significant results.

The next chapter will discuss another component of the main architecture in details. This component is the scheduler component that considers energy consumption and performance parameters. The outcomes of the predicted energy consumption and execution time in the GPU energy framework, as stated in section 5.5, will be used for scheduling purposes.

Chapter 6 Proposed Scheduling Policies

6.1 Overview

This chapter presents the design and evaluation of the proposed scheduling algorithms. An introduction to the chapter is given in section 6.2. System architecture and system model are presented in sections 6.3 and 6.4. System design is illustrated in section 6.5. Finally, the evaluation techniques for the proposed algorithms and their results are presented in sections 6.6 and 6.7.

6.2 Introduction

This chapter presents the details of the scheduler component which is a part of the propped architecture to manage heterogeneous GPU resources in Cloud computing introduced in Chapter 4, as shown in Figure 6.1 (highlighted in red).

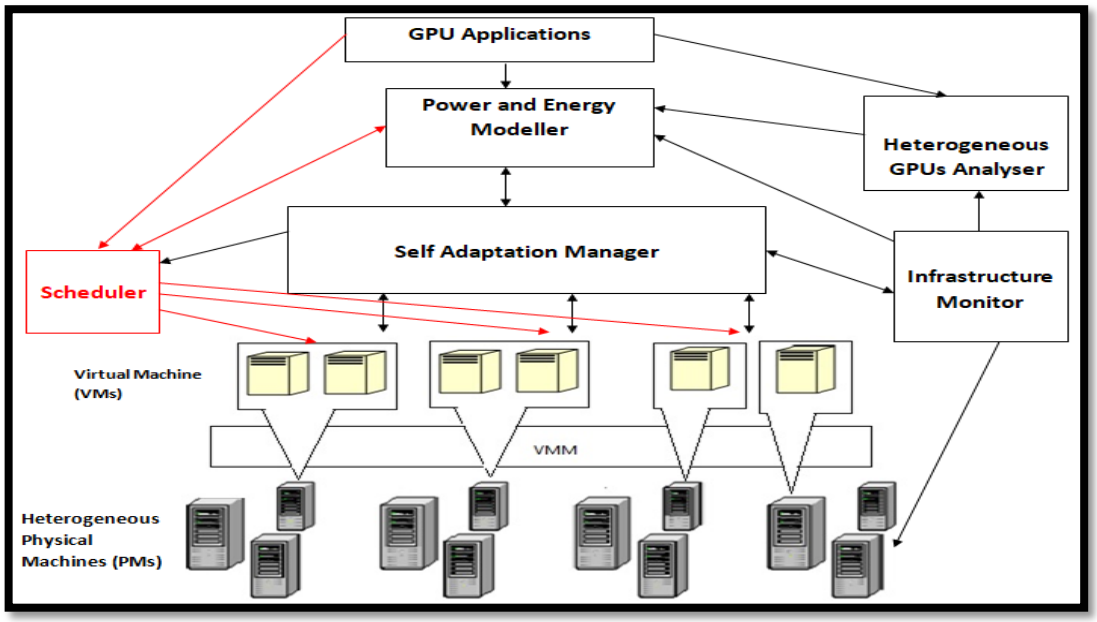


Figure 6.1: The Proposed Architecture with the Scheduler (Highlighted in Red)

There are several types of workload used in Cloud data centres. In general, they can be classified into two main categories: interactive and non-interactive workloads [219]. Interactive, also called transactional, workloads should be working continuously without interruption and need real-time processing for completing at a certain time. Examples of interactive workloads are web services and business applications. Non-interactive, also called batch, workloads can delay the tolerance and can be scheduled for execution any time before their deadline.

So, non-interactive workloads allow enhancing resource management in the Cloud computing environments by reducing the energy consumption by scheduling the application to the available energy-efficient VM under SLA constraints. Examples of non-interactive are scientific applications, image processing, simulations and financial applications

Different types of applications have been running on Cloud computing infrastructure, such as compute-intensive and data-intensive applications. These applications are managed with Quality of Service requirements established by the user, under an agreement between the user and the Cloud provider Called Service Level Agreement (SLA).

The energy efficiency of Cloud data centres is considered one of the most critical issues in the last decade. One way to solve this issue is by energy-efficient Cloud resource management for sustainable usage of Cloud infrastructure [220].

The servers' energy consumption in Cloud data centres is mainly caused by executing a specific application that runs on each server. The varieties of applications design including precise implementation and the hardware characteristics that executes the application can create different power readings [221]. Consequently, indiscriminately allocating applications on heterogeneous servers that have diverse capabilities can cause substantial differences in energy consumption. Thus, energy aware policies cooperating with effective applications energy modelling can lead to significant energy consumption saving.

The problem of minimising energy consumption under SLA constraints is considered an NP-hard problem, which is hard to be solved in polynomial time [222],[223],[224]. It becomes more challenging if the heterogeneity of the Cloud infrastructure factor is added. Therefore, researchers develop approximation or heuristic algorithms for solving this problem.

This work presents a heuristic energy aware scheduling policy that aims to reduce energy consumption. Another scheduling policy is proposed to maintain the consequences of reducing energy consumption when these consequences are taken place and establish a balanced trade-off between energy consumption and performance.

6.3 System Architecture

Figure 6.2 shows the high level of the system architecture. The Cloud end-user submits CUDA application to the Cloud provider. Each application hosted on the Cloud has a different storage capacity and processing demand. Additionally, each application's energy differs when it is executed in different VMs since the divergence of the composed resources of VMs. Therefore, the Cloud provider has to efficiently analyse the behaviour of the application when it executed in a heterogeneous Cloud infrastructure to increase profits by reducing the operational costs. Then, the Cloud receives the submitted application with SLA constraints created by the Cloud end-user. In the middleware layer, the Heterogeneous Cloud Infrastructure Scheduler (HCIS) is responsible for placing the application on a preconfigured VM created through the VMM. The VMM interacts with the Virtual Infrastructure Manager (VIM) to efficiently manage and control VMs created by heterogeneous resources, in this work the VIM is OpenNebula. Each VM has different resources from another VM. The heterogeneity in this work is by using different types of GPU architectures in the Cloud infrastructure. The first VM is supported by Nvidia Fermi C2075 GPU and the second VM is supported by Nvidia Kepler K40c GPU. HCIS aims to reduce energy consumption and maintain QoS at the deployment time.

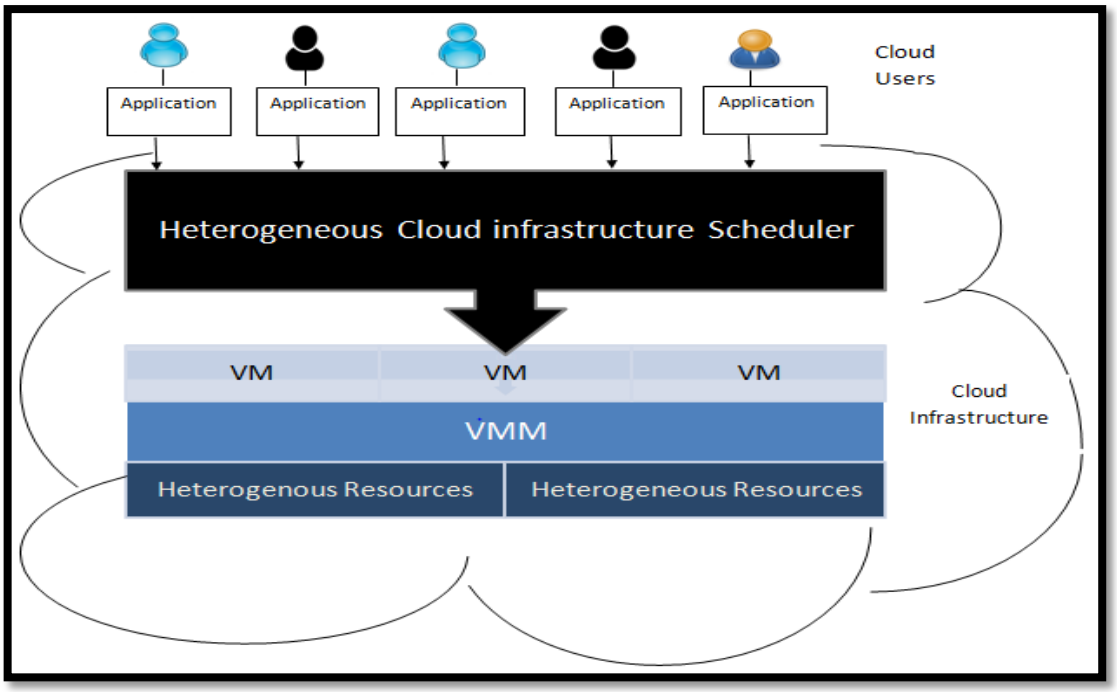


Figure 6.2: High-Level View System Architecture

More specifically, the submitted application by the Cloud end-user goes through HCIS to be scheduled to an appropriate VM. HCIS has two main stages, as depicted in Figure 6.3. Firstly, the application needs to be checked by the admission control. The application has to pass the requirements of admission control, which will be discussed in detail in Section 6.5.1. The admission control communicates with the energy model by sending the application's name and its size to obtain the estimated energy and execution time for both Fermi and Kepler executions. Moreover, the admission control contacts with the application mapping to find when the application will start executing whether in VM1 or VM2.

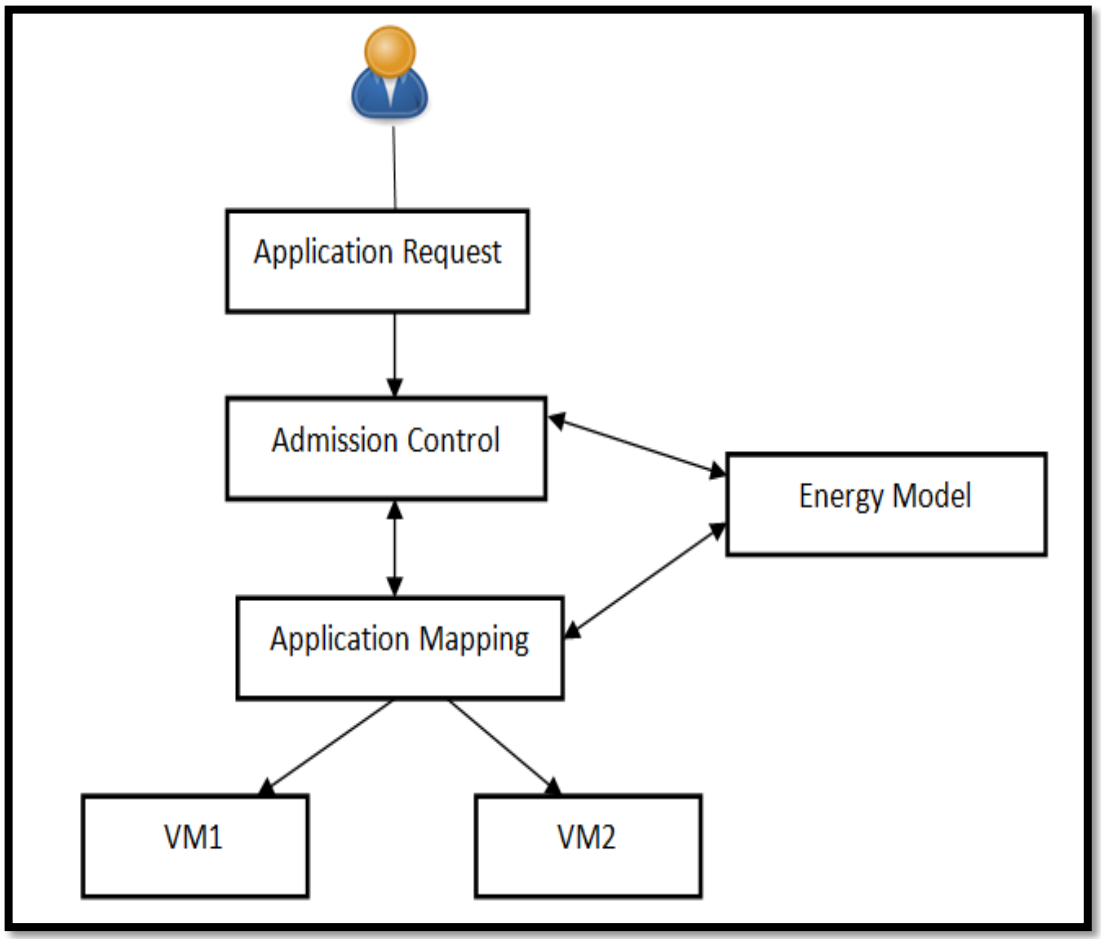


Figure 6.3: The HCIS Workflow

Then, when the application passed the admission control's conditions, the application will be moved to the next stage. The scheduling algorithms are responsible for mapping the application that came from the admission control to a specific VM for execution. Similar to the admission control, the application mapping will interact with the GPU end-to-end energy framework to obtain the application's energy consumption and execution time for both Fermi and Kepler

executions. Based on the assigned policy, the application will be mapped to an appropriate VM.

6.4 System Model

A queue is a list of things, such as a list of people waiting their turn to be served [225]. Queuing Theory is the mathematical process to analyse waiting in the lines or queues [226]. Queuing Theory investigates the waiting time of an item, the serving time of this item, the priority of serving this item and the rules of serving this item. Queuing Theory is widely utilised in computer systems to describe and formulate the contention of hardware or software components. Moreover, queuing theory manages the aforementioned components through scheduling policies, such as FCFS. Queuing theory can be utilised to increase functionalities of systems and services in various ways, such as increasing the response time of the system

The queue system approach is used to develop the structure of HCIS combined with a policy-based approach that contains several rules to reduce the energy consumed by executing applications. HCIS aims also to maintain the QoS and reduce SLA violations. Figure 6.4 illustrates the queue structure of the HICS.

After the application passed the admission control's requirements, the application will be put in the arriving queue. In this queue, the application will be served based on its location in the queue in the First Input First Output (FIFO) technique. Then, the application mapping will select the appropriate queue to send the application. In this stage, there are two types of mapping algorithms have been developed. The first algorithm will consider energy consumption as the main parameter. This algorithm is called the Energy-Aware (EA) algorithm. The second algorithm will consider the deadline constraint alongside energy consumption. This algorithm is called the Energy and Deadline Aware (EADA) algorithm. There are two dispatching queues for each VM. The Primary queue is the queue for the applications where they will be executed in the VM supported by Kepler GPU. The second queue named the Secondary queue which is the queue for the applications that will be executed in the VM supported by Fermi GPU. Since the VM supported by Kepler is

more energy-efficient, the goal is to run applications on this VM as much as possible with maintaining the QoS.

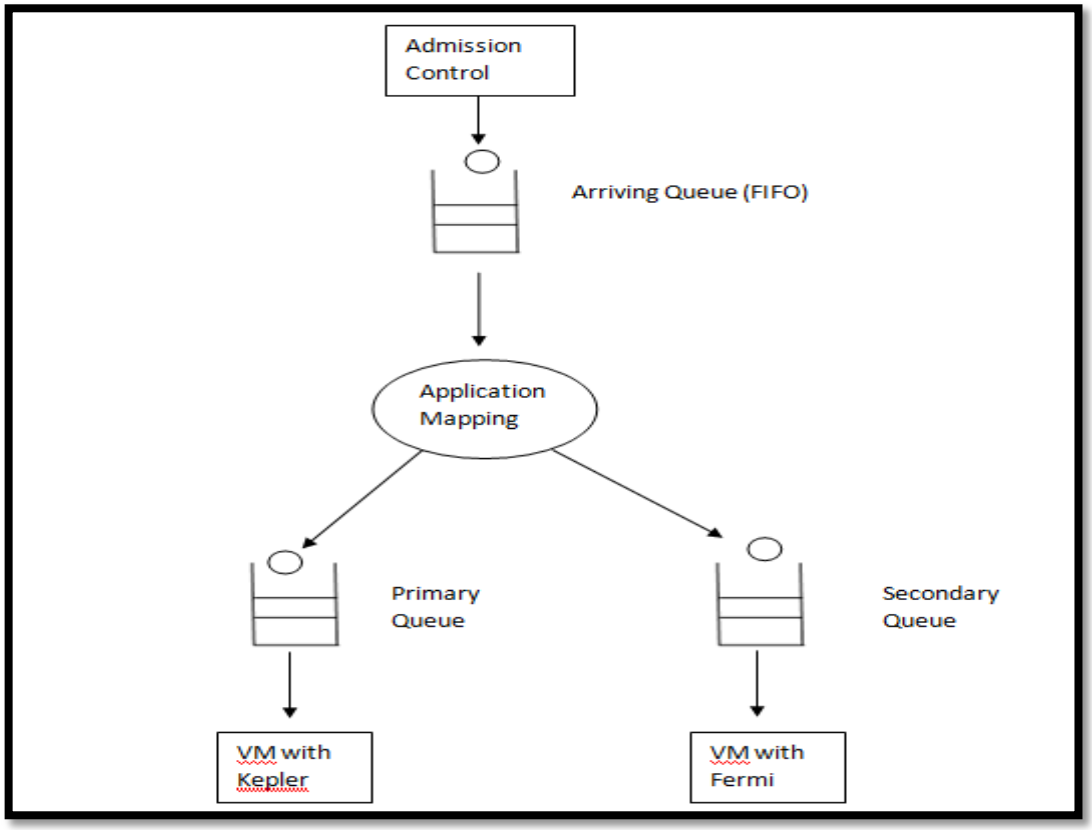


Figure 6.4: the Queue Structure

The notations used to develop the algorithms (EA and EADA) are:

GPU_K Kepler GPU

GPU_F Fermi GPU

Ar [ ] Arrival applications array

Pr [ ] Primary array for execution on GPU_K

Sec [ ] Secondary array for execution on GPU_F

a Application name

s Application size

D User deadline

R Application's arrival time

T Time due to the application to be finish

ST_F Start time of the application for execution on GPU_F

ST_K Start time of the application for execution on GPU_K

W_F Application waiting time on Sec [ ]

W_K Application waiting time on pr [ ]

C_F Completion time of the application on GPU_F

C_K Completion time of the application on GPU_K

TiS_F Time in the system on GPU_F

TiS_K Time in the system on GPU_K

DT_F The difference between the completion time and time due of the application on GPU_F

DT_K The difference between the completion time and time due of the application on GPU_K

DT_F^{tot} The total difference between the completion time and time due of all applications in GPU_F

DT_K^{tot} The total difference between the completion time and time due of all applications in GPU_K

exe_F Application execution time on GPU_F

exe_K Application execution time on GPU_K

E_F Application energy consumption on GPU_F

E_K Application energy consumption on GPU_K

E_{Diff} Energy threshold percentage, the energy difference between E_F and E_K

E_F^T Total energy consumption of applications executed on GPU_F

E_K^T Total energy consumption of applications executed on GPU_K

E_{tot} Total energy on both GPU_F and GPU_K

srv_F Application service time in GPU_F

srv_K Application service time in GPU_F

Suppose an application *a* ∈ A = { $a_1, a_2, a_3, \dots, a_n$} is submitted by the Cloud end-user. Each application *a* is associated with specific SLA requirements, such as deadline D ∈ R^+ and arrival time Ar ∈ R^+. the application *a* needs to be executed in a virtual machine *vm* ∈ VM= { $vm_1, vm_2, vm_3, \dots vm_n$ }, each *vm* has different features. The scheduler has the responsibility to allocate a sequence of applications in A to vm_1 or vm_2. The proposed scheduling policy EA has an essential objective to

reduce. This objective is to reduce the energy consumption $e \in R^+$ of the executed application a. The second proposed scheduling policy EADA has two objectives.

The second scheduling policy EADA aims to balance the energy e and the time due T ϵR^+, which can be calculated by:

$$T_a = D_a + R_a \quad (1)$$

Each interactive or batch application has special stochastic Characteristics for a request, such as an arrival time, service time and SLA requirements, e.g. deadline. To obtain the application's starting time for execution, the following equation is used:

$$ST_a = \begin{cases} 0, & if\ a_i = 1 \\ \max(R_a, C_{pr}), & otherwise \end{cases} \quad i = 1,2,\dots,n \quad (2)$$

Where C_{pr} is the completion time of the previous application on Pr [ ] or Sec [ ]. If the application has the first order in the queue, its starting time is 0. Otherwise, the starting time is the maximum value between the application's arrival time and the previous completion time in the same array, Pr [ ] or Sec [ ]. This can be applied for ST_F or ST_K

To obtain the application's completion time when it is executing in GPU_K or GPU_F, the following equation is used:

$$C_a = ST_a + srv_a \quad (3)$$

Where the estimated execution time obtained by the GPU end-to-end energy framework is considered as the service time. The completion time is calculated based on the VM that will execute the application.

To obtain the application's waiting time, the following equation is used:

$$W_a = ST_a - R_a \quad (4)$$

Where the application's starting time value is based on its location in Pr [ ] or Sec [ ].

To calculate the application's time in the system since it arrived and completed the execution, the following equation is used:

$$TiS_a = C_a - R_a \quad (5)$$

Where the completion time is calculated based on the VM that will execute the application.

To obtain the difference between the completion time and time due to the application, the following formula is used:

$$DT_a = C_a - T_a \quad (6)$$

Where the completion time is calculated based on the VM that will execute the application. DT_a indicates to imagine how far the application's execution time Chronologically from the deadline.

6.4.1 Assumptions

This sub-suction defines the main assumptions of developing scheduling policies and admission control. These assumptions are:

- HCIS is responsible to manage a PaaS platform that consists of several different CUDA applications. Hence, the user is only responsible to submit the application and its size.
- Each VM can execute only one application 1:1.
- The deadline is the selected factor because it is an important parameter of SLA requirements to maintain.
- The waiting time will be counted when the application arrived in a dispatching queue (primary or secondary). The arriving queue has a negligible waiting time.

6.5 System Design

6.5.1 Admission Control

After the user submitted the application containing SLA requirements like the deadline to the Cloud, admission control is responsible to accept or reject the service of this application execution. This decision is based on the application's requirements and the available Cloud resource that can fit the application's requirements. Admission control is used to balance the usage of the Cloud infrastructure and to mitigate the servers' saturation which can produce

performance degradation. Additionally, the admission control manager can reduce SLA violations.

The admission control, as highlighted in Algorithm 6.1, requires several inputs. The required inputs are the application's name, the application's size, user deadline and the application's arrival time. The arrival queue has been initialised with a null value to put the applications that passed the admission control's conditions. For a proactive check, the admission control communicates with the energy model in the GPU end-to-end framework to get the estimated application's execution time whether it is executed on the VM with $\mathrm{GPU_F}$ or the VM with $\mathrm{GPU_K}$, as stated in line 2. There are two essential requirements for the application to accept. The first requirement is the user's deadline should be greater than the estimated execution time on the VM with $\mathrm{GPU_F}$ or the VM with $\mathrm{GPU_K}$, as stated in line 3. So, the application should be admitted when its deadline is greater than only one of the existing service time ($\mathrm{GPU_F or\ GPU_K}$). If the deadline is greater than both service times in the VM with $\mathrm{GPU_F}$ and the VM with $\mathrm{GPU_K}$, the deadline will be considered as a soft deadline. On the other hand, if the deadline is only greater than the service time of the VM with $\mathrm{GPU_K}$, the deadline will be considered as a hard deadline because the Cloud has a limited time for executing the admitted application. However, the admission control will reject the requested application when the user's deadline is smaller than both the VM with $\mathrm{GPU_F}$ and the VM with $\mathrm{GPU_K}$ service time. Then, if the application passed this condition, it will move to the next requirement. Admission control will contact the scheduler to find the application's starting time for calculating completion time in the VM with $\mathrm{GPU_F}$ and the VM with $\mathrm{GPU_K}$. Moreover, the time due will proactively be calculated for the next condition. The second requirement in the admission control is the time due to the application should be greater than or equal the completion time on the VM with $\mathrm{GPU_F}$ or the VM with $\mathrm{GPU_K}$, in line 10. If the application passed this condition, it directly will be put in the arriving queue. However, if the application did not correspond to this condition, a new time due negotiation will be started, line 13. Hence, the admission control policy eliminates application with a rigid deadline.

Algorithm 6.1: Admission Control

Input: a, s, D, R.

Output: Ar [ ]

1: set Ar [ ] = null
2: Estimate exe_F(a,s), exe_K(a,s)
3: **if** (D > exe_F(a,s)) || ((D> exe_K(a,s)) **then**
4: Go to the next check (line 10)
5: **else**
6: Reject a
7: **endif**
8: Find $ST_F(a,s), ST_K(a,s)$
9: Calculate T (a,s), $C_F(a,s), C_K(a,s)$
10: **if** (T (a,s) >= $C_F(a,s)$) || (T (a,s) >= $C_K(a,s)$)**then**
11: Enqueue a into Ar[ ]
12: **else**
13: Negotiate with the user for flexible T (a,s)
14: **if** (user accept) **then**
15: Enqueue a into Ar[ ]
16: **else**
17: Reject a
18: **endif**
19: **endif**

A flexible time due means expanding the time due and let the Cloud completes executing the application as soon as possible. Moreover, with a flexible time due, the Cloud provider can avoid SLA violation. If the Cloud user accepts the negotiations, the requested application will be accepted and put in the arriving queue. Otherwise, the application will be rejected.

6.5.2 Energy-Aware Algorithm

A single Cloud data centre consumes a considerable quantity of electricity that increases operational costs. According to the continuous proliferation of Cloud data centres since the increased demands of Cloud services, energy consumption and

CO2 emission have become substantial issues. Therefore, Cloud providers are on the hunt for efficient ways to mitigate the impact of Cloud data centres on energy consumption and the environment. They request for developing sustainable management techniques for their Cloud infrastructures.

After the application's completed the requirements of the admission control, the application will be located in the application mapping stage. In this stage, the application will be dispatched to a selected VM by the Energy-Aware (EA) algorithm. In general, the EA algorithm, as highlighted in Algorithm 6.2, consists of three main sections. The first section is responsible for the application selector policies (lines 3-14). The second section is responsible for Pr [] queue that manages applications that will be executed in the VM connected with GPU_K (lines 15-23). The final section is responsible for Sec [] queue that manages applications where will be executed in the VM connected with GPU_F (lines 24-31). EA requires the application's name, application's size, the user deadline, the application's arrival time and the arriving queue as inputs. Pr and Sec queues have been initialised with null values to put the applications for execution in the VM with $\mathrm{GPU_F}$ and the VM with $\mathrm{GPU_K}$. A static energy threshold defined by the Cloud provider is created. For every application Ar queue, EA interacts with the energy model in the GPU end-to-end framework by sending the application's name and size to estimate the application's execution time whether it is executed on the VM with $\mathrm{GPU_F}$ or the VM with $\mathrm{GPU_K}$. Then, these estimated execution times are set as service times. EA also interacts with the energy model by sending the application's name and size to estimate the energy consumed by the application whether executed in the VM with $\mathrm{GPU_F}$ or the VM with $\mathrm{GPU_K}$, as stated in line 7. The energy difference and time due are calculated. To calculate the energy difference for each application, the following equation is used:

$$E^{a}_{Diff} = \frac{E^{a}_{F} - E^{a}_{K}}{E^{a}_{F}} \text{ x } 100 \qquad (7)$$

Where E^{a}_{F} is the application energy consumption when executed by VM with $\mathrm{GPU_F}$ and E^{a}_{K} is the application energy consumption when executed by the VM with $\mathrm{GPU_K}$. The decision of sending the application to whether the VM with $\mathrm{GPU_K}$ or the VM with $\mathrm{GPU_F}$ is mainly made by the comparison between the

threshold and E^{a}_{Diff} . This comparison assumes that if the value of E^{a}_{Diff} exceeds the energy threshold, the energy cost is high. To mitigate this cost, the application should be sent to the most energy-efficient VM, in this case, the VM with $\mathrm{GPU_K}$ support is the most energy-efficient VM (lines 9-12).

After the application has been dispatched to the Pr queue, each application will be in processing steps before its execution. For every application in the Pr queue, the algorithm finds the starting time and calculates the completion time. If there is more than an application waiting for execution, the algorithm will select the application with the smallest time due to meet the user deadline and reduce the overdue time. Then, the algorithm calculates W_K, TiS_K and DT_K for every application executed on the VM with $\mathrm{GPU_K}$. Then, E^{T}_{K} is calculated by using the following formula:

$$E^{T}_{K} = \sum_{i=0}^{n} E_K \qquad (8)$$

Where n is the number of applications executed on the VM wit $\mathrm{GPU_K}$.

Similarly, the applications that dispatched from Sec queue to be executed in the VM with $\mathrm{GPU_F}$ will be processed similar to the applications that dispatched from Pr queue.

Algorithm 6.2: Energy Aware

Input: a, s, D, R, Ar [ ]
Output: Pr [ ], Sec [ ]
1: set Pr [ ], Sec [ ] = null
2: set threshold =assigned by the provider
3: **for each** a in Ar [ ] **do**
4: Estimate exe_F(a,s), exe_K(a,s)
5: Set srv_F = exe_F(a,s)
6: Set srv_K = exe_K(a,s)
7: Estimate E_F(a,s), E_K(a,s)
8: Calculate E_{Diff} (a,s), T (a,s),
9: **if** (E_{Diff}(a,s) >= threshold) **then**
10: Enqueue a into Pr[ ]
11: **else**
12: Enqueue a into Sec [ ]

13: **end if**
14: **end for**
15: **for each** a in Pr [ ] **do**
16: Find $ST_K(\mathrm{a},\mathrm{s})$
17: Calculate $C_K(\mathrm{a},\mathrm{s})$
18: **while** current $C_K(\mathrm{a},\mathrm{s})$ > remaining R (a,s) **do**
19: Dequeue the smallest T (a,s) first
20: **end while**
21: Dequeue (a,s)
22: Calculate $W_K(a,s)$, TiS_K (a,s), DT_K (a,s), E_K^T
23: **end for**
24: **for each** a in Sec [ ] **do**
25: Find $ST_F(\mathrm{a},\mathrm{s})$
26: Calculate $C_F(\mathrm{a},\mathrm{s})$
27: **while** current $C_F(\mathrm{a},\mathrm{s})$ > remaining R (a,s) **do**
28: Dequeue the smallest T (a,s) first
29: **end while**
30: Dequeue (a,s) to GPU_F
30: Calculate $W_F(a,s), TiS_F(a,s), DT_F$ (a,s) , E_F^T
31: **end for**

6.5.3 Energy and Deadline Aware

The previous algorithm was focusing on reducing the energy consumed by application execution as a particular objective of concern.

SLA contracts determine the requirements and specifications of Cloud users. Examples of these requirements are availability, response time, types of allocated VMs and deadline. These requirements create thresholds used for appropriate scheduling. The Cloud provider strives to meet these requirements to avoid SLA violations or even to mitigate these violations. In this work, the considered parameter used for developing the algorithm is the deadline. The deadline can be defined as the maximum allowed time to complete the application execution requested by the user. It can be measured by a unit of time like Seconds (S).

Adding deadline constraint as a second objective in the scheduling algorithm whilst reducing the energy consumption in heterogeneous Cloud infrastructures will increase the complexity. Therefore, dynamic scheduling for placing applications to maintain the deadline and reduce energy consumption or even to produce an adequate trade-off between energy and performance whilst maintaining the QoS has been developed.

Similar to EA, EADA, as illustrated in Algorithm 6.3, includes three main sections. These sections include the selector procedure, Pr queue operations and Sec queue operations. The selector section dynamically decides the appropriate VM to execute the application (lines 2-18). Then, the second section is responsible for managing applications in the Pr queue where will be executed in the VM connected with $\mathrm{GPU_K}$ (lines 19-27). Finally, the third section is responsible for managing applications in the Sec queue where will be executed in the VM connected with $\mathrm{GPU_F}$.

To begin the process of placing applications, the application's name, the application size, the user deadline, the application's arrival time and the arriving queue are required as inputs. Pr and Sec queues are declared as null values to put the applications for execution in the VM with $\mathrm{GPU_F}$ and the VM with $\mathrm{GPU_K}$. For every application in the Ar queue, the algorithm communicates with the energy model located in the GPU end-to-end energy framework by sending the application's name and size to estimate the application's execution time whether it is executed on the VM with $\mathrm{GPU_F}$ or the VM with $\mathrm{GPU_K}$. After the response of the energy model, the estimated execution times are set as service times. EADA also communicates with the energy model by sending the application's name and size to estimate the energy consumed by the application where it is executed in the VM with $\mathrm{GPU_F}$ or the VM with $\mathrm{GPU_K}$. Then, the starting time due is calculated, and starting times are found. According to the estimated execution times, EADA can calculate C_K and ST_K for an application if it will be executed in the VM with $\mathrm{GPU_K}$; EADA also can calculate C_F and ST_F. The decision will be made if the application passes one of the following conditions (line 11-17). If DT_K is smaller than or equal 0, which means that the application will be completed in VM with $\mathrm{GPU_K}$ before its assigned deadline, then the application will be sent to the Pr queue. If DT_K is

greater than DT_F, which means that if the application executed in the VM with $\mathrm{GPU_F}$, it will be completed before the execution of the VM with $\mathrm{GPU_K}$, then the application will be sent to the Sec queue. Otherwise, if DT_K is smaller or equal than DT_F , then the application will be sent to Pr to reduce the time that will exceed the deadline.

After the application has been dispatched to the Pr queue, each application will be passing through some processing steps before its execution. For each application in the Pr queue, if there is more than an application waiting for execution, EADA will select first the application that has the maximum energy and smallest time due to complete executing the applications that have great energy consumption. This is because the VM connected with $\mathrm{GPU_K}$ is more energy-efficient than the VM connected with $\mathrm{GPU_F}$. Then, EADA will send the application that has the smallest energy consumption to the Sec queue to reduce its waiting time and can pass its deadline. Otherwise, if there is not any application waiting, the application will be executed based on its order in the queue. Then, the algorithm calculates W_K, TiS_K and DT_K for every application executed on the VM with $\mathrm{GPU_K}$. E_K^T will be calculated by using equation 8.

Similar to Pr, after the application has been dispatched to the Sec queue, each application will be passing through some processing procedures before its execution. For each application in the Sec queue, if there is more than an application waiting for execution, EADA will select first the application that has the smallest time due for reducing the overdue time that will exceed the deadline. Otherwise, if there is not any application waiting in the queue, the application will be executed based on its order in the queue. Finally, W_F, TiS_F, DT_F and E_F^T will be calculated.

Algorithm 6.3: Energy and Deadline Aware

Input: a, s, D, R, Ar [ ]

Output: Pr [ ], Sec [ ]

1: set Pr [], Sec [] = null

2: **for each** a in Ar [] **do**

3: Estimate exe_F(a,s), exe_K(a,s)

4: Set srv_F = exe_F(a,s)

5: Set srv_K = exe_K(a,s)
6: Estimate E_F(a,s), E_K(a,s)
7: Calculate T (a,s)
8: Find $ST_K(a, s)$, $ST_F(a, s)$
9: Calculate $C_K(a, s)$, $DT_K(a, s)$
10: Calculate $C_F(a, s)$, $DT_F(a, s)$
11: **if** ($DT_K(a, s)$ <= 0) **then**
12: Enqueue a into Pr [ ]
13: **else if** ($DT_K(a, s)$ > $DT_F(a, s)$) **then**
14: Enqueue a into Sec [ ]
15: **else if** ($DT_K(a, s)$ <= $DT_F(a, s)$) **then**
16: Enqueue a into Pr [ ]
17: **end if**
18: **end for**
19: **for each** a in Pr [] **do**
20: **while** current $C_K(a, s)$ > remaining R (a,s) **do**
21: **If** ((a,s) has the largest E_K && the smallest T))||((a,s) has the largest E_K)) **then**
22: Dequeue (a,s) first
23: Send the smallest E_K(a,s) to Sec [ ]
24: **end while**
25: Dequeue (a, s)
26: Calculate W_K(a, s), $TiS_K(a, s)$, DT_K(a,s), E_K^T
27: **end for**
28: **for each** a in Sec [] **do**
29: **while** current $C_F(a, s)$ > remaining R (a,s) **do**
30: Dequeue the smallest T (a,s) first
31: **end while**
32: Dequeue (a, s)
33: Calculate W_F(a, s), $TiS_F(a, s)$, $DT_F\ (a, s)$, E_F^T
34: **end for**

6.6 Evaluation

6.6.1 Performance Metrics

To evaluate the developed algorithms (EA and EADA), there are two major criteria are used: performance metrics and total energy. Performance metrics include:

- Average Waiting Time ($\boldsymbol{W_{avg}^{T}}$): measures the time that the application spends before it be processed. W_{avg}^{T} is calculated by equation 9:

  $W_{avg}^{T} = \sum_{i=0}^{n} \frac{W_a}{n}$ (9)

  Where n is the total number of applications performed by VMs.

- Average time in the system ($\boldsymbol{TiS_{avg}^{T}}$) measures the average of the overall time of the applications since they arrived until it executed. TiS_{avg}^{T} is determined by equation 10:

  $TiS_{avg}^{T} = \sum_{i=0}^{n} \frac{TiS_a}{n}$ (10)

- The utilisation percentage of VMs that connected with Fermi and Kepler GPUs **($\boldsymbol{Uti_F}$, $\boldsymbol{Uti_K}$)** checks the magnitude of the VM utilisation. The following formula is used to measure Uti_F and Uti_K:

  $Uti = \frac{\sum_{i=0}^{n} srv}{l}$ X100 (11)

  Where l represents the last application's completion time executed in the VM. The values of srv and l depend on the performed VM.

- The total Overdue existing in GPU_F and GPU_K($\boldsymbol{OVD_{tot}}$) calculates the amount of time that exceeds the deadline. $\boldsymbol{OVD}_{tot}$ is referred to equation 12:

  $$\boldsymbol{OVD}_{tot} = \begin{cases} 0, & if \quad DT_F^{tot} + DT_K^{tot} = 0 \\ DT_F^{tot} + DT_K^{tot}, & otherwise \end{cases} \quad (12)$$

  OVD_{tot} is countable if the value of the summation of $DT_F^{tot}$and DT_K^{tot} are greater than 0. Otherwise, the results will be 0 which mean there is no missing time.

The second criterion is the total energy consumption ($\boldsymbol{E_{tot}}$) which calculates the full amount of the energy consumed by executing the applications. To obtain E_{tot}, equation 13 is utilised:

$E_{tot} = E_F^T + E_K^T$ (13)

6.6.2 Experimental Setup

Several methods can be used to evaluate the performance scheduling algorithms like direct experiments, analytical and simulation. The selected method that used

to evaluate the developed EA and EADA algorithms is the simulation method since it has several advantages, such as:

- Performing experiments with low cost and time in comparison with direct experiments [28];
- conducting a large number of different experimental scenarios;
- eliminating experiments interruption; and
- The ability to analysing and investigating the effect of updating a single variable on other several constant variables.

The purpose of performing simulation experiments is to evaluate the placement of CUDA applications into two VMs composed of heterogeneous resources by using EA and EADA scheduling algorithms. Each algorithm will be compared to three different algorithms. These algorithms are First Input First Output (FIFO), Round Robin (RR) and Shortest Job First (SJR) in terms of W_{avg}^{T}, TiS_{avg}^{T}, E_{tot}, Uti_{F} , Uti_{K} and OVD_{tot} since they are well-known scheduling algorithms.

There are a number of simulation frameworks that deal with Cloud computing environments like CloudSim [28], CloudAnalyst [227], iCanCloud [228], NetworkCloudSim [229], EMUSIM [230], GreenCloud [231] and SPECI [232]. However, these simulations tools do not support GPUs in Cloud computing. A new simulation tool called GPUCloudSim [23] is used to simulate a GPU architecture in Cloud computing. However, GPUCloudSim does not support different types of GPU architectures. Other simulation frameworks developed to support GPU accelerators are GPGPUSim [114] and Barra [233]in non-Cloud computing environments. Both are supporting NVIDIA GPUs that can run CUDA codes. Moreover, other simulation frameworks that support HPA (CPU+GPU) are Ocelot [234], FusionSim [235] and Multi2Sim [135].

Thus, computational modelling has been selected to evaluate energy-aware and energy and deadline aware scheduling algorithms and compare them with the aforementioned algorithms. The computational modelling scenarios have been developed using Microsoft Excel.

6.6.3 Design of experiments

Several scenarios have been implemented to evaluate the developed algorithms in HICS (EA and EADA). These algorithms are cooperating with the energy model in the GPU end-to-end energy framework to obtain the estimated energy and execution time. So the energy consumption and execution time for all applications are estimated in these scenarios. All applications are assumed to be accepted by the admission control even in normal or flexible modes. Each scenario will be evaluated with 10 applications selected randomly based on the applications that used to train and evaluate the GPU end-to-end energy framework, as shown in section 5.5.

The algorithms are examined into two main types of applications groups. First, mixed applications without specific size restrictions have been randomly selected from the applications set. Second, the applications have been classified in a cluster of applications based on their execution time where they executed in the VM connected with $\mathrm{GPU_K}$ into small, medium and large groups with a random selection. The application will be considered as a small type if the execution time ranges between 0 and 100 seconds to evaluate the proposed scheduling policies in different scenario applications sizes. The application will be considered as a medium type if the execution time ranges between 101 and 500 seconds. Moreover, the application will be considered as a large type if its execution time is greater than 500 seconds. Table 6.1 shows the applications set and their classifications.

Table 6.1: The Applications Set and Applications Classification

Application	Size	Type
nbody	1000192	Small
srad	5600	Small
srad	7200	Small
streamcluster	90000	Small
gaussian	9000	Medium
streamcluster	3200000	Medium
gaussian	13000	Large
nbody	2500096	Large

Each group of applications has a different setup. The arriving time of each application is modelled based on Poisson distribution λ=1/$t_{interarriv}$. Each application group has an acceptable inter-arrival time $t_{interarriv}$ which is the time between arrival times to avoid the saturation situation. Deadline is created randomly from a reasonable range in each application type group.

The arriving time of mixed applications is modelled based on Poisson distribution with λ equals 1/3 minutes. The deadline of the mixed applications group is created randomly from the range between 744 and 2185 seconds. Table 6.2 shows the setup details of the mixed applications group.

Table 6.2: The MixedApplications Setup

Application Name	Application Size	Deadline	Deadline Type	Arrival Time	Time Due
gaussian	9000	1004	Soft	0	1004
srad	5600	902	Soft	294	1196
srad	7200	1074	Soft	618	1692
streamcluster	3200000	2061	Soft	721	2782
streamcluster	3200000	2097	Soft	1936	4033
srad	7200	1451	Soft	2218	3669
nbody	1000192	1969	Soft	2343	4312
gaussian	13000	991	Hard	2424	3415
gaussian	9000	1533	Soft	2416	3949
streamcluster	3200000	1481	Soft	2733	4214

The arriving time of the small applications group is modelled based on Poisson distribution with λ equals of 1/1minute. The deadline for small applications is created randomly from the range of 52 and 208 seconds. Table 6.3 illustrates the setup details of the small applications group.

Table 6.3: The Small Applications Group Setup

Application Name	Application Size	Deadline	Deadline Type	Arrival Time	Time Due
srad	7200	112	Hard	0	112
nbody	1000192	154	Hard	87	241
srad	5600	145	Soft	179	324
streamcluster	90000	192	Soft	404	596
srad	7200	190	Hard	646	836
srad	7200	202	Hard	780	982
nbody	1000192	184	Hard	849	1033
srad	7200	116	Hard	944	1060
srad	5600	187	Soft	1084	1271
srad	5600	153	Soft	1242	1395

The arriving time of the medium applications group is modelled based on Poisson distribution with λ equals of 1/2 minutes. The deadline for medium applications is formed randomly from the range of 500 and 644.5 seconds. Table 6.4 demonstrates the setup details of the medium applications group.

Table 6.4: The Medium Applications Group Setup

Application Name	Application Size	Deadline	Deadline Type	Arrival Time	Time Due
streamcluster	3200000	623	Hard	0	623
gaussian	9000	620	Soft	222	842
streamcluster	3200000	560	Hard	409	969
gaussian	9000	536	Soft	670	1206
gaussian	9000	642	Soft	703	1345
gaussian	9000	565	Soft	821	1386
streamcluster	3200000	552	Hard	938	1490
streamcluster	3200000	546	Hard	1314	1860
streamcluster	3200000	503	Hard	1673	2176
gaussian	9000	601	Soft	1730	2331

The arriving time of the large applications group is modelled based on Poisson distribution with λ equals of 1/4 minutes. The deadline for large applications is shaped randomly from the range of 744 and 2285 seconds. Table 6.5 demonstrates the setup details of the large applications group.

Table 6.5: The Large Applications Group Set up

Application Name	Application Size	Deadline	Deadline Type	Arrival Time	Time Due
nbody	2500096	798	Hard	0	798
gaussian	13000	1707	Soft	68	1775
gaussian	13000	1823	Soft	101	1924
nbody	2500096	1421	Hard	816	2237
gaussian	13000	1784	Soft	704	2488
nbody	2500096	939	Hard	1757	2696
nbody	2500096	1832	Hard	1539	3371
gaussian	13000	2073	Soft	1694	3767
nbody	2500096	2014	Hard	2569	4583
nbody	2500096	1800	Hard	2989	4789

6.7 Results

6.7.1 Energy-Aware Scheduling Policy

The overall experimental scenarios aim to test the behaviour of the EA algorithm in different situations. For a comprehensive view, EA considers upper and lower thresholds for applications allocation and energy consumption evaluation to allow the CSP to select the appropriate threshold for energy reduction, as used in [7]. The selected thresholds are ranging between 30% and 60%. These thresholds based on the energy difference of applications when they executed on Fermi or Kepler GPUs. The behaviour of the EA algorithm is evaluated in four different groups of applications. These groups are mixed, small, medium and large applications, as stated in section 6.6.3. In addition, the EA algorithm is compared with FIFO, RR and SJF in terms of the aforementioned criteria (Average Waiting Time, Average Time in the System, Overdue Time, Energy Consumption and VMs utilisation) in each scenario, as stated in section 6.6.1.

6.7.1.1 Experiment 1: Mixed Applications

In terms of average waiting time, SJF has the minimum average waiting time (2.44 seconds) compared to others, while EA-T30%, EA-T40% and EA-T50% thresholds are the highest amount of the average waiting time (154.62 seconds), as shown in Figure 6.5(A).

It can be observed that FIFO performed better than other policies in reducing the average time in the system by 319.25 seconds. In contrast, the proposed EA-T60% policy is the worst value in this criterion with 515.23 seconds, as shown in Figure 6.5(B).

The proposed EA-T30%, EA-T40% and EA-T50%, FIFO and RR do not have an overdue time. However, the proposed EA-T60% is the highest overdue time with 674.48 seconds, as shown in Figure 6.5(C).

Moreover, EA-T30%, EA-T40% and EA-T50% perform well in terms of conserving energy consumption. It consumes 17% less energy than the FIFO policy, which exhibits the second smallest energy value since all applications are executed in the VM connected with GPU_K; this is because the VM with Kepler GPU is more energy-efficient. Figure 6.5(D) shows the total energy consumed by each algorithm with the EA algorithms in the mixed applications group.

Both VMs share the task of applications execution in all algorithms with different utilisation percentages except EA-T30%, EA-T40% and EA-T50%. In EA-T30%, EA-T40% and EA-T50%, the VM connected with GPU_K is merely responsible to execute all applications. Figure 6.5(E) shows the VMs utilisation for each algorithm including the EA algorithms in the mixed applications group.

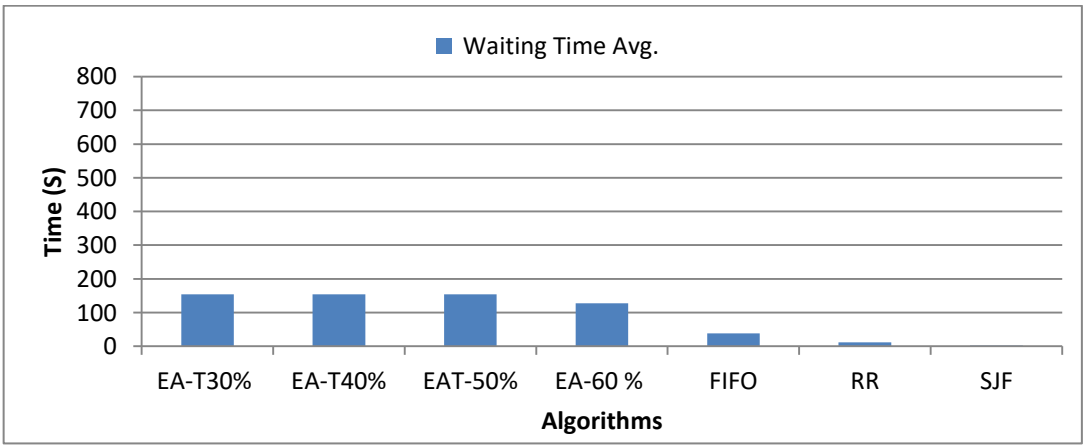


Figure 6.5(A): Average Waiting Time (Mixed Applications-EA)

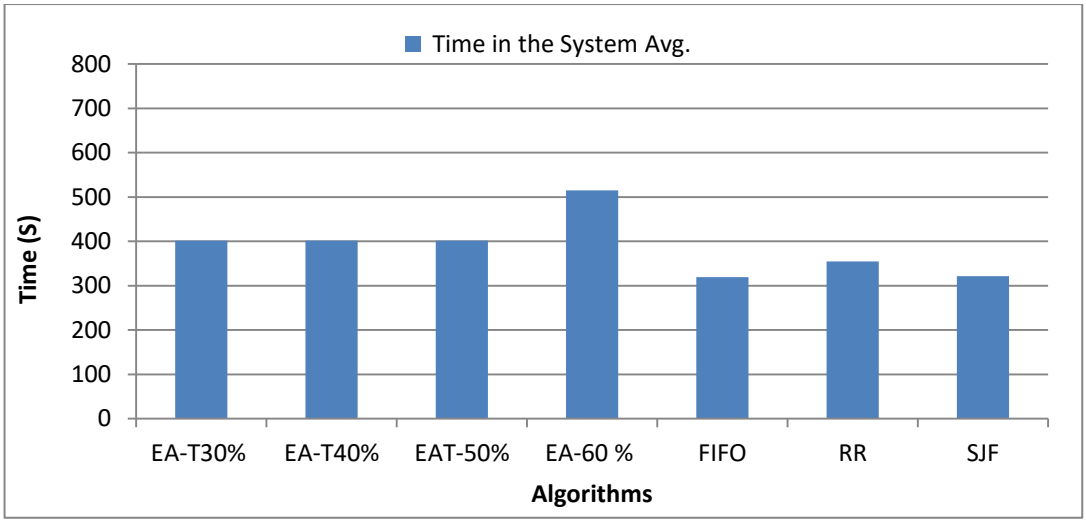


Figure 6.5(B): Average Time in the System (Mixed applications-EA)

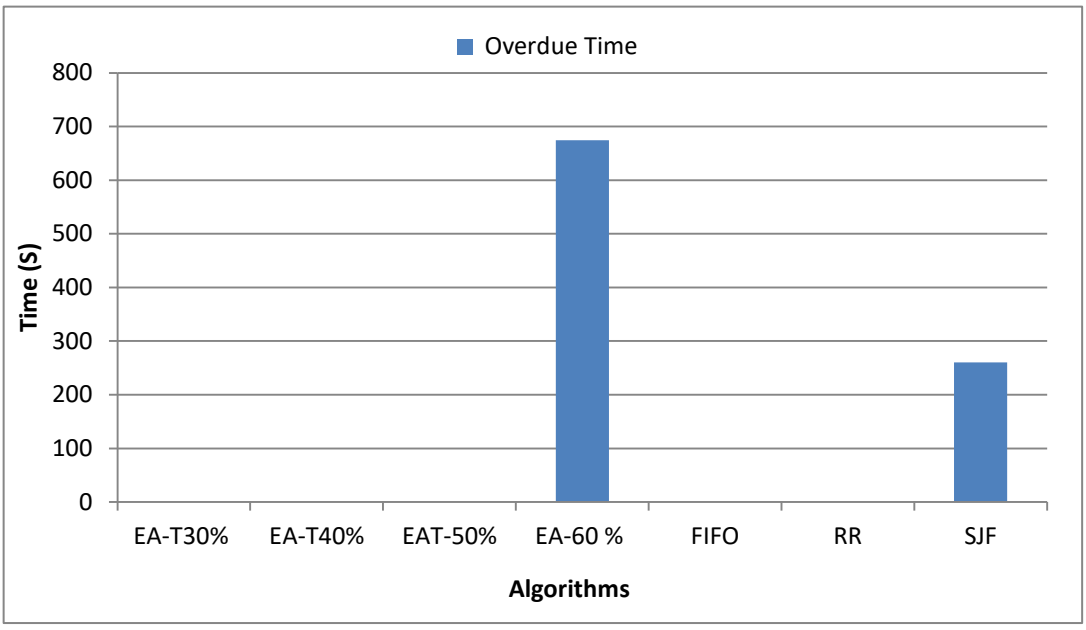


Figure 6.5 (C): Overdue Time (Mixed applications-EA)

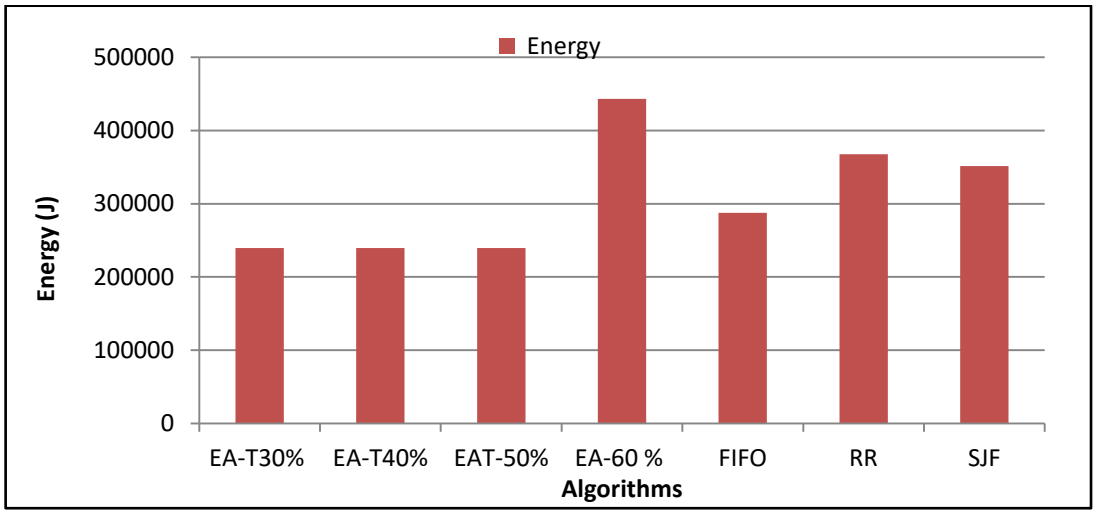


Figure 6.5 (D): Energy Consumption (Mixed applications-EA)

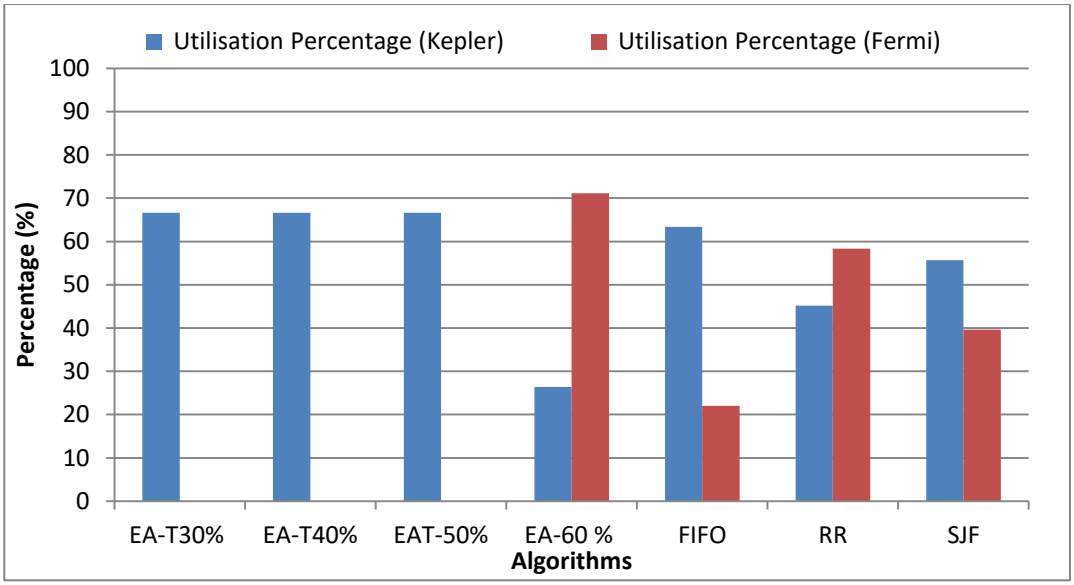


Figure 6.5 (E): VMs Utilisation (Mixed applications-EA)

Figure 6.5: The EA Algorithm Results Evaluated on Mixed Applications Cluster

As shown in Figure 6.5, the proposed EA-T30%, EA-T40% and EA-T50% algorithms have identical behaviour to allocate applications to the VMs in all evaluation factors. To find the optimal point to four factors is complex, so we use energy consumption and the overdue time to obtain endpoints of Pareto frontiers [236]. In the mixed application scenario, the Pareto frontier endpoint point between energy consumption and the overdue time is EA-T60% (674.48, 443246.72).

6.7.1.2 Experiment 2: Small Applications

In the small applications scenario, FIFO and SJF algorithms perform well to eliminate the waiting time. In opposite to FIFO and SJF algorithms, the RR algorithm performs inadequately to reduce the average waiting time (4.39 seconds), as shown in Figure 6.6 (A).

All proposed EA thresholds values have the minimum average time in the system with 74.1 seconds compared to the rest algorithms, but the RR algorithm is the worst for reducing this criterion by 110.72 seconds, as shown in Figure 6.6 (B).

All proposed EA thresholds values have the best performance to eliminate the overdue time among the others. However, the RR algorithm is the maximum overdue time of 142.66 seconds, as depicted in Figure 6.6 (C).

In addition, all proposed EA thresholds values perform well to conserve energy consumption. They consume 44% less energy than the FIFO policy, which exhibits

the second smallest energy value since all applications are executed in the VM connected with GPU_K, which is more energy-efficient VM than another VM. Figure 6.6 (D) shows the total energy values for every algorithm in the small application cluster for EA evaluation.

In this scenario, in FIFO, RR and SJF policies, both VMs share the responsibility of applications execution. However, in all proposed EA thresholds percentage values, the VM connected with GPU_K merely executes the applications, as shown in Figure 6.6 (E).

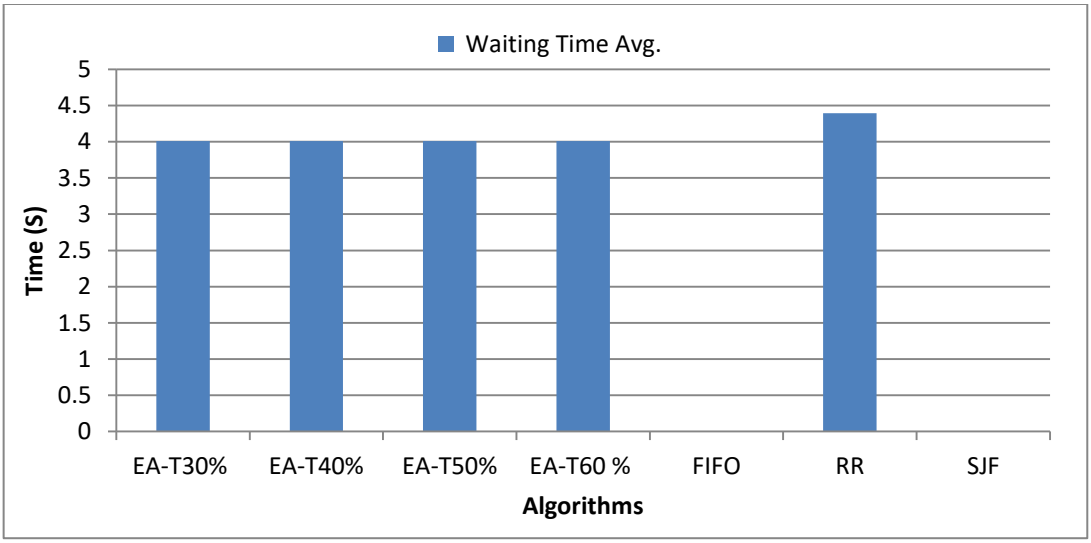


Figure 6.6 (A): Average Waiting Time (Small Applications-EA)

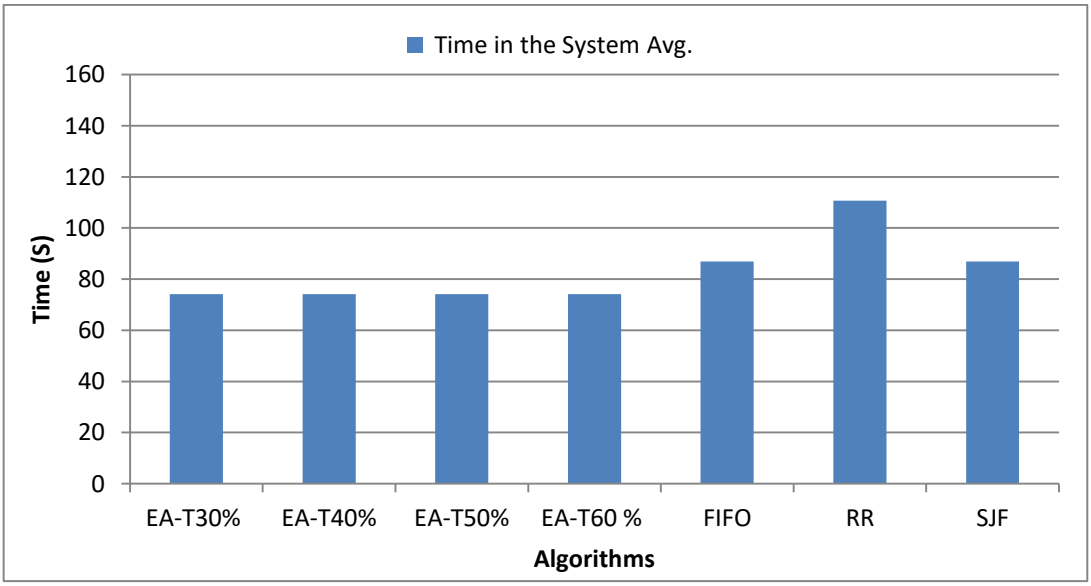


Figure 6.6 (B): Average Time in the System (Small Applications-EA)

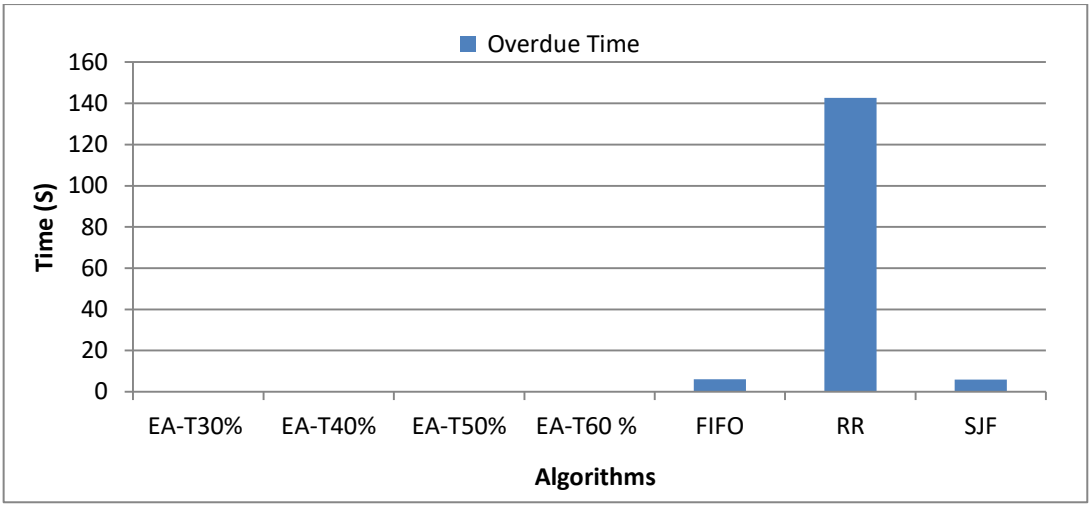


Figure 6.6 (C): Overdue Time (Small Applications-EA)

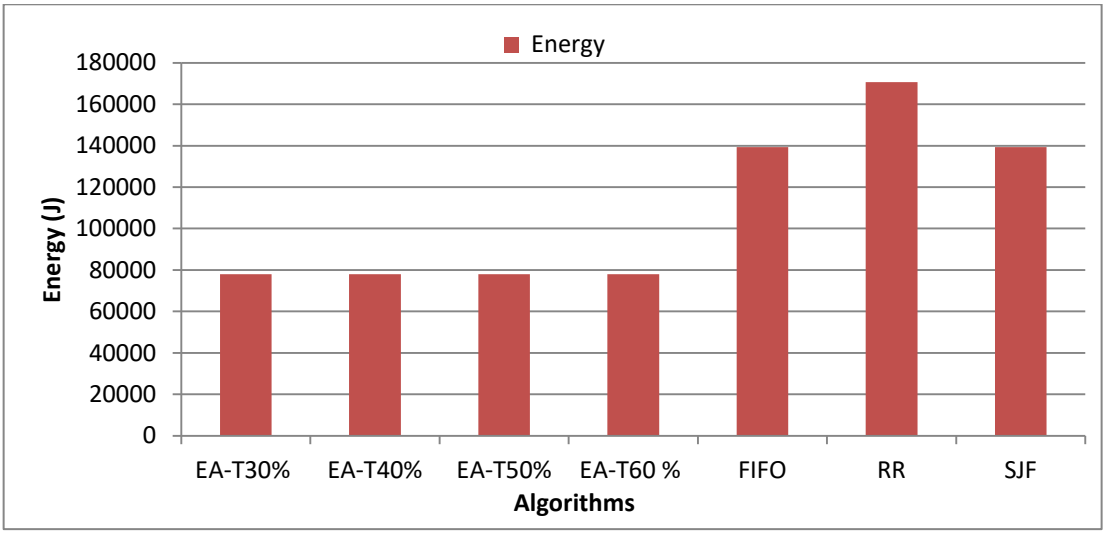


Figure 6.6 (D): Energy Consumption (Small Applications-EA)

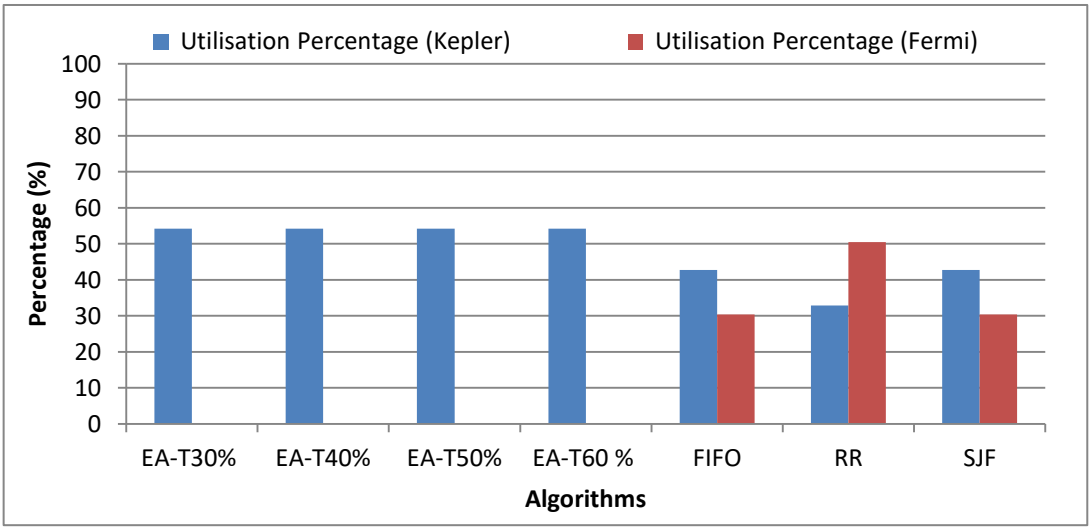


Figure 6.6 (E): VMs Utilisation (Small Applications-EA)

Figure 6.6: The EA Algorithm Results Evaluated on Small Applications Cluster

As shown in Figure 6.6, all proposed algorithms have identical behaviour to allocate applications to the VMs. Therefore, all EA thresholds are optimal for energy and overdue time.

6.7.1.3 Experiment 3: Medium Applications

In terms of the average waiting time, SJF and FIFO policies are the minimum average waiting time with 38.21 seconds compared to the rest policies, while EA-T30%, EA-T40% and EA-T50% are the highest value of the average waiting time (404.75 seconds), as shown in Figure 6.7(A).

FIFO and SJF policies are the minimum value of the average time in the system with 411.15 seconds compared to the rest algorithms. However, the proposed EA-T30%, EA-T40% and EA-T50% are the worst for reducing this criterion with 685.24 seconds since their waiting time is the highest, as shown in Figure 6.7(B).

All algorithms exceed the assigned deadline. However, FIFO and SJF are the minimum value for reducing the overdue time by 229.05 seconds. In contrast, the proposed EA-T60% is the maximum overdue time that missed the deadline with 2549.94 seconds, as depicted in Figure 6.7(C).

EA-T30%, EA-T40% and EA-T50% thresholds percentage perform well in terms of conserving energy consumption. They consume 35% less energy than FIFO and SJF algorithms, which have the same energy consumption value. The reason for this saving is that all applications are executed in the VM connected with GPU_K, which is a more energy-efficient VM. Figure 6.7(D) illustrates the total energy consumed for each algorithm in the medium applications group for EA evaluation.

Moreover, both VMs share the task of executing applications in some algorithms with different utilisation percentages except EA-T30%, EA-T40% and EA-T50%. In EA-T30%, EA-T40% and EA-T50%, the VM connected with GPU_K is merely responsible to execute all applications. Figure 6.7(E) shows the VMs utilisation for each algorithm in the medium applications group for EA evaluation.

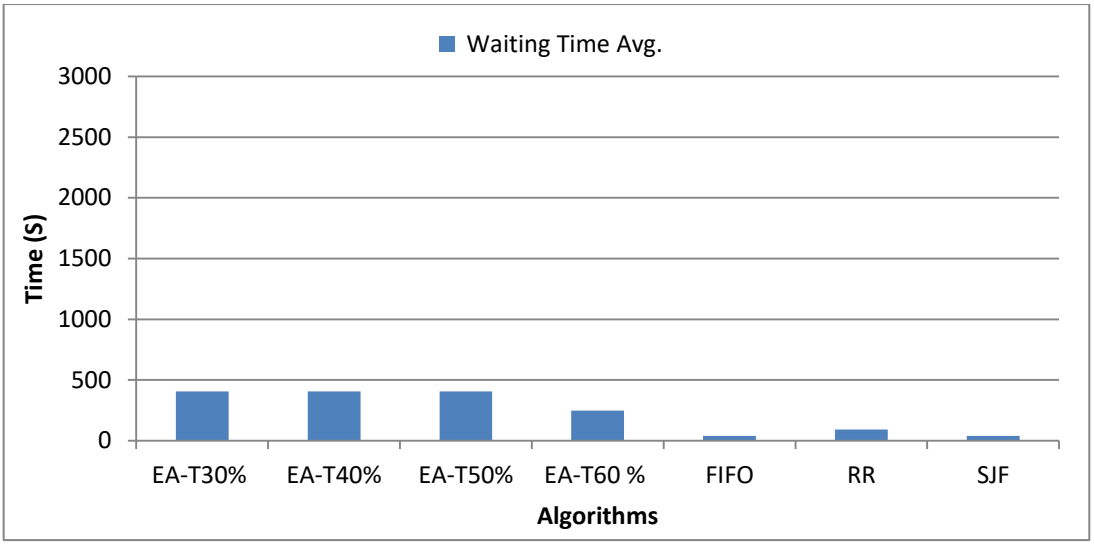


Figure 6.7(A): Average Waiting Time (Medium Applications-EA)

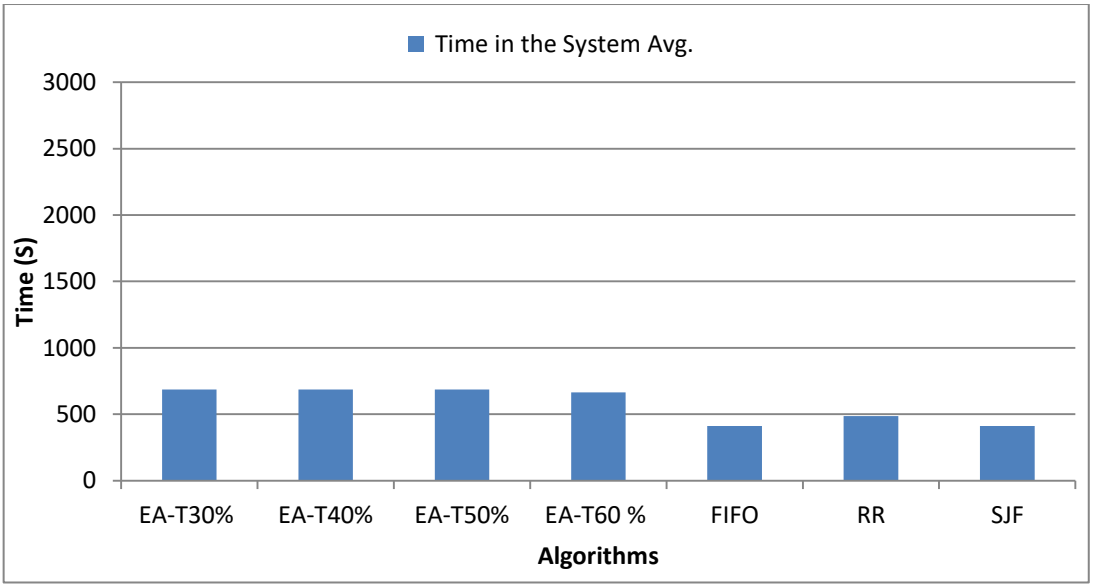


Figure 6.7 (B): Average Time in the System (Medium Applications-EA)

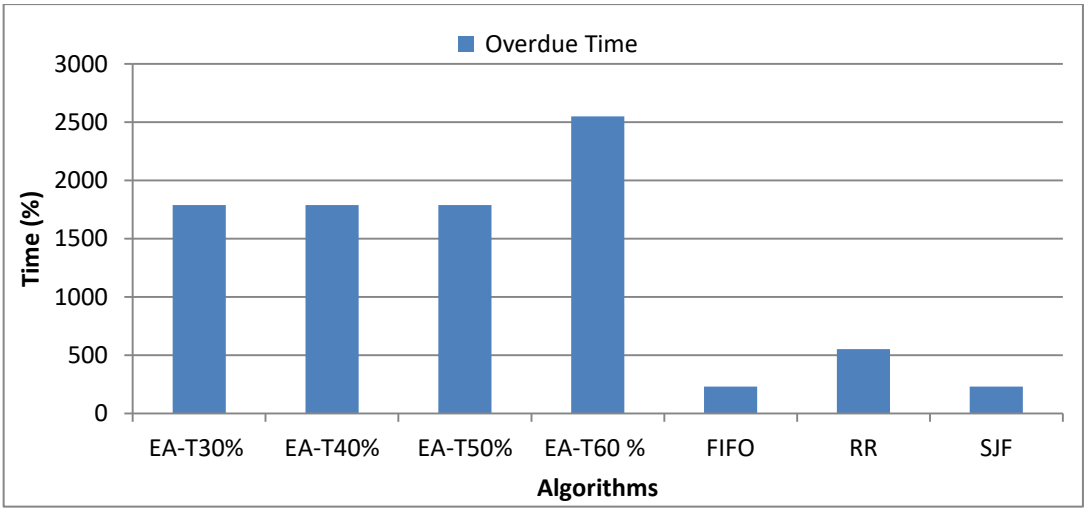


Figure 6.7 (C): Overdue Time (Medium Applications-EA)

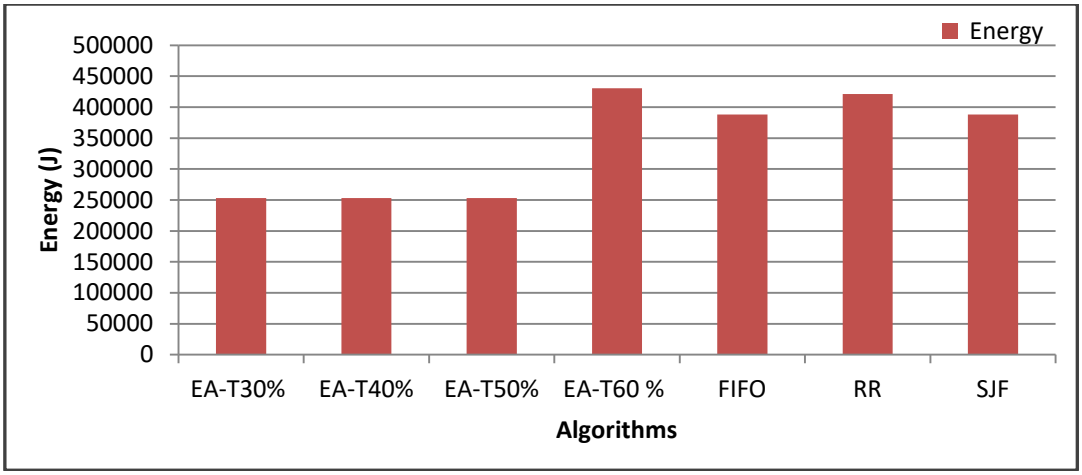


Figure 6.7(D): Energy Consumption (Medium Applications-EA)

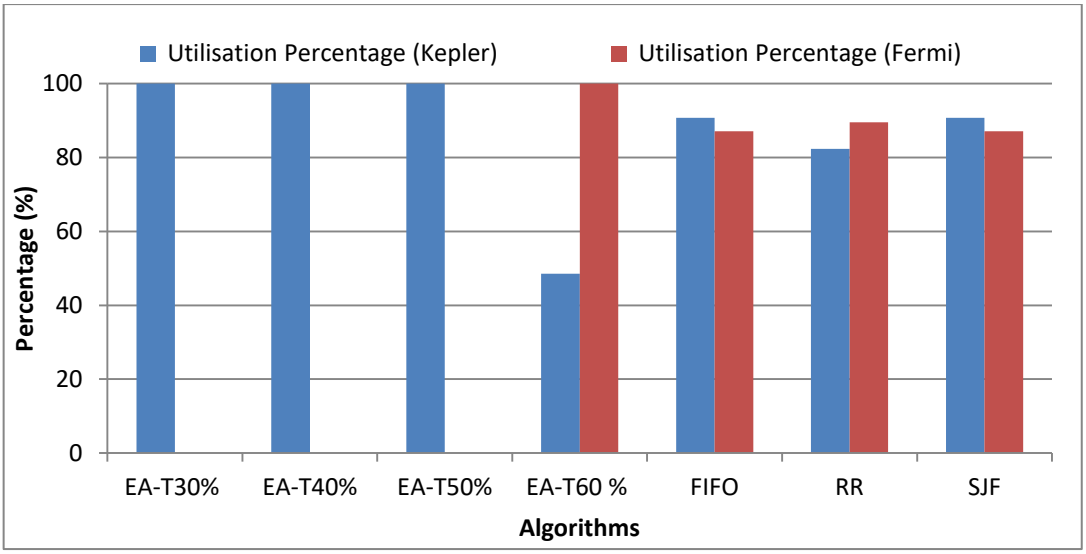


Figure 6.7(E) : VMs Utilisation (Medium Applications-EA)

Figure 6.7: The EA Algorithm Results Evaluated on Medium Applications Cluster

In this scenario, the Pareto front endpoint between energy and the overdue is EA-T 60% (2549.94, 430519.24).

6.7.1.4 Experiment 4: Large Applications

The SJF policy performs well to reduce the average waiting time compared to other algorithms with 450.29 seconds. In opposite to SJF, the proposed EA-T30%, EA-T40% and EA-T50% are the worst among others for reducing the average waiting time by 1580.97 seconds, as shown in Figure 6.8(A).

SJF policy is the minimum value of the average time in the system with 1247.53 seconds compared to the rest algorithms. However, the proposed EA-T30%, EA-

T40% and EA-T50% are the worst for reducing this criterion by 2199.16 seconds since their waiting time is the highest, as depicted in Figure 6.8(B).

Similar to the medium scenario, all algorithms miss the assigned deadline. However, SJF policy performs fine to reduce the overdue time of the assigned deadline by 464.98 seconds. In contrast, the RR policy is the highest overdue time that missed the deadline by 10030.08 seconds, as shown in Figure 6.8(C).

EA-T30%, EA-T40% and EA-T50% perform well in terms of conserving energy consumption. EA-T30%, EA-T40% and EA-T50% consume 25% less energy than the SJF algorithm, which exhibits the second smallest energy value. The reason for this saving is that all applications are executed in the VM connected with GPU_K, which is more energy-efficient VM than another VM. Figure 6.8(D) illustrates the total energy consumed for each algorithm in the large applications group for EA evaluation.

Similar to previous scenarios, Both VMs share tasks to execute applications in all algorithms with different utility percentages except the proposed EA-T30%, EA-T40% and EA-T50%. In EA-T30, EA-T40% and EA-T50% algorithms, the VM connected with GPU_K is merely in charge of executing all applications. Figure 6.8(E) shows the VMs utilisation for each algorithm in the large applications group for EA evaluation.

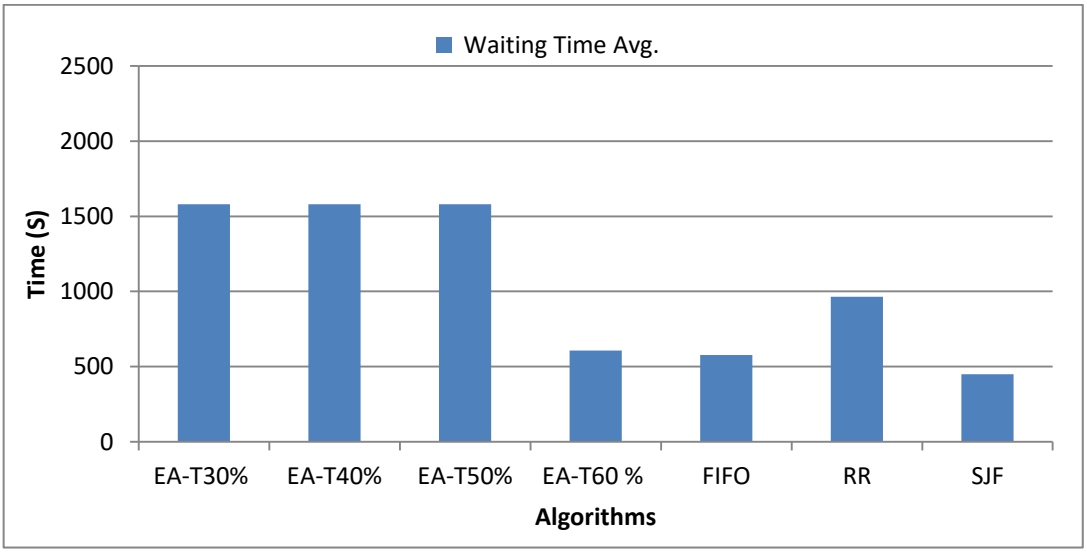


Figure 6.8(A): Average Waiting Time (Large Applications-EA)

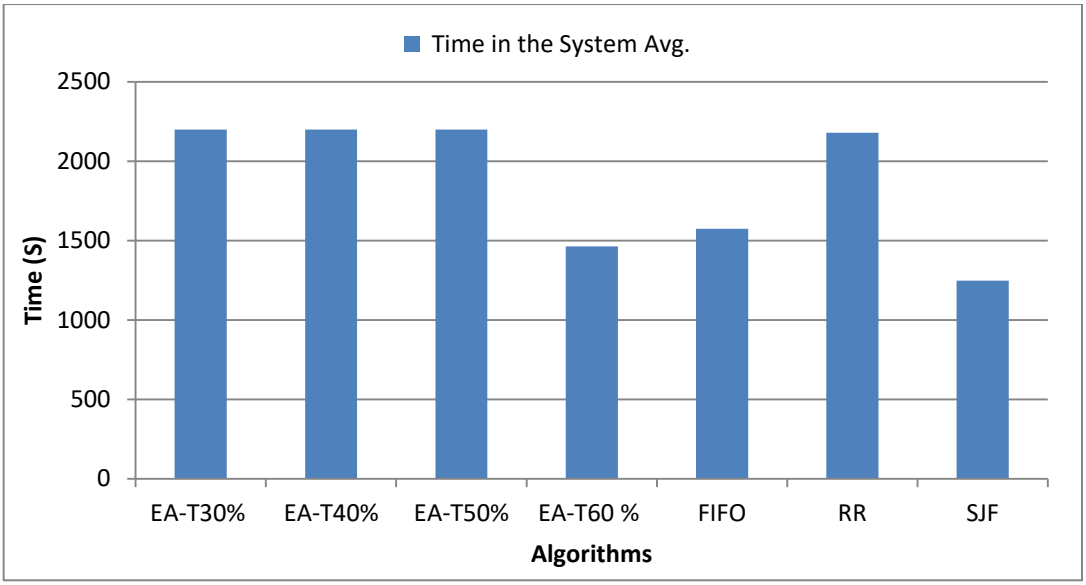


Figure 6.8 (B): Average Time in the System (Large Applications-EA)

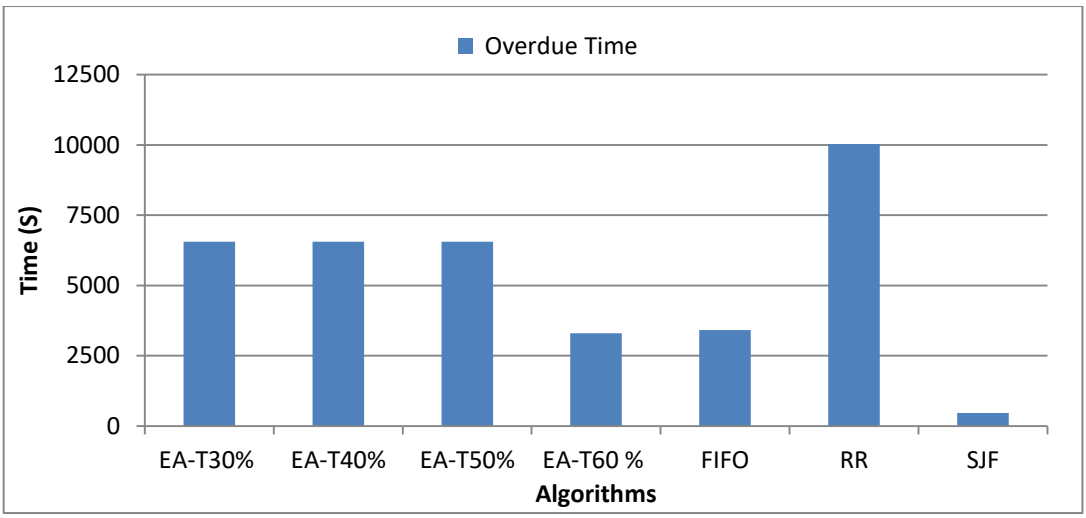


Figure 6.8 (C): Overdue Time (Large Applications-EA)

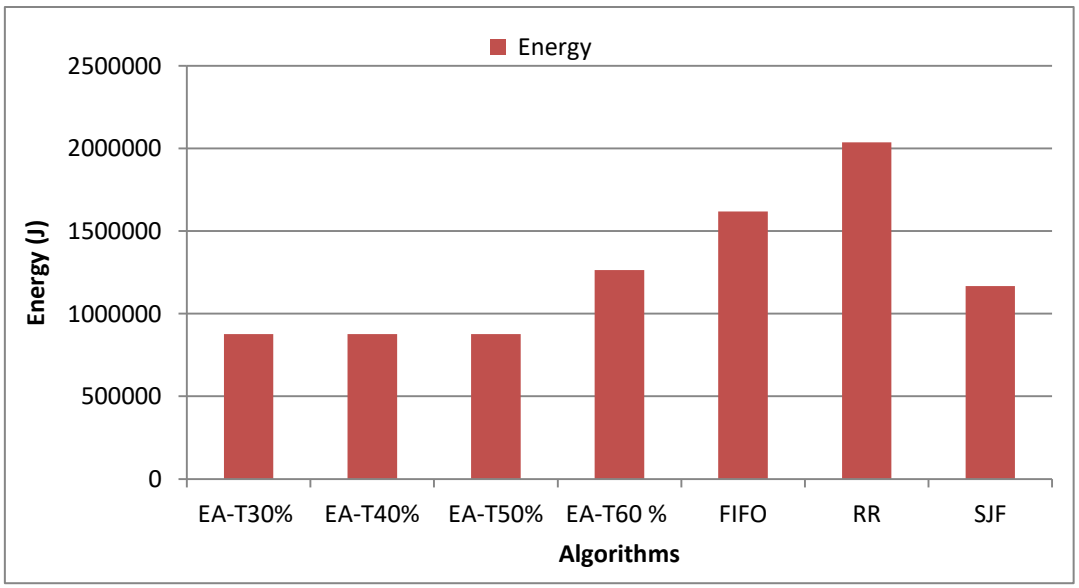


Figure 6.8(D): Energy Consumption (Large Applications-EA)

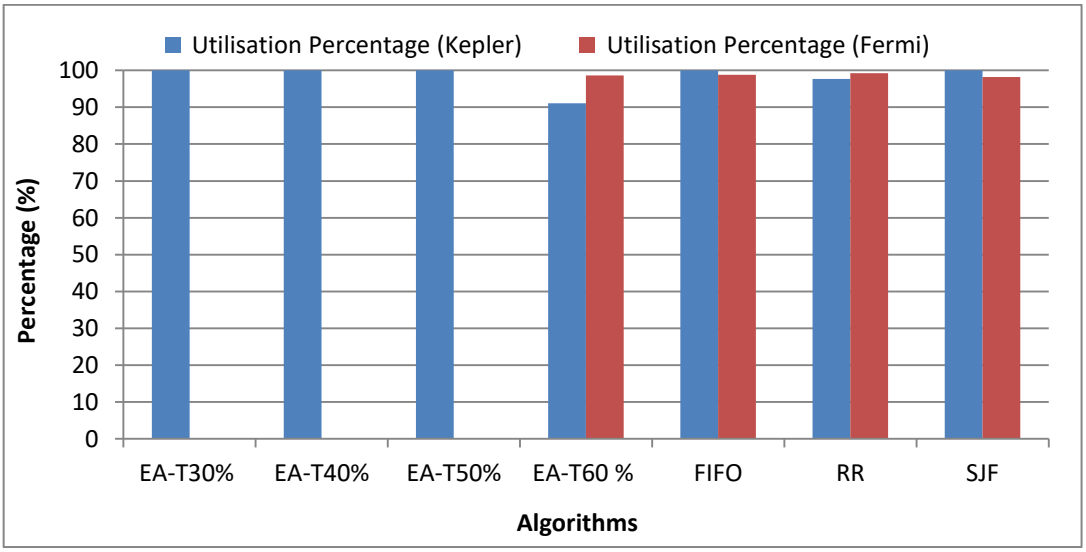


Figure 6.8(E) : VMs Utilisation (Large Applications-EA)

Figure 6.8: The EA Algorithm Results Evaluated on Large Applications Cluster

Figure 6.9 shows the two endpoints of the Pareto front between energy consumption and the overdue time in the large applications of EA evaluation.

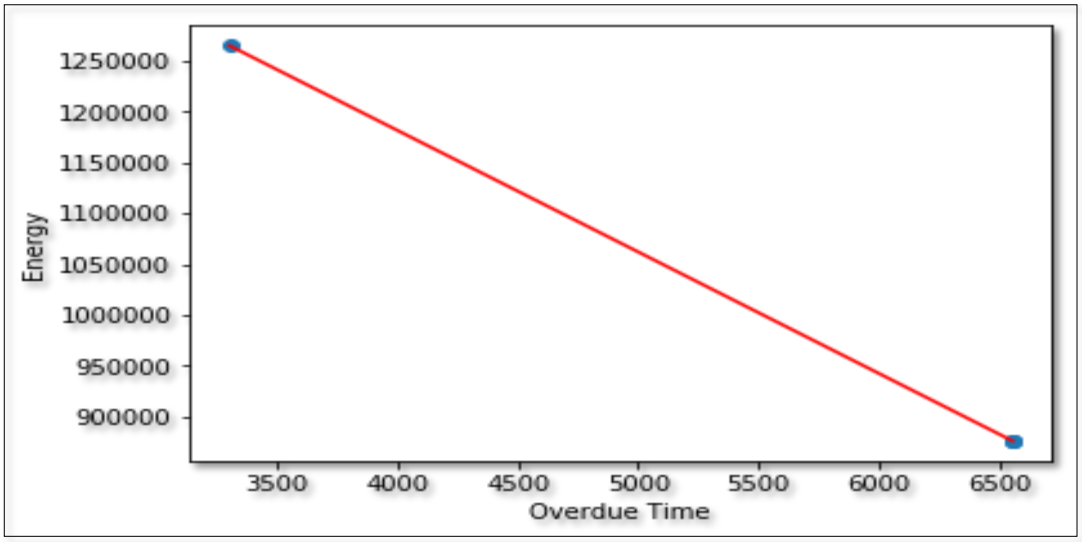


Figure 6.9: Pareto Front Endpoints between Energy and Overdue Time in Large Applications Scenario

6.7.2 Energy and Deadline Aware Scheduling Policy

Similar to the EA algorithm, EADA is assessed in different experimental scenarios. EADA is evaluated with multiple clusters of applications: mixed, small, medium and large applications and compared with FIFO, RR and SJF algorithms in every scenario.

6.7.2.1 Experiment 1: Mixed Applications

In terms of the average waiting time factor, the SJF policy performs well to reduce the minimum average waiting time by 2.44 seconds compared to others, while the proposed EADA policy has the maximum value of the average waiting time of 154.62 seconds, as shown in Figure 6.10(A).

It can be observed that the FIFO policy is performed better than other algorithms to reduce the average time in the system by 319.25 seconds. In contrast, the proposed EADA policy is the worst value in this criterion with 401.7 seconds, as depicted in Figure 6.10(B).

The proposed EADA, FIFO and RR algorithms successfully meet the assigned deadline. However, the SJF policy is the only policy that has an overdue time of 260.08 seconds, as shown in Figure 6.10(C).

EADA performs well to reduce energy consumption. EADA consumes 17% less energy than the FIFO policy, which produces the second smallest energy value among other algorithms since all applications are executed in the VM connected with GPU_K. Figure 6.10(D) shows the total energy consumed by each algorithm in the mixed applications group for EADA evaluation.

Moreover, both VMs share the task of applications execution in all algorithms with different utilisation percentages except the proposed EADA policy. In EADA policy, the VM connected with GPU_K is merely responsible to execute all applications, as shown in Figure 6.10(E). Figure 6.10 shows the results of all algorithms in the mixed applications scenario for EADA evaluation.

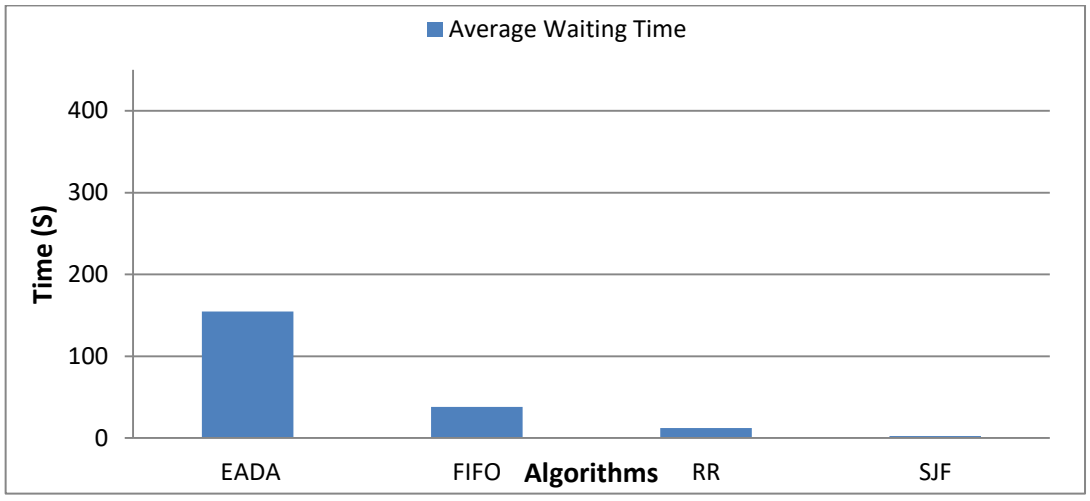


Figure 6.10(A): Average Waiting Time (Mixed Applications-EADA)

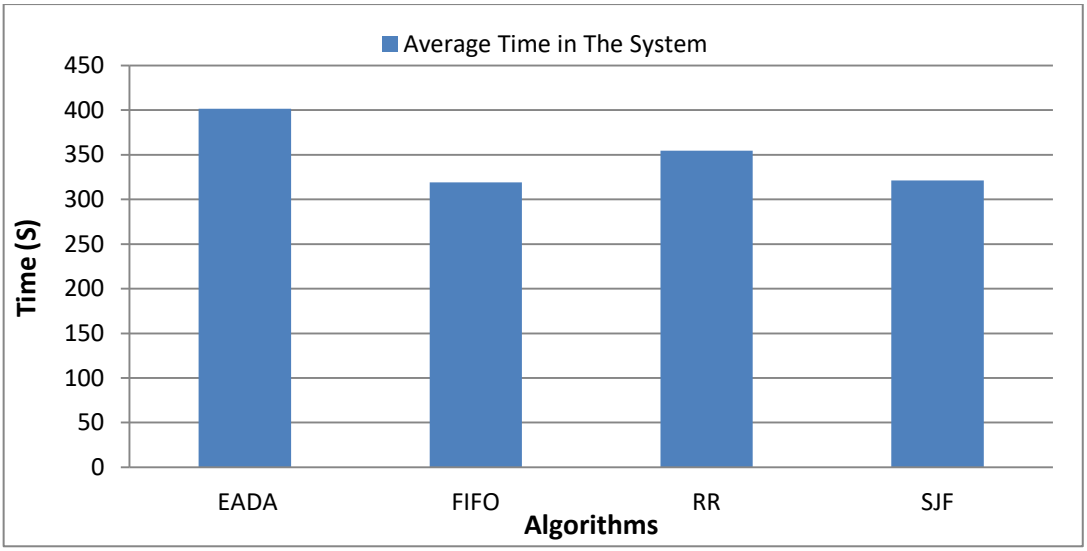


Figure 6.10(B): Average Time in the System (Mixed Applications-EADA)

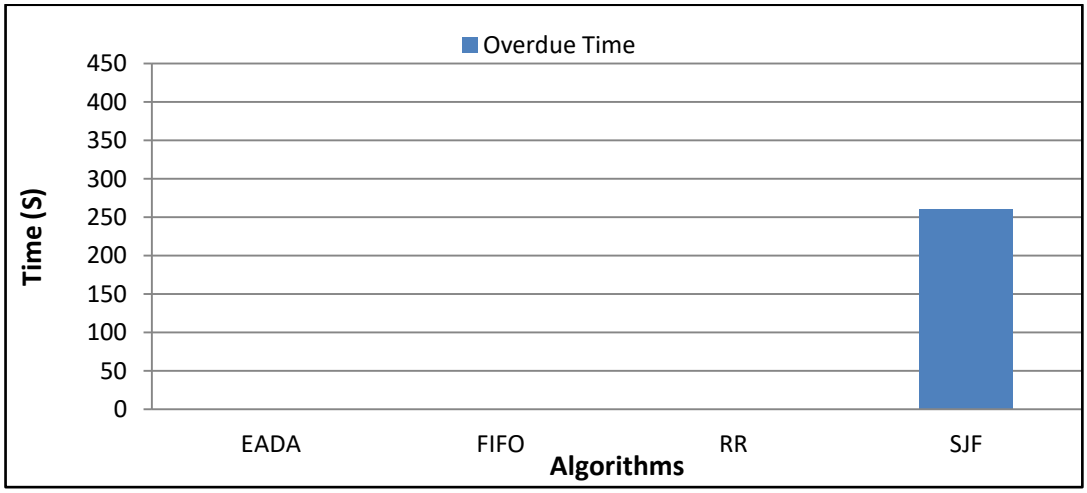


Figure 6.10(C): Overdue Time (Mixed Applications-EADA)

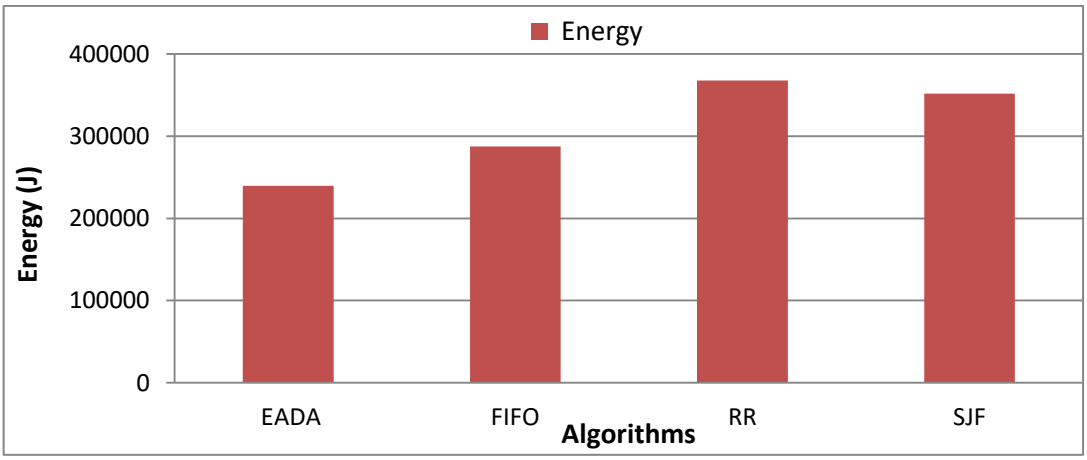


Figure 6.10(D): Energy Consumption (Mixed Application-EADA)

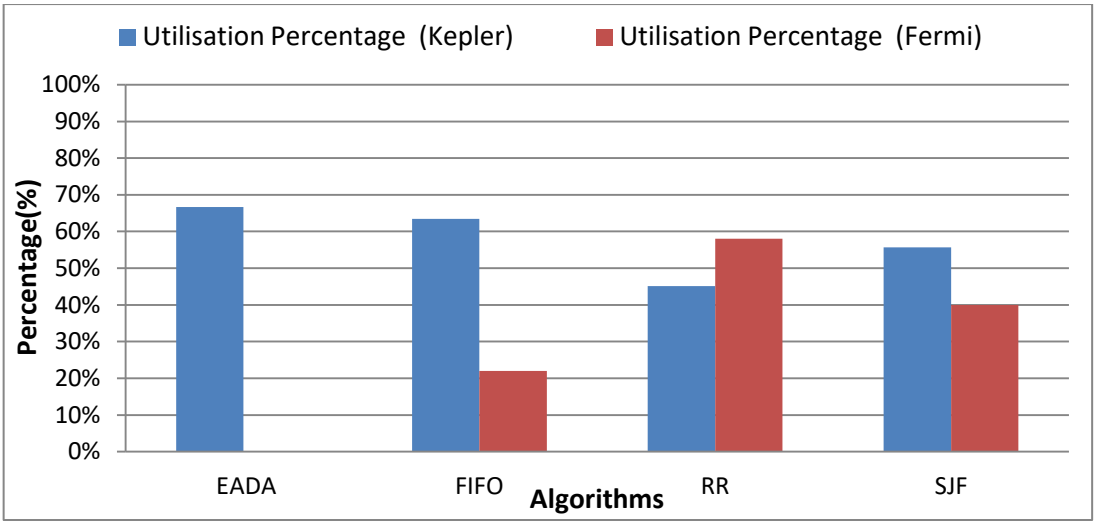


Figure 6.10 (E): VMs Utilisation (Mixed Applications-EADA)

Figure 6.10: The EADA Algorithm Results Evaluated on Mixed Applications Cluster

6.7.2.2 Experiment 2: Small Applications

In this scenario, FIFO and SJF algorithms perform well to eliminate the average waiting time. In opposite to FIFO and SJF algorithms, RR performs inadequately to reduce the average waiting time by 4.39 seconds, as depicted in Figure 6.11(A).

The proposed EADA algorithm performs well to reduce the average time in the system by 74.1 seconds compared to the rest algorithms. In contrast, the RR algorithm is the worst algorithm for reducing this criterion with 110.72 seconds since the waiting time of this algorithm is the highest, as shown in Figure 6.11(B).

The proposed EADA policy performs well to meet the assigned deadline and eliminate any overdue time. However, all other algorithms exceed the assigned deadline. RR policy is the highest algorithm that has an overdue time of 142.66 seconds, as shown in Figure 6.11(C).

Additionally, the EADA algorithm performs well to conserve energy consumption. EADA consumes 44% less energy than the FIFO algorithm, which produces the second smallest energy value among other algorithms since all applications are run in the VM connected with GPU_K. Figure 6.11(D) shows the total energy values for every algorithm for EADA evaluation.

Both VMs share the task to execute applications in all algorithms with different utilisation percentages except the proposed EADA. In EADA, the VM connected

with GPU_K is only accountable to execute all applications, as shown in Figure 6.11(E).

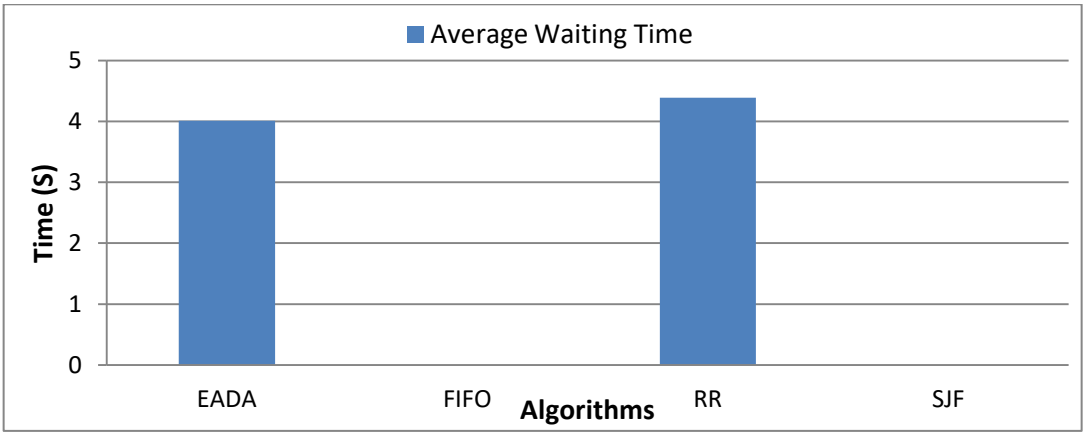


Figure 6.11 (A): Average Waiting Time (Small Application-EADA)

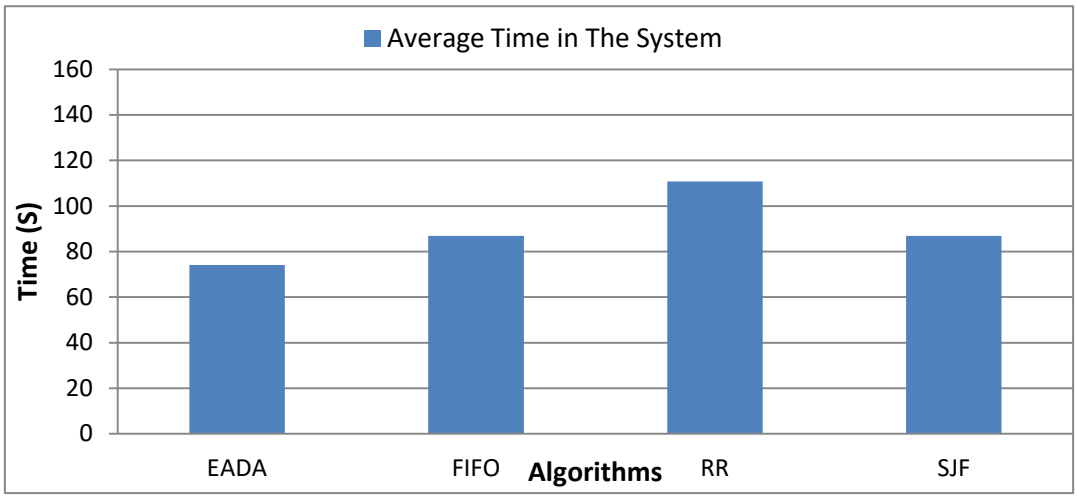


Figure 6.11(B): Average Time in the System (Small Applications-EADA)

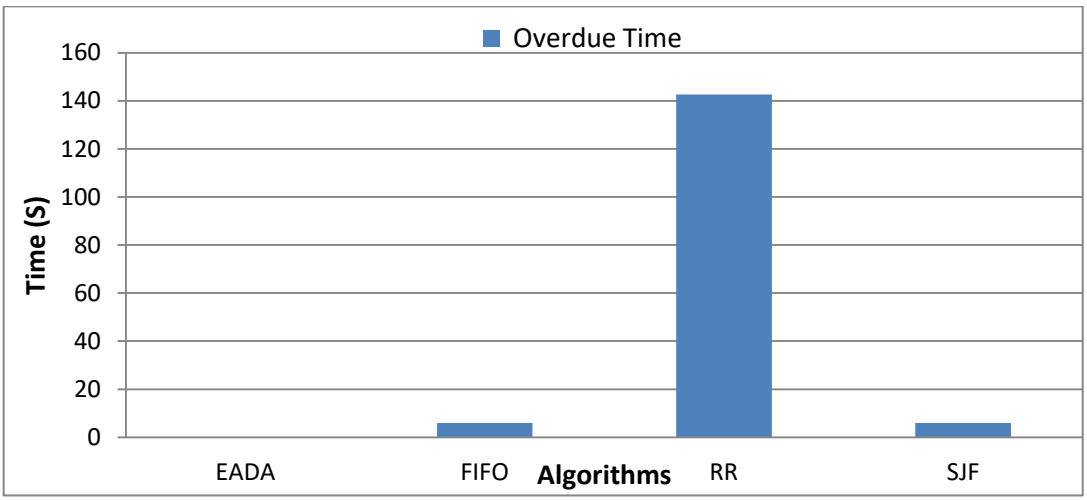


Figure 6.11 (C): Overdue Time (Small Applications-EADA)

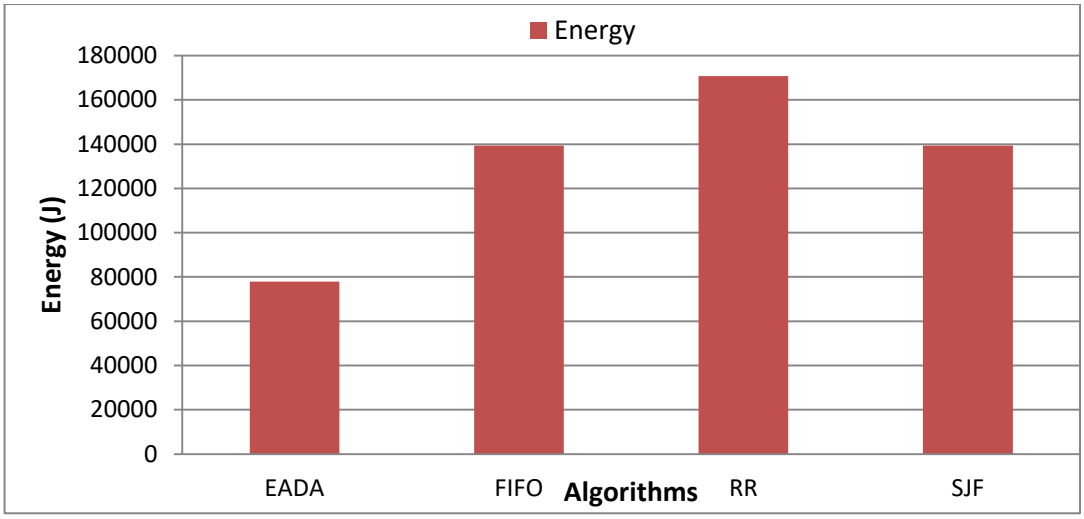


Figure 6.11 (D): Energy Consumption (Small Applications-EADA)

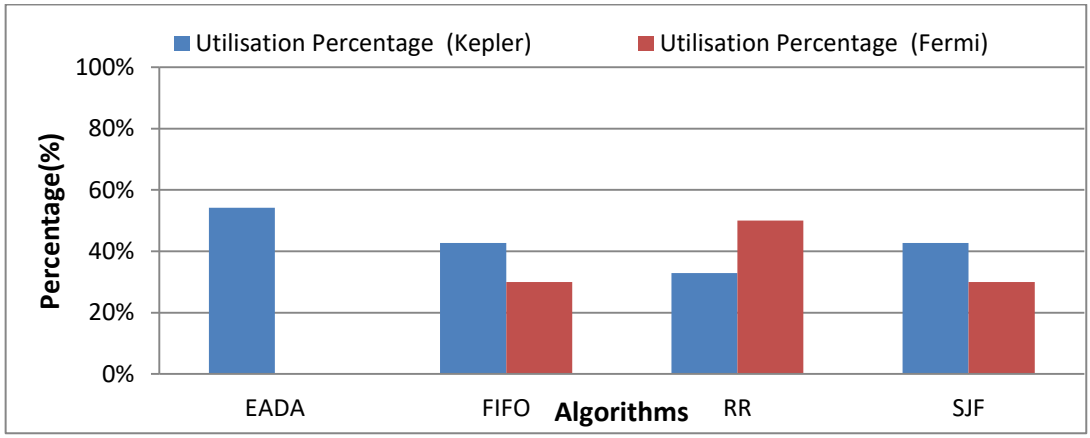


Figure 6.11 (E): VMs Utilisations (Small Applications-EADA)

Figure 6.11: The EADA Algorithm Results Evaluated on Small Applications Cluster

6.7.2.3 Experiment 3: Medium Applications

In terms of the average waiting time, SJF and FIFO algorithms are the minimum values of the average waiting time of 38.21 seconds compared to the rest algorithms, while the proposed EADA is the maximum value of the average waiting time of 142.28 seconds, as shown in Figure 6.12(A).

FIFO and SJF algorithms are the minimum values of the average time in the system with a similar value of 411.15 seconds compared with the rest algorithms. However, the proposed EADA algorithm is the worst for reducing this criterion with 488.25 seconds since its waiting time is the highest among other algorithms, as shown in Figure 6.12(B).

All the evaluated algorithms produce an overdue time. However, the proposed EADA algorithm is the minimum value to reduce the overdue time with 6% less than FIFO and SJF algorithms with 214.65 seconds. In contrast, the RR algorithm is the highest overdue time that missed the assigned deadline with 552.3 seconds, as depicted in Figure 6.12(C).

Moreover, the EADA algorithm performs well in terms of conserving energy consumption. EADA consumes 9% less energy than FIFO and SJF algorithms, which exhibit the second smallest energy value. The reason for this saving in the EADA algorithm is that the VM connected with GPU_K executes the allocated applications, which is more energy-efficient VM than another VM. Figure 6.12(D) illustrates the total energy consumed by each algorithm in the medium applications group for EADA evaluation.

Moreover, both VMs share the task of applications execution in all evaluated algorithms with different utilisation percentages. However, in the EADA algorithm, the VM connected with Kepler GPU does not have any idle time; therefore, it has a full-time utilisation (100%). Figure 6.12(E) shows VMs utilisation for each algorithm in the medium applications group for EADA evaluation.

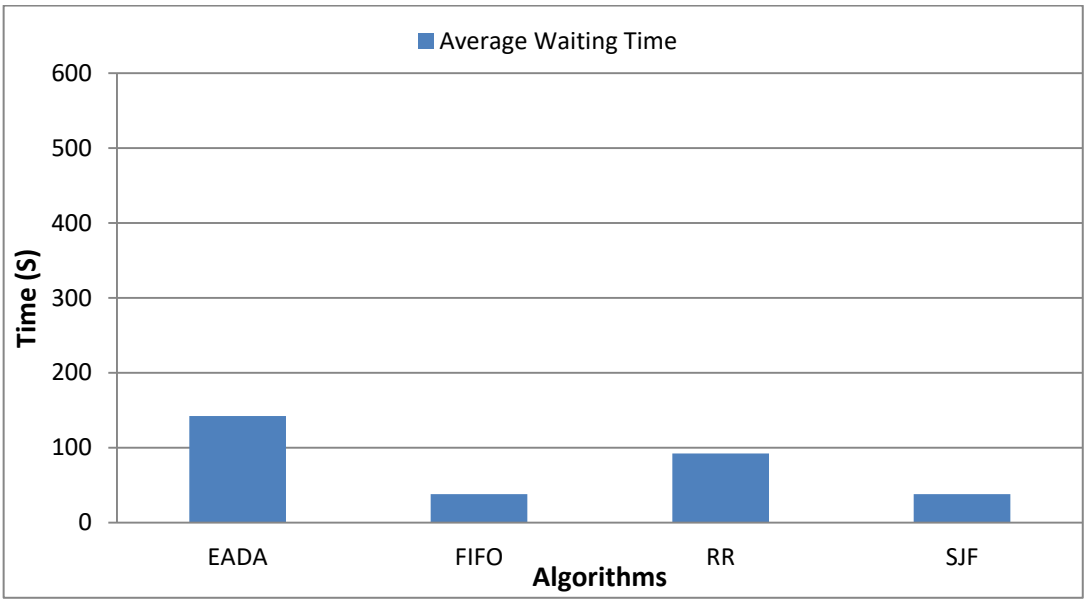


Figure 6.12(A): Average Waiting Time (Medium Applications-EADA)

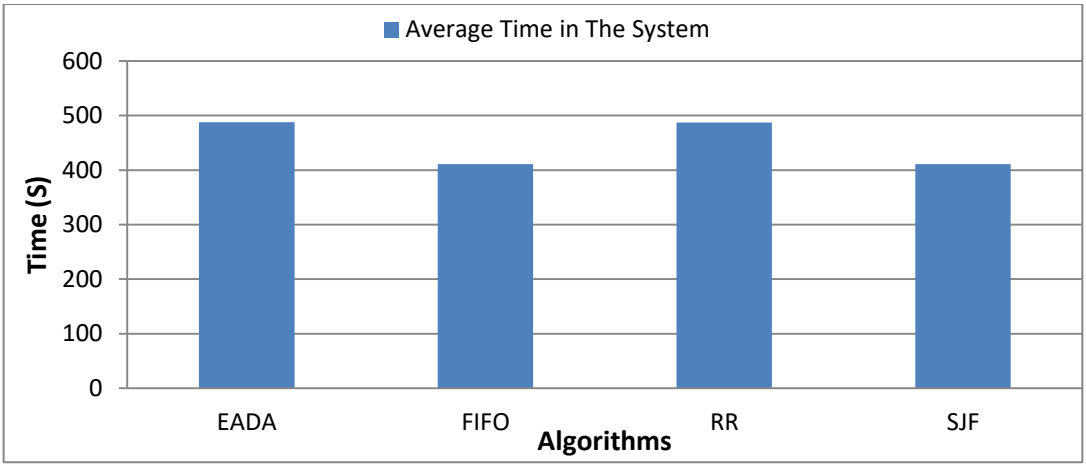


Figure 6.12(B): Average Time in the System (Medium Applications-EADA)

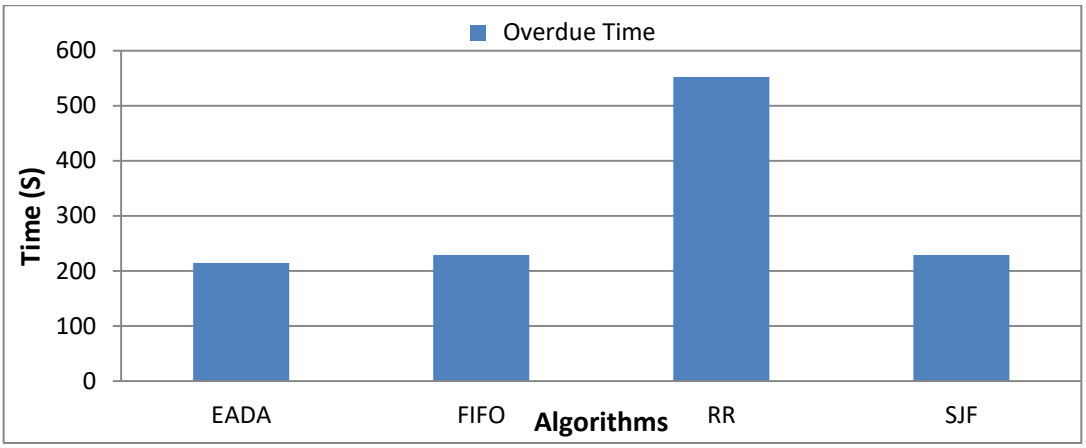


Figure 6.12(C): Overdue Time (Medium Applications-EADA)

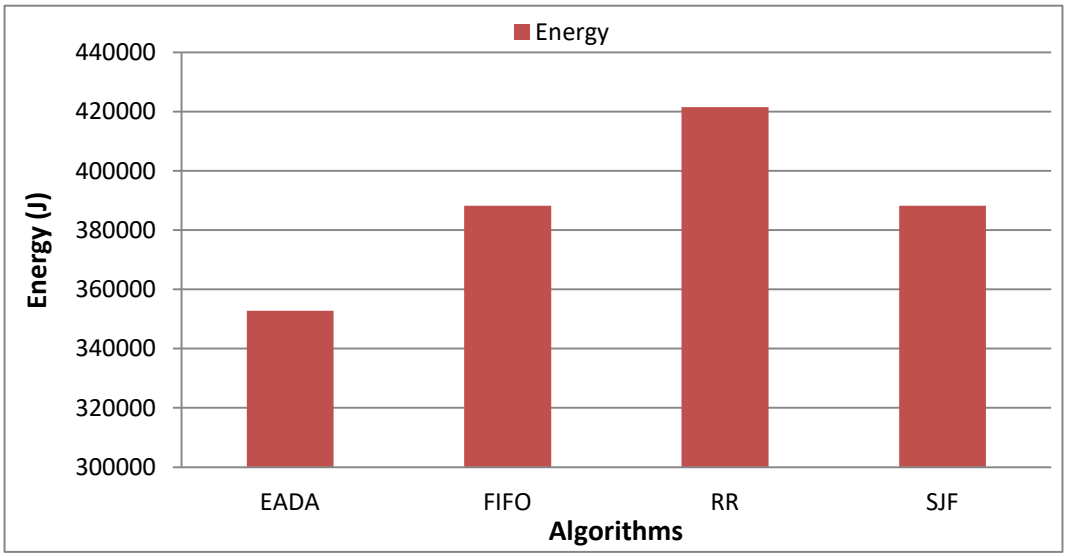


Figure 6.12(D): Energy Consumption (Medium Applications-EADA)

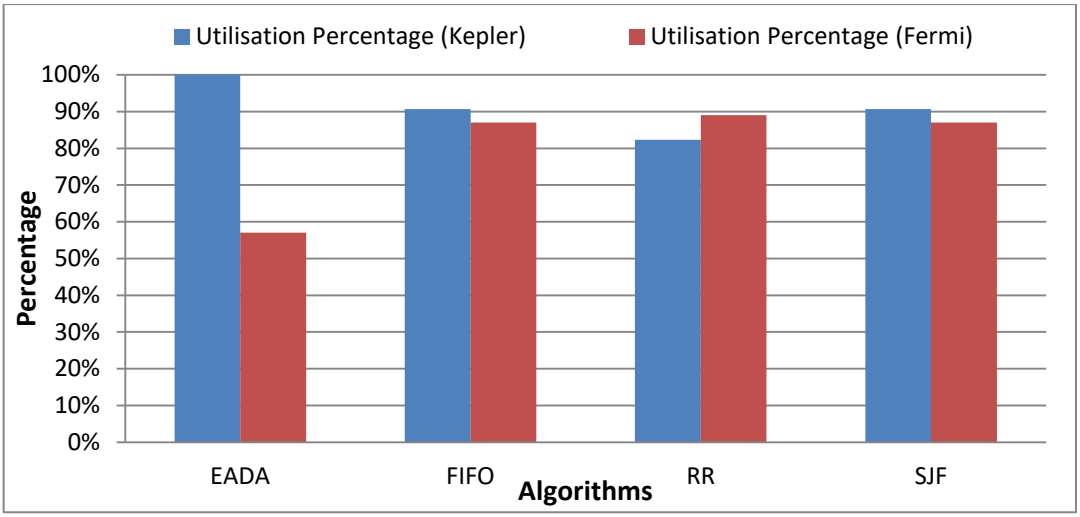


Figure 6.12(E): VMs Utilisations (Medium Applications-EADA)

Figure 6.12: The EADA Algorithm Results Evaluated on Medium Applications Cluster

6.7.2.4 Experiment 4: Large Applications

SJF algorithm performs well to reduce the average waiting time compared to others with 450.29 seconds. In opposite to SJF, the RR algorithm is the worst among others for reducing the average waiting time by 965.73 seconds, as shown in Figure 6.13(A).

EADA and SJF algorithms are the minimum value of the average time in the system with 1247.53 seconds compared with the other algorithms. However, the RR algorithm is the worst algorithm for reducing this criterion with 2180.12 seconds since its waiting time value is the highest, as depicted in Figure 6.13(B).

Furthermore, all evaluated algorithms produce an overdue time passing the assigned deadline. However, the proposed EADA algorithm is the minimum value to reduce the overdue time with 193.2 seconds, which is 58.4% lower than the SJF algorithm. In contrast, RR is the highest overdue time with 10030.08 seconds, as depicted in Figure 6.13(C).

Finally, EADA and SJF algorithms perform well in terms of conserving energy consumption. EADA and SJF consume 28 % less energy than the FIFO algorithm, which produces the second smallest energy value. The reason for this saving is that the VM connected with GPU_K executes all allocated applications, which is an energy-efficient VM. In contrast, RR is the worst algorithm for conserving energy

consumption. Figure 6.13(D) illustrates the total energy consumed for each algorithm in the large applications group for EADA evaluation.

In this scenario, both VMs share the task of executing applications in all algorithms with different utilisation percentages. The VM Figure 6.13(E) shows VMs utilisation for each algorithm in large applications group EADA evaluation.

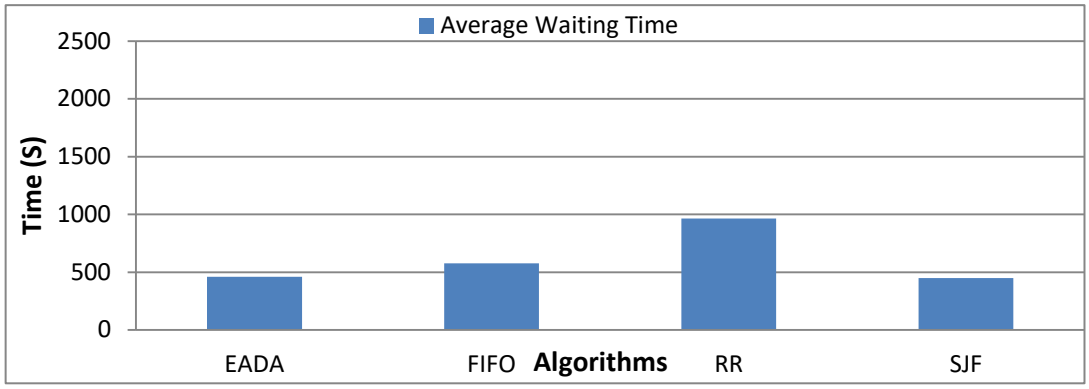


Figure 6.13(A): Average Waiting Time (Large applications-EADA)

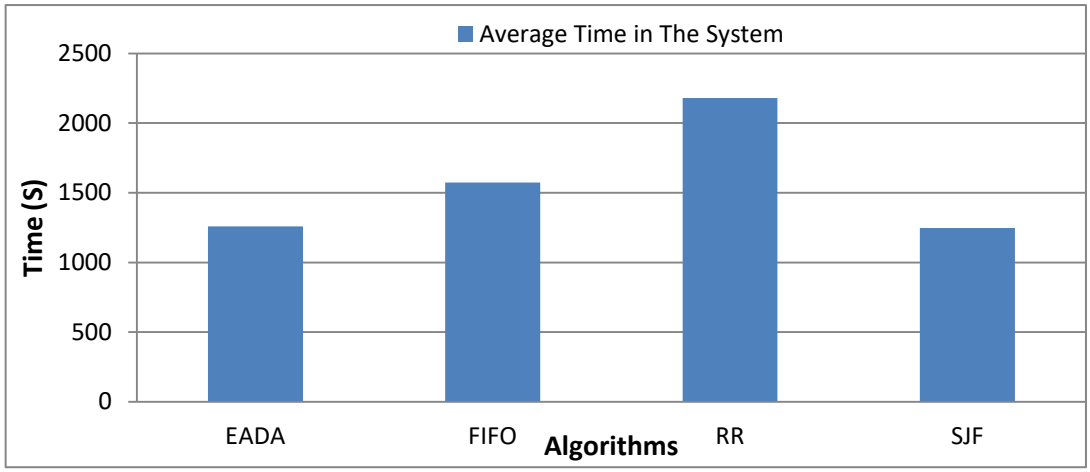


Figure6.13 (B): Average Time in the System (Large Applications-EADA)

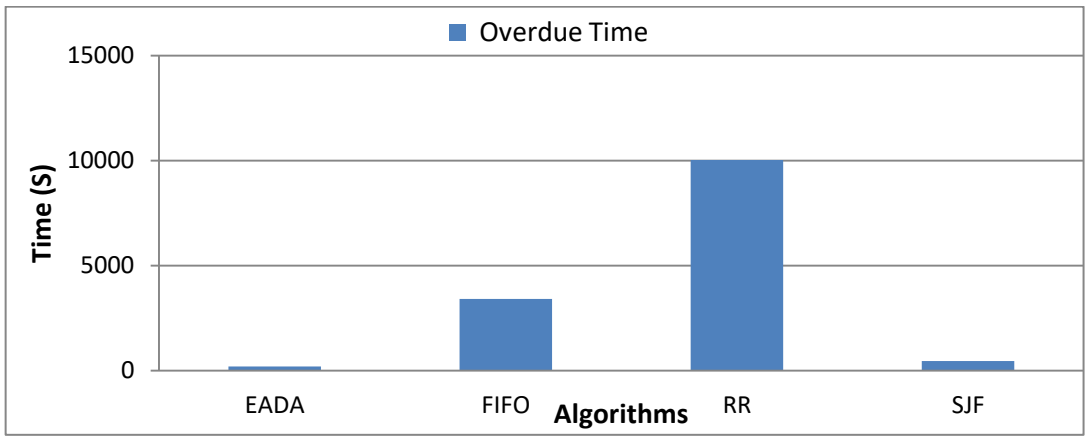


Figure 6.13 (C): Overdue Time (Large Applications-EADA)

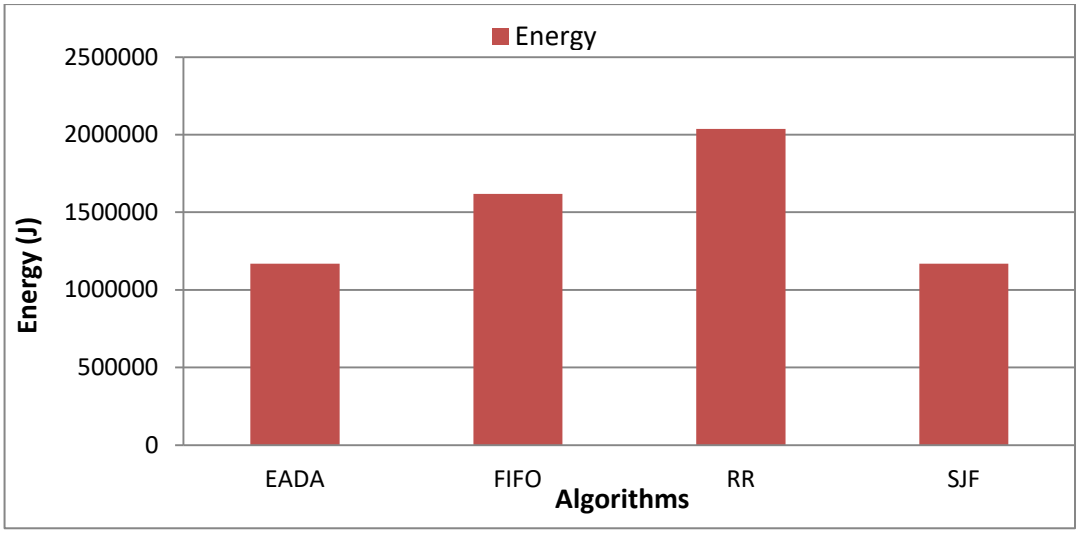


Figure 6.13 (D): Energy Consumption (Large-Applications-EADA)

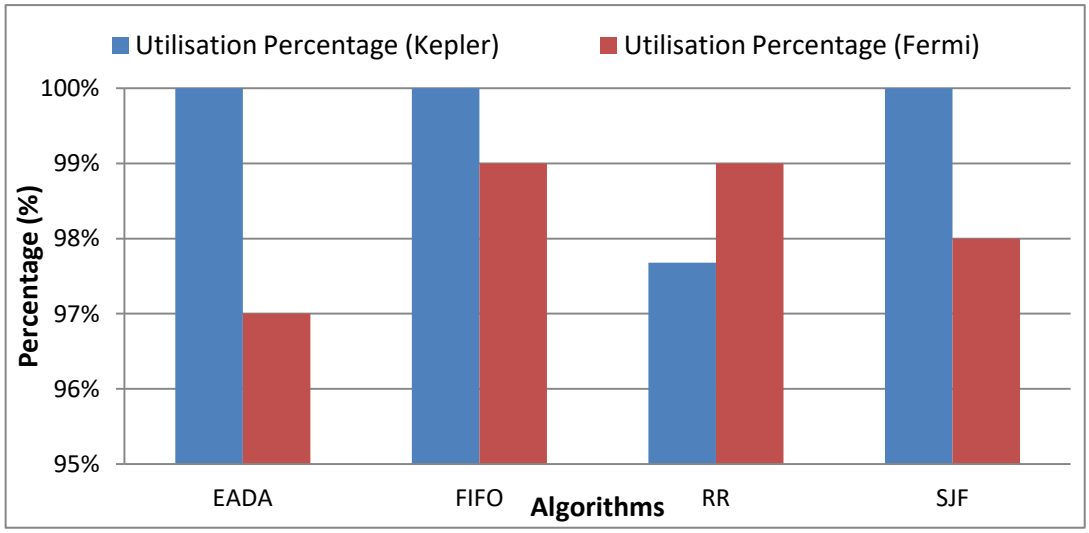


Figure 6.13 (E): VMs Utilisation (Large Applications-EADA)

Figure 6.13: The EADA Algorithm Results Evaluated on Large Applications Cluster

6.7.3 Energy-Aware and Energy and Deadline Aware Algorithms Comparison

This subsection aims to compare the EA thresholds percentages between EA-T30% and EA-T60% with EADA in terms of average waiting time, the average time in the system, overdue time, VMs utilisation and total energy. Another aim is to evaluate the trade-off among the assigned criteria (Average Waiting Time, Average Time in the System, Overdue Time, VMs utilisation and Energy Consumption). Similar to previous evaluations, the comparison is performed in four main scenarios. These scenarios are mixed, small, medium, and large applications.

6.7.3.1 Experiment 1: Mixed Applications

EA-T30%, EA-T40% and EA-T50% have an identical behaviour with EADA in all criteria in the mixed applications cluster scenario, as shown in Figures 6.14, 6.15, 6.16, 6.17 and 6.18. In contrast, EA-T60% is the worst behaviour in all criteria among other algorithms except in the average waiting time criterion. It is the best value to reduce the average waiting time compared to the rest algorithms with a reduction of 21.2%, as shown in Figure 6.14. In EA-T60%, both VMs share the task of applications execution. However, in other EA threshold values and EADA, the VM with GPU_K is merely responsible for applications execution, as shown in Figure 6.18.

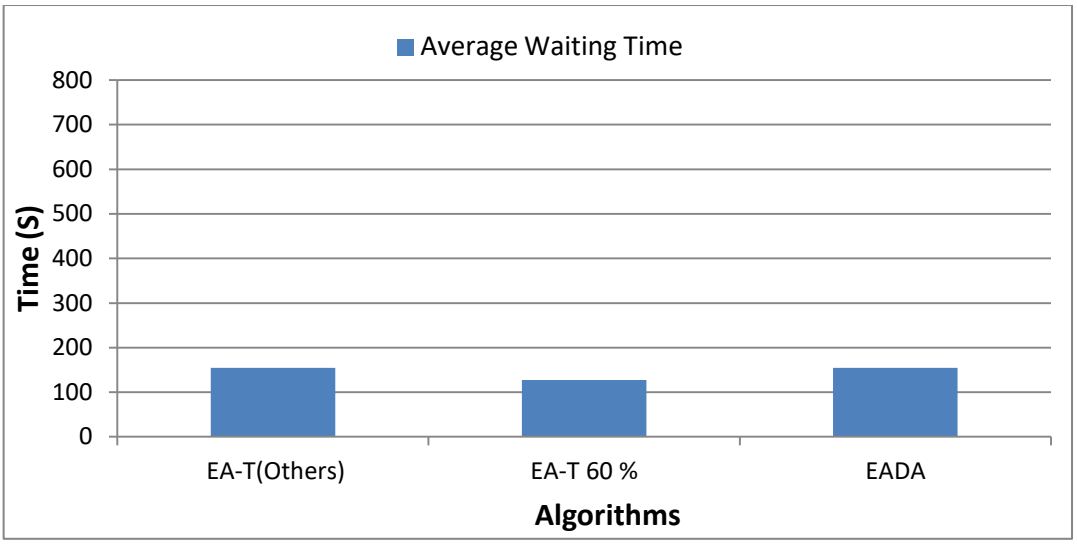


Figure 6.14: Average Waiting Time between EA and EADA in Mixed Applications

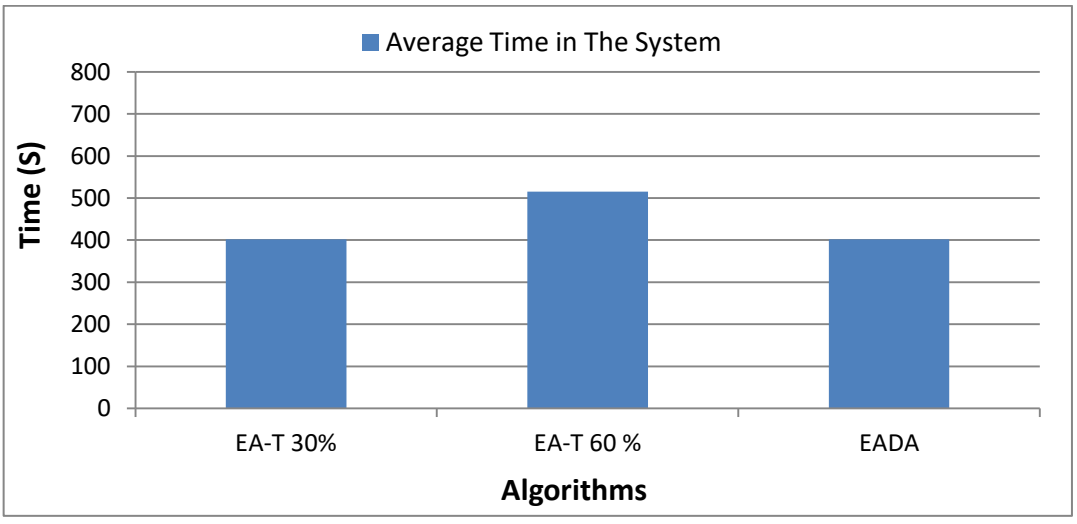


Figure 6.15: Average Time in the System between EA and EADA in Mixed Applications

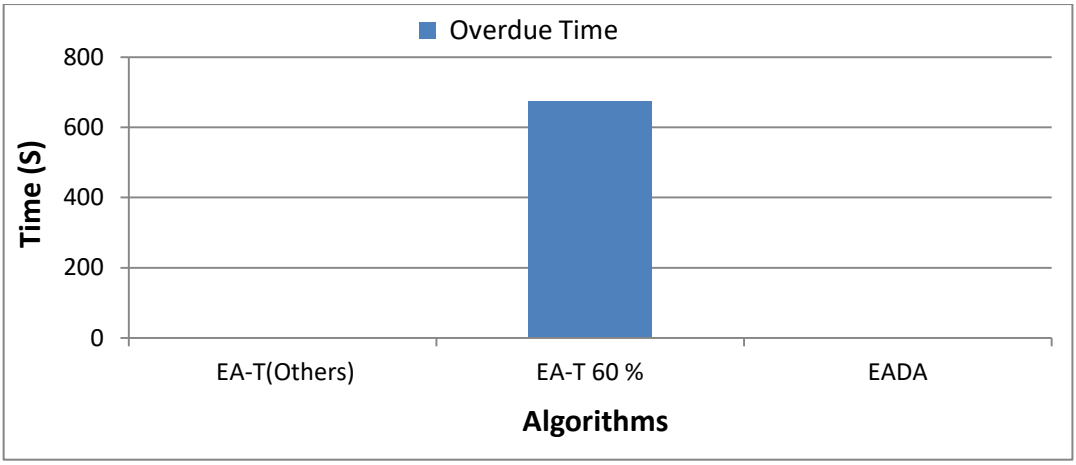


Figure 6.16: The Deadline Overdue Time between EA and EADA in Mixed Applications

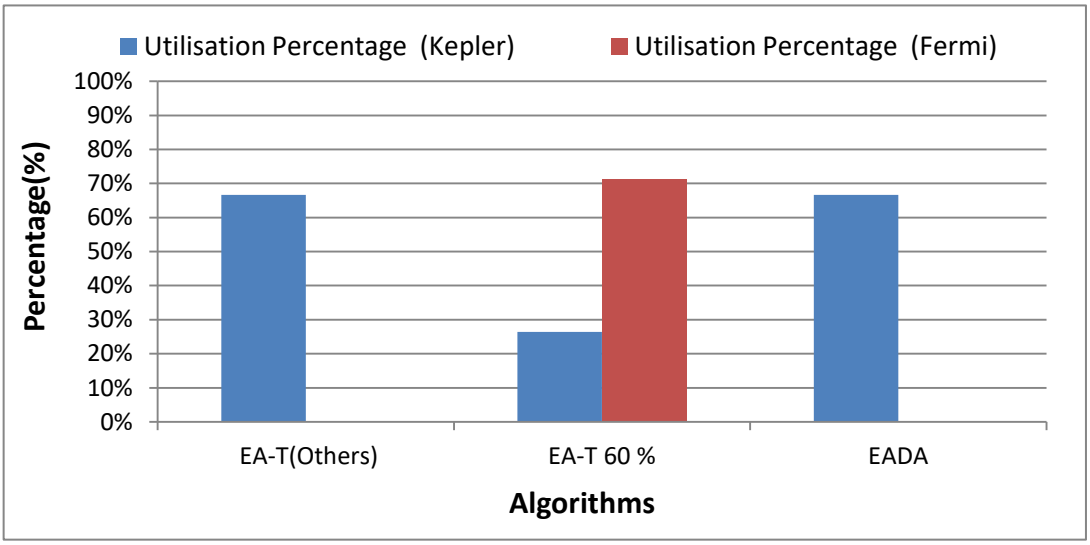


Figure 6.17: VMs Utilisation between EA and EADA in Mixed Applications

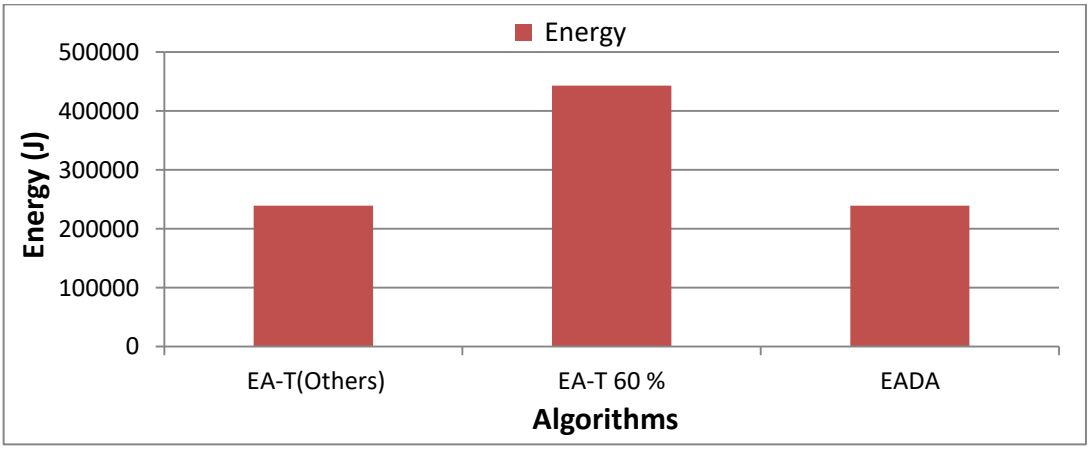


Figure 6.18: The Total Energy between EA and EADA in Mixed Applications

6.7.3.2 Experiment 2: Small Applications

In this scenario, all proposed EA algorithms with threshold values between 30% and 60% and also EADA have identical behaviour in all criteria since the system is not under stress. All algorithms do not produce an overdue time, and also the VM with GPU_K merely executes all applications in all algorithms.

6.7.3.3 Experiment 3: Medium Applications

EADA algorithm performs well to reduce the average waiting time compared to others with 142.28 seconds. EADA is 43% less than EA-T60%, which exhibits the second-best value. In the opposite of EADA, other EA thresholds are the worst among other algorithms for reducing the average waiting time by 404.75 seconds, as shown in Figure 6.19.

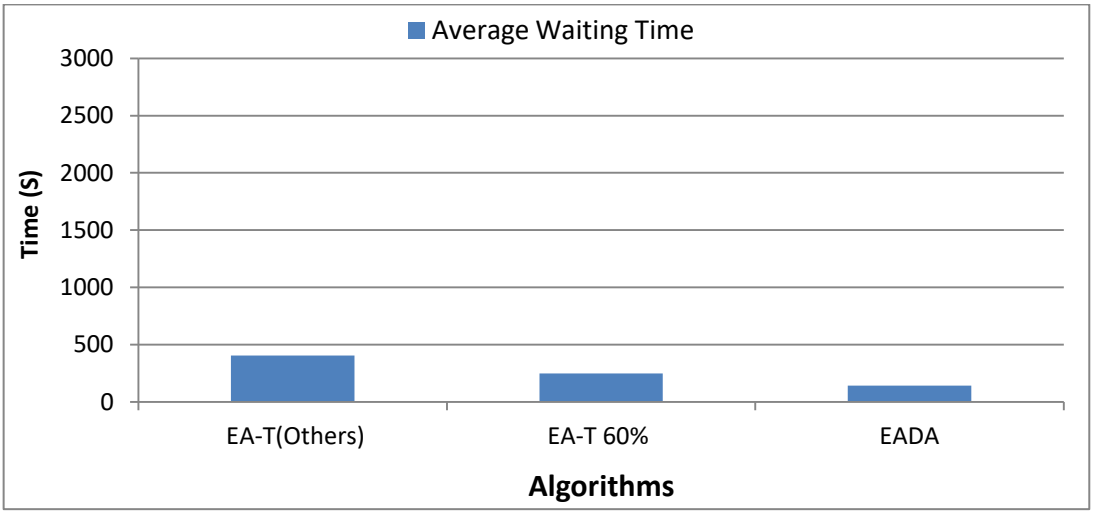


Figure 6.19: Average Waiting Time between EA and EADA in Medium Applications

EADA has the minimum value of the average time in the system with 488.25 seconds compared with the rest algorithms. EADA is 26% less than EA-T60%. However, other EA threshold values are the worst for reducing this criterion with 685.24 seconds since their waiting time is the highest, as shown in Figure 6.20.

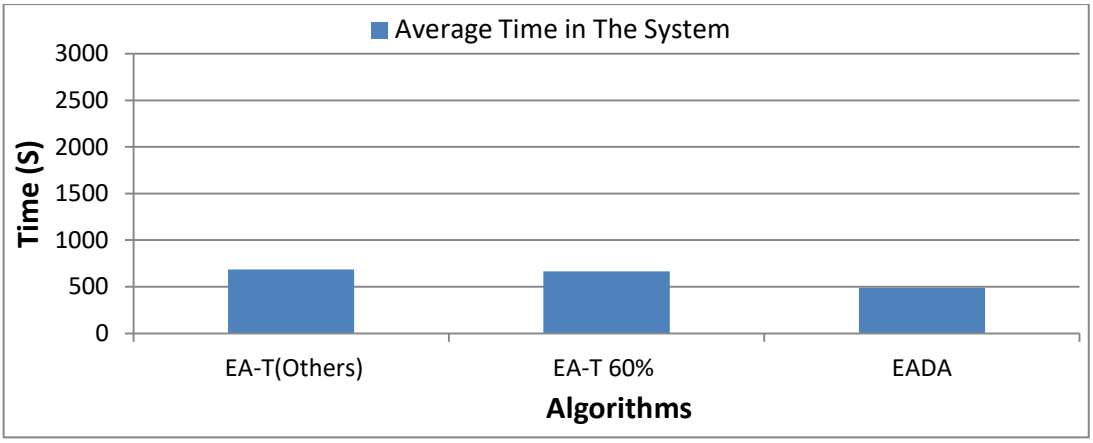


Figure 6.20: Average Time in the System between EA and EADA in Medium Applications

All algorithms produce an overdue time. However, the proposed EADA algorithm has the minimum value to reduce the overdue time with 88% less than EA-T30%, EA-T40% and EA-T50%. In contrast, EA-T60% has the highest overdue time with 2549.94 seconds, as depicted in Figure 6.21.

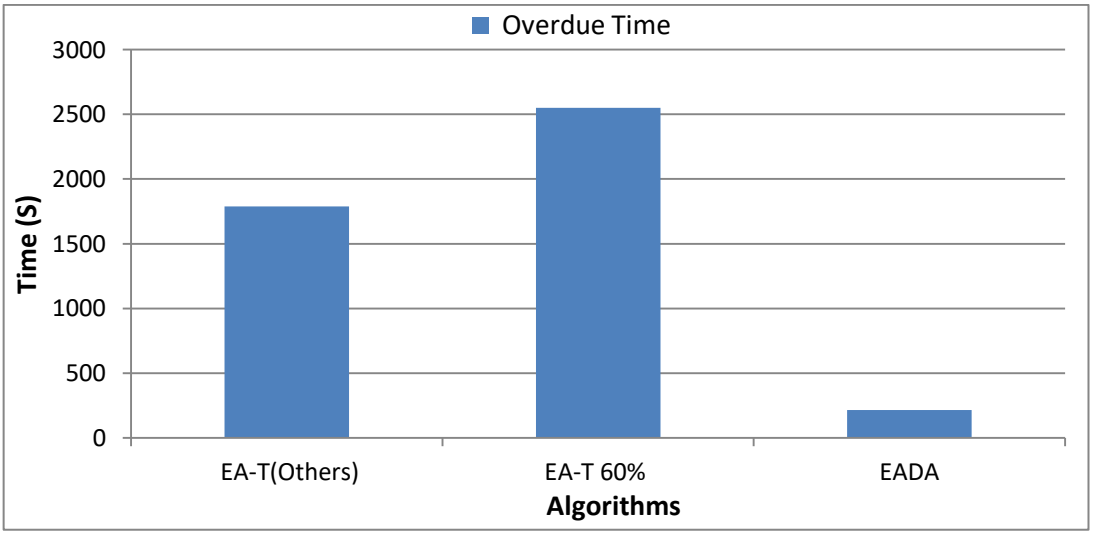


Figure 6.21: The Deadline Overdue Time between EA and EADA in Medium Applications

Both VMs share the task to execute applications in all the evaluated algorithms with different utilisation percentages except EA-T30%, EA-T40% and EA-T50%. In EA-T30%, EA-T40% and EA-T50%, the VM connected with GPU_K is merely responsible to execute all allocated applications with 100% utilisation, which indicates that the VM does not have any idle time, as shown in Figure 6.22.

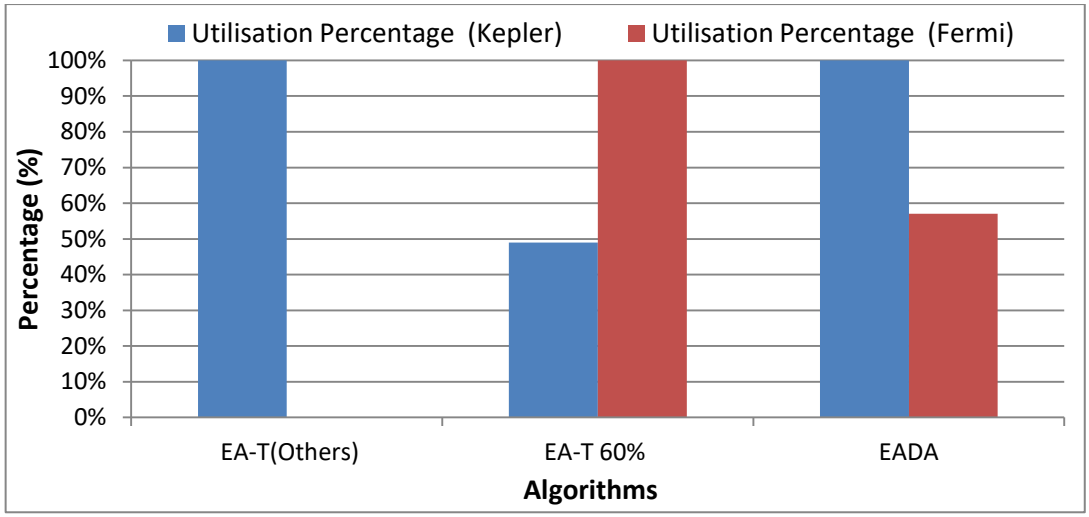


Figure 6.22: VMs Utilisation between EA and EADA in Medium Applications Scenario

Moreover, EA-T30%, EA-T40% and EA-T50% perform well in terms of conserving energy consumption. They consume 28% less energy than the EADA algorithm, which exhibits the second smallest energy value. The reason for this saving is that all applications are run in the VM connected with GPU_K, which is an energy-efficient VM, as shown in Figure 6.23.

In this scenario, the trade-off between energy consumption and performance whilst maintaining QoS requirements in EA-T30%, EA-T40% and EA-T50% is significant. However, the EADA algorithm successfully achieves to balance the trade-off between energy consumption and performance whilst maintaining the QoS requirements. The EADA algorithm simultaneously reduces energy consumption and the overdue time compared to the EA algorithm.

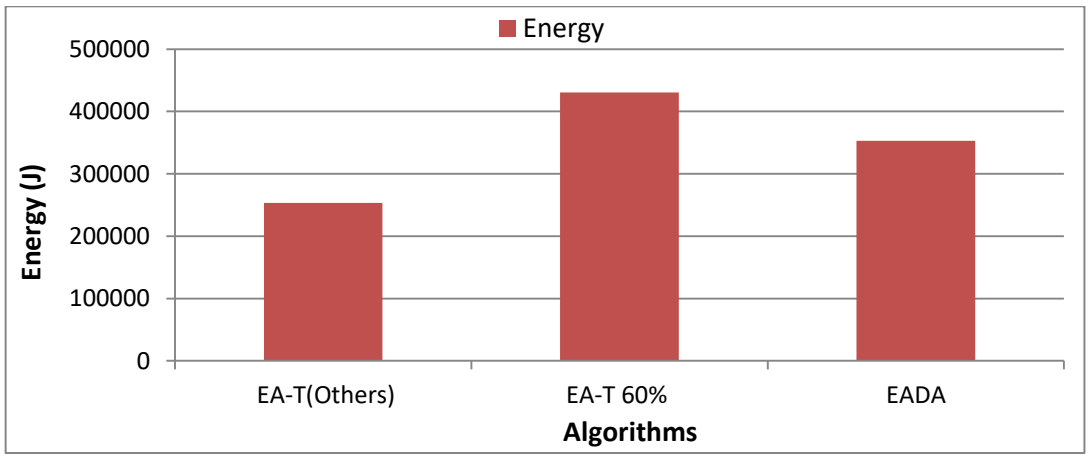


Figure 6.23: The Total Energy between EA and EADA in Medium Applications

6.7.3.4 Experiment 4: Large Applications

EADA performs well to reduce the average waiting time among others with 460.19 seconds. EADA is 24% less than the EA-T60% algorithm, which is the second-best value. In opposite to EADA, EA-T30%, EA-T40% and EA-T50% are the worst among others for reducing the average waiting time by 1580.97 seconds, as shown in Figure 6.24.

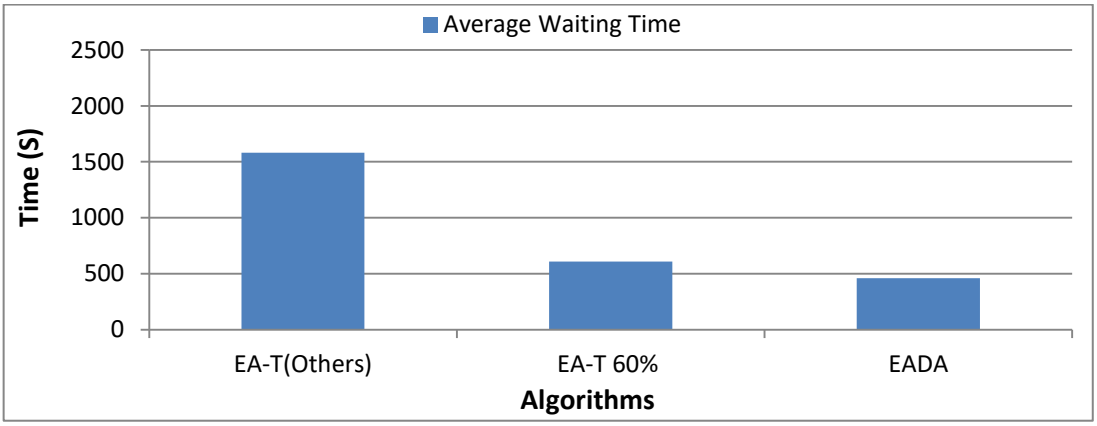


Figure 6.24: Average Waiting Time between EA and EADA in Large Applications

EADA is the minimum value of the average time in the system with 1257.43 seconds compared with the rest algorithms. EADA is 14% less than EA-T60%. However, other proposed EA threshold values are the worst for reducing this criterion with 2199.16 seconds since their waiting time is the highest among others, as shown in Figure 6.25.

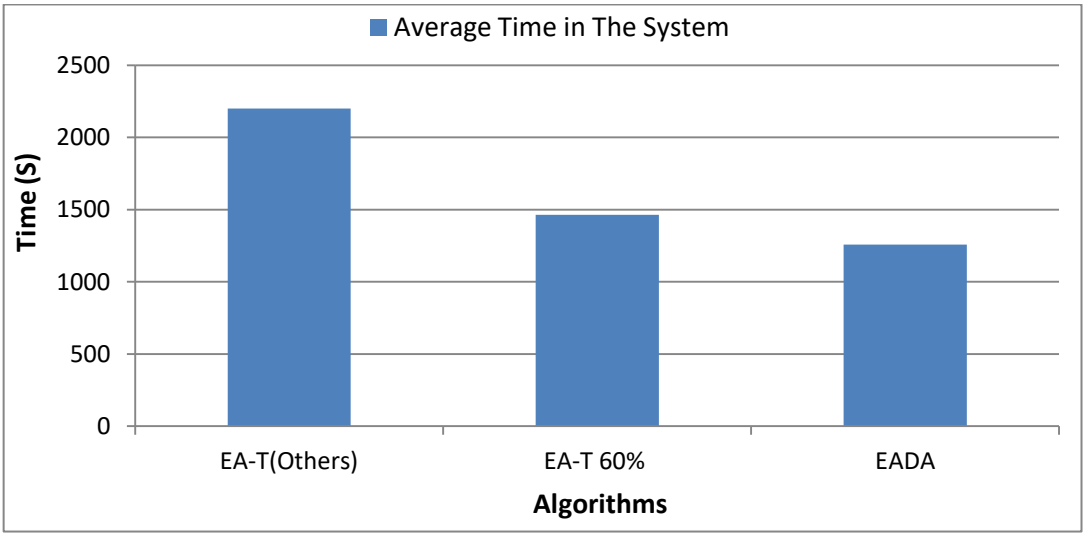


Figure 6.25: Average Time in the System between EA and EADA in Large Applications

All algorithms produce an overdue time and do not meet the assigned deadline in this scenario. However, the proposed EADA is the minimum value to reduce the overdue time by 94% less than EA-T60% with 193.2 seconds. In contrast, other EA threshold values are the highest overdue time by 6554.42 seconds, as depicted in Figure 6.26.

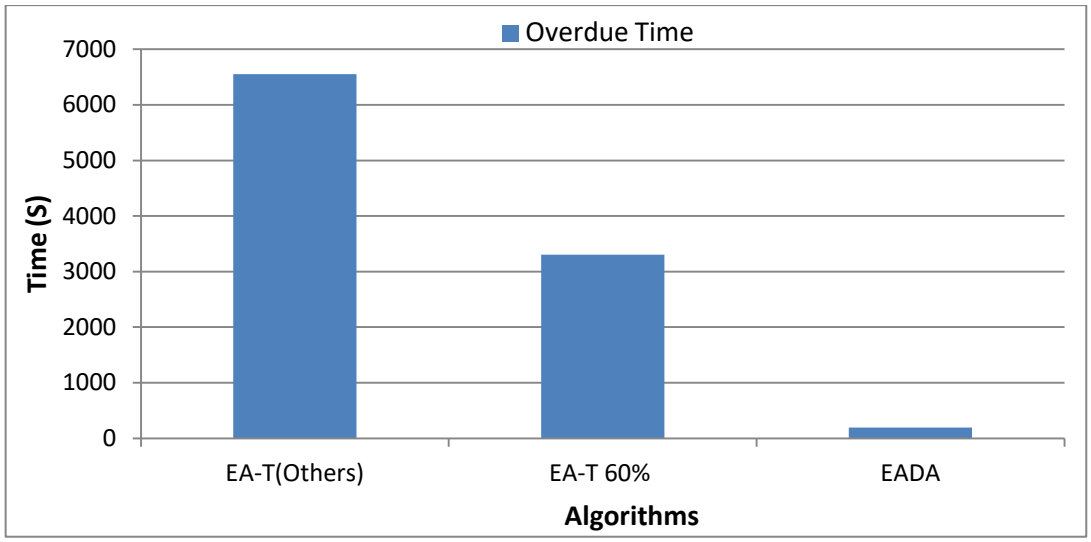


Figure 6.26: The Deadline Overdue Time between EA and EADA in Large Applications

Both VMs share the task to execute applications in all algorithms with different utilisation percentages except EA-T30, EA-T40% and EA-T50% EA threshold values. In EA-T30, EA-T40% and EA-T50%, the VM connected with GPU_K is merely responsible to execute all allocated applications with 100% utilisation, as shown in Figure 6.27.

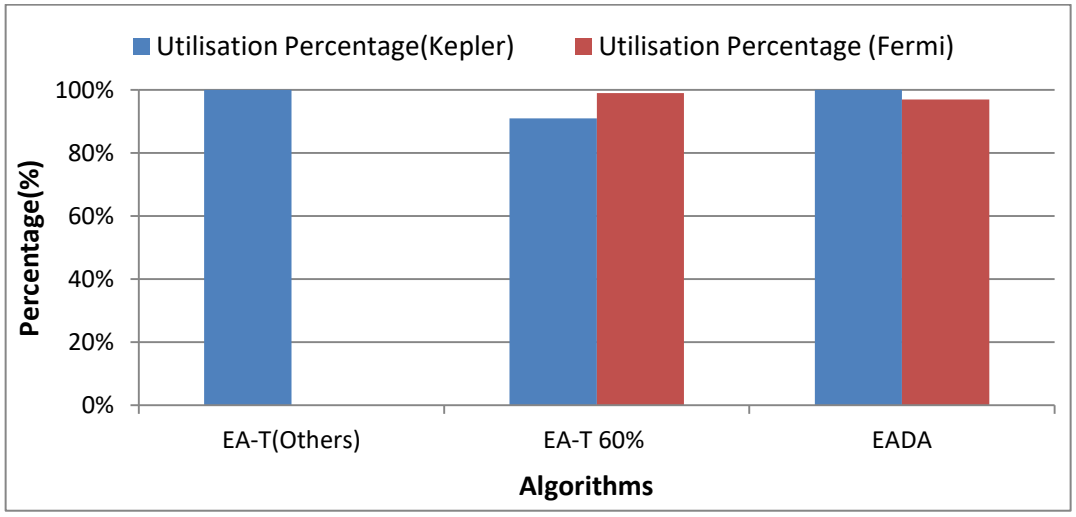


Figure 6.27: VMs Utilisation between EA and EADA in Large Applications

Finally, EA-T30%, EA-T40% and EA-T50% perform well in terms of energy consumption conservation. EA-T30%, EA-T40% and EA-T50% similarly consume 25% less energy than the EADA algorithm, which is the second smallest energy value. The reason for this saving is that all applications are run in the VM connected with GPU_K, which is more energy-efficient than another VM, as shown in Figure 6.28.

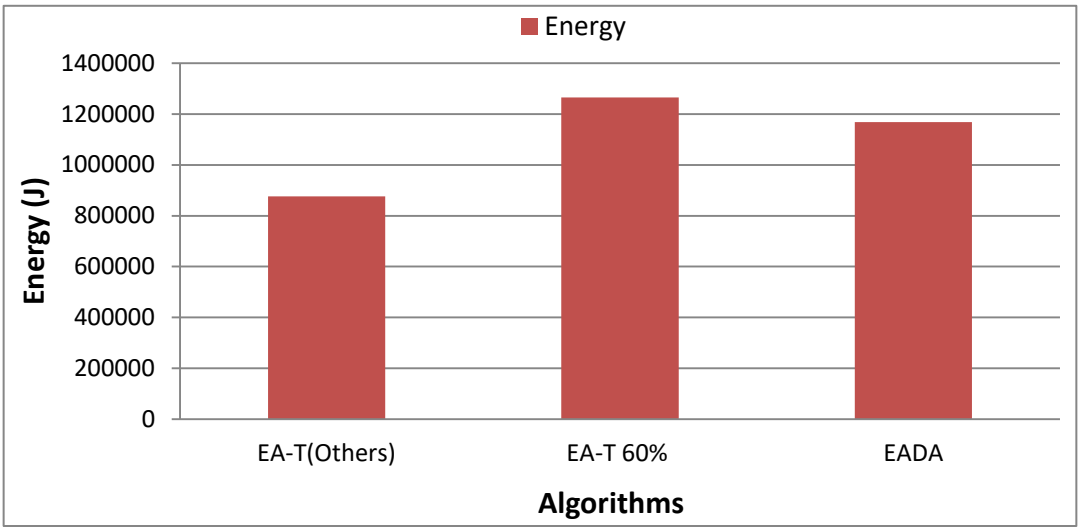


Figure 6.28: The Total Energy between EA and EADA in Large Applications

Similar to the medium applications scenario, the trade-off between energy consumption and performance whilst maintaining QoS requirements in EA-30%, EA-T40% and EA-T50% is significant. However, EADA successfully achieves balancing the trade-off between energy consumption and performance whilst maintaining the QoS requirements. The EADA algorithm simultaneously reduces energy consumption and the overdue time compared to the EA algorithm in the large applications group scenario.

6.8 Discussion

The Proposed EA and EADA algorithms have been developed and evaluated through several scenarios. These several scenarios have revealed a comprehensive behaviour in the proposed algorithms. In each scenario, these proposed algorithms (EA and EADA) have been compared with other algorithms (FIFO, RR and SJF). The experiments have revealed that each algorithm has had different characteristics when they have performed in each scenario. For example, in EA evaluation experiments, SJF has succeeded to reduce the average waiting time criterion in all

scenarios. Table 6.6 illustrates the comprehensive comparison among algorithms in EA evaluation, EA represents all threshold values.

Moreover, in all EA evaluation scenarios, the proposed EA algorithm has succeeded to meet the objective goal to conserve energy consumption which addressed the research question **Q.4**, as shown in Figures 6.5(D), 6.6(D), 6.7 (D) and 6.8(D). However, there is an unbalanced trade-off between conserving energy consumption and other performance criteria, especially in medium and large applications clusters experiments. For example, EA has been the worst algorithm to meet the deadline or even to reduce the overdue time in medium and large applications scenario. Using the EA algorithm with static thresholds to reduce energy consumption can be applied on small Cloud infrastructures, as we used in this research. However, applying this scheduling technique on a large scale Cloud computing infrastructure cannot be beneficial because the complexity and the overhead will increase. Therefore, a dynamic scheduling technique will be an appropriate option in large Cloud Computing infrastructures.

Therefore, EADA has aimed to create a balanced trade-off between energy and performance whilst maintaining the QoS requirements (the deadline), which addressed the research question **Q.5.** EADA has succeeded to reduce energy consumption and eliminate or reduce overdue time. Additionally, it has been observed that EADA has had the best value to reduce the overdue time compared to others in all scenarios with acceptable energy consumption. Table 6.7 illustrates a comprehensive comparison among algorithms in EADA evaluation.

EA with 30%, 40% and 50% threshold values and EADA have had similar behaviour in all criteria in mixed and small applications scenarios. EA with the 30%, 40% and 50% threshold values have had better energy conservation than others in the rest scenarios. However, this conservation has had great consequences on other criteria. It has created an unbalanced trade-off between energy consumption and other performance. Therefore, EADA has aimed to balance this trade-off, as seen in section 6.7.3. For example, EADA has performed well to reduce the overdue deadline time in medium and large applications scenarios and has consumed energy less than the EA with the 60% threshold policy.

Table 6.6: A Comprehensive Comparison among Algorithms in EA Evaluation

	Mixed				Small				Medium				Large			
	EA	FIFO	RR	SJF	EA	FIFO	RR	SJF	EA	FIFO	RR	SJF	EA	FIFO	RR	SJF
Average Waiting Time				X		X		X		X		X				X
Average Time in The System		X			X					X		X				X
Deadline Overdue Time	X	X	X		X					X		X				X
Total Energy	X				X				X				X			

Table 6.7: A Comprehensive Comparison among Algorithms in EADA Evaluation

	Mixed				Small				Medium				Large			
	EADA	FIFO	RR	SJF	EADA	FIFO	RR	SJF	EADA	FIFO	RR	SJF	EADA	FIFO	RR	SJF
Average Waiting Time				X		X		X		X		X				X
Average Time in The System		X			X					X		X				X
Deadline Overdue Time	X	X	X		X				X				X			
Total Energy	X				X				X				X			X

6.9 Summary

In this chapter, the proposed EA and EADA algorithms have developed and also have evaluated to allocate GPU applications to two VMs connected with heterogeneous GPUs in detail. Then, the algorithms have been evaluated comprehensively through several scenarios compared with other algorithms FIFO, RR and SJF. Another evaluation has been conducted to compare EA with EADA and evaluating the trade-off between energy and performance whilst maintaining the QoS.

The next chapter will aim to conclude the presented work in this thesis, the main contributions, limitations and the suggested future works.

Chapter 7 Conclusion

7.1 Research Summary

The work presented in this thesis has illustrated the ability to enable energy awareness for managing heterogeneous resources in Cloud computing infrastructures. In order to achieve this aim, a systematic architecture that consists of several components communicating with each other has been developed.

- **Chapter 2** presented the background material in relation to this research. This chapter introduced the Cloud computing concept, its service and deployment types. Virtualisation definition with its types, hypervisors types, Virtual Infrastructure Managers types and containers were covered. GPU architectures, GPU and CPU comparison, the general purpose of using GPUs including the supportive programming languages were also presented. Then, this chapter demonstrated power and energy consumption differences. Moreover, GPU virtualisation types and GPU usage in Cloud computing were presented. Finally, Machine Learning and Deep Learning were also covered in this chapter.
- **Chapter 3** presented the related work and techniques of energy awareness for GPU resource management in Cloud computing. This chapter began by presenting the conducted work on GPU power modelling and its techniques types in Cloud and non-Cloud computing environments. This chapter also provided a survey regarding the performed works for GPU energy modelling and the used techniques types. Moreover, the work related to GPU performance modelling, workload prediction and energy models based on workload prediction in Cloud computing were presented. Finally, the work of GPU scheduling types in Cloud computing was reviewed in detail.
- **Chapter 4** emphasised the importance of considering energy awareness in Cloud computing. This chapter discussed the proposed architecture aimed to enable energy awareness for heterogeneous resource management in Cloud computing**.** The architecture's components including the role of each component were explained. Early experiments to explore the architectural

impact of heterogeneous GPU architectures and key factors that affect performance, power and energy in Cloud computing were also performed.

- **Chapter 5** introduced the power and energy modeller in the proposed architecture. This chapter began with a presentation of the steps leading to the development of the GPU power model and was followed by the GPU energy model. The development of the GPU end-to-end energy framework was also presented. Finally, the power model, energy model and the GPU energy framework evaluation results were discussed.
- **Chapter 6** presented the scheduler component in the proposed architecture. This chapter reviewed the importance of developing scheduling policies to reduce energy consumption whilst maintaining the QoS requirements in Cloud computing. The energy-aware scheduling policy was introduced and was followed by the presentation of the energy and deadline aware scheduling policy. The evaluation of these policies with the consideration of the performance metrics, experimental setup and design of experiments was performed. Finally, the results of the developed scheduling policies were discussed.

7.2 Comparison of Research Approaches with the Related Work

Enabling energy consumption awareness and optimising performance whilst maintaining QoS requirements in heterogeneous Cloud computing has become an important and attractive research question. Power and energy modelling can be beneficial in optimising the power consumption of applications without harmfully affecting performance. Power and energy modelling can also save time and financial costs [106]. The usage of energy modelling in Cloud computing can support software developers and designers to develop energy aware applications. Furthermore, power and energy modelling can help Cloud computing providers to instantiate efficient energy-aware management techniques. As discussed in section 3.2, different models and approaches have been developed to predict GPU power in Cloud and non-cloud computing environments. Table 7.1 shows a comparison between the related conducted works for GPU power modelling and the GPU power model that has been developed in this research.

Table 7.1: Comparison of GPU power models

Criteria / Reference	Heterogeneous GPU architecture	Modelling Technique Considered	Cloud Computing Environment
[104]-[108]	No	Statistical and Machine Learning	No
[109]- [111]	Yes	Statistical and Machine Learning	No
[29], [112]	No	Low-Level Analytical Models	No
[31], [115]	Yes	Simulation-based	No
[116]	No	Power modelling with DVFS Technique	No
[117]- [119]	Yes	Power modelling with DVFS Technique	No
[23]	No	Statistical and Machine Learning	Yes
This Research	Yes	Deep Learning	Yes

As shown in Table 7.1, several modelling techniques have been used to predict GPU power consumption including statistical and machine learning techniques, low-level analytical models, simulation-based models, and power modelling integrated with different DVFS configurations. All these GPU models have been evaluated in non-Cloud computing environments, except for the power model in [23]. Although the power model in [23] has been evaluated on a Cloud computing environment, they considered a single GPU architecture to evaluate the model. Moreover, the power model in [23] has used a linear relationship for power modelling, which has been reported as inappropriate for power models on modern GPUs which have complex architectures [107]. Therefore, GPU power modelling in Cloud computing environments needs more consideration.

In addition, statistical and machine learning GPU power models have only used *hardware performance counters* as input parameters to develop GPU power models, but have not investigated other factors, such as *GPU and memory utilisations* to enhance the models' results. Considering only hardware

performance counters for developing power models can produce insufficient results because the sensors that measure these counters do not cover all the device components [185].

The power model developed in this thesis is different compared to other existing power models in the literature. It has considered the heterogeneity of GPU architectures in a Cloud computing environment without the need for deep knowledge of the GPU micro-architecture by using Deep Neural Networks (DNN), which is a part of Deep Learning. The model has also used a combination of selected hardware performance counters, and GPU and memory workloads (GPU utilisation and memory utilisation) to enhance the model's results. Therefore, this proposed model is unique compared to other current power models since it has considered the heterogeneity of GPU architectures in Cloud computing using DNN and the hybrid approach of inputs, as discussed in Chapter 5.

Several GPU *energy* modelling techniques in Cloud and non-cloud computing environments have been reviewed, as detailed in section 3.3. Table 7.2 shows a comparison between the related studies works for GPU energy modelling and the GPU energy model developed in this thesis.

Table 7.2: Comparison of GPU energy models

Criteria / Reference	Heterogeneous GPU architecture	Modelling Technique Considered	Cloud Computing Environment
[10], [29], [123], [109], [27]	No	Indirect	No
[124]	Yes	Indirect	No
[106]	No	Direct	No
[126]	Yes	Indirect	No
[24]	No	Indirect	Yes
This Research	Yes	Direct	Yes

As shown in Table 7.2, the considered GPU energy consumption modelling techniques can be divided into direct and indirect. Direct modelling techniques aim to directly estimate GPU energy consumption without developing several individual models, such as the energy model presented in [106]. Indirect models aim to estimate GPU energy consumption using a collection of individual sub models or

developing analytical models using several complex mathematical equations, such as the energy model in [124].

The majority of GPU energy consumption models have been performed on non-Cloud environments. The GPU energy model presented in [24] has been evaluated on a Cloud computing environment, but the heterogeneity of GPU architectures has not been used to evaluate the model. Moreover, the authors in [24] have developed GPU power and performance models individually. Then, they have estimated GPU energy consumption.

The energy model developed in this thesis is different compared to other existing energy models in the literature. The developed energy model has considered the heterogeneity of GPU architectures in a Cloud computing environment. It has aimed to directly estimate energy consumption by using DNNs without performing several sub-models. The model has also normalised the heterogeneity of GPU architectures by creating an agnostic model that has selected common input factors to represent the heterogeneous GPU architectures. Thus, the developed energy model in this thesis is unique because it has considered the heterogeneity of GPU architectures in Cloud computing, direct and agnostic using DNN, as discussed in Chapter 5.

To automatically estimate the GPU energy consumption without the need for profiling and collecting the required factors of the GPU applications running on a VM, it has been necessary to first predict the required resources such as the workload, power consumption, performance and temperature, which have been then used to estimate the energy consumption, as presented in section 3.4. Table 7.3 shows a comparison between the approaches for energy consumption estimation based on workload prediction in Cloud computing and the end-to-end GPU energy framework presented in this research.

Table 7.3: Comparison of Energy models based on Workload Prediction

Criteria / Reference	Consideration of GPU resources and workloads	Cloud Computing Environments
[156]	No	Yes
[157]	No	Yes
[158]	No	Yes
[159]	No	Yes
[160]	No	Yes
This Research	Yes	Yes

As shown in Table 7.3, the studies have proposed frameworks to predict energy consumption by firstly estimating the required resources and workloads in Cloud computing. Other studies, such as [157] has used the estimated resources to develop managerial actions like life migration to reduce energy consumption. However, all the presented studies have considered the resources of CPUs, the main memory and the network traffics, but they have neglected the required resources of GPUs to execute GPU applications in Cloud Computing environments**.**

The presented GPU end-to-end energy framework is different in this thesis in comparison with other existing energy models based on the workload prediction approach in the literature. The presented framework has considered the resources of heterogeneous GPU architectures in Cloud computing for running GPU applications. The presented framework has considered the required factors to estimate GPU energy consumption without the need for profiling the features of GPU applications. It has considered GPU execution time, GPU temperature, GPU resource usage and GPU memory usage. Therefore, based on the considered factors, the presented framework is different from other existing works, as discussed in Chapter 5.

GPU scheduling techniques performed in Cloud computing has been discussed in section 3.5. Table 7.4 presents a comparison between the related conducted works for GPU scheduling in Cloud computing and the scheduling policy model that has been developed in this research.

Table 7.4: GPU Scheduling in Cloud Computing Comparison

Criteria / By	Scheduling Consideration			GPU Architecture Type	Heterogeneous GPU architecture
	Performance	Energy Consumption	QoS		
[83], [164]	Yes	No	Yes	Integrated	No
[161], [162]	Yes	No	No	Integrated	No
[20], [165], [21]	Yes	No	Yes	Discrete	No
[19], [169], [75], [172], [173], [175], [22]	Yes	No	No	Discrete	No
[170], [177]	Yes	No	Yes	Discrete	Yes
[178], [23], [121], [24]	Yes	Yes	Yes	Discrete	No
[181]	Yes	Yes	No	Discrete	Yes
This Research	Yes	Yes	Yes	Discrete	Yes

As shown in Table 7.4, The studies performed in [83]-[162] and [164] have dealt with GPU scheduling in Cloud computing, and they have considered *integrated* GPUs. However, *discrete* GPUs have better computing capabilities than integrated GPUs. Even the studies in [20], [165], [19], [21]-[75], [172], [173], [175], [177] and [22] have used discrete GPU types for GPU scheduling in Cloud computing, they have only considered the performance factor for allocating virtual GPUs to physical GPUs, neglecting the energy consumption criterion.

The scheduling policy presented in this thesis is different in comparison with the previous scheduling policies. The proposed scheduling policy has considered discrete GPUs and has aimed to reduce the energy consumption by allocating GPU applications to the appropriate VM based on the outcomes of the GPU energy end-to-end framework. The scheduling policy has also considered two VMs, each of which has a different GPU architecture with different features. Moreover, the trade-off between reducing energy consumption and performance whilst maintaining the QoS requirements has been identified. Therefore, based on the

aforementioned factors the proposed energy-aware scheduling policy is unique, as discussed in Chapter 6.

Further, the studies in [178] and [121] have developed GPU energy-aware scheduling policies with the consideration of QoS in Cloud gaming but have used a single homogeneous GPU architecture. GPUCloudSim [23] has proposed scheduling policies aimed to reduce energy consumption; however, heterogeneous GPU architecture and QoS requirements have not been considered. Similarly, the work in [181] has not considered the application's deadline as a QoS requirement. Although the work in [24] has developed an energy-aware scheduling policy taking into account QoS requirements (specifically the deadline), a single homogeneous GPU architecture was considered to evaluate the scheduling policy.

The presented energy and deadline aware scheduling policy in this thesis is different compared to the previous scheduling policies. It has aimed to balance the trade-off between reducing energy consumption and performance whilst maintaining the QoS requirements when executing GPU applications in Cloud computing. It has dealt with heterogeneous GPUs, each of which is connected with different VMs. The presented policy has communicated with the GPU end-to-end energy framework to obtain the estimated energy and execution time to achieve the required goal of this policy. Thus, the presented energy and deadline aware policy is unique from other related studies, as discussed in Chapter 6.

7.3 Research Contributions

As discussed in section 1.2, several research objectives have been addressed to enable energy awareness to manage heterogeneous resources in Cloud computing. The major contributions that support the identified research objectives in this thesis are:

- *A systematic architecture for managing heterogeneous GPU resources in Cloud computing environments for general purpose usage.* In this architecture, the considered factors are performance and energy consumption. The architectural components, alongside their roles and interactions, are proposed to achieve the identified research objectives.
- *A GPU power consumption model in Cloud computing*. This model is aimed to predict the power consumed by GPU applications running on

heterogeneous Cloud computing infrastructures. The model considers different types of NVIDIA GPU generations. This model has addressed the first proposed research question **(Q.1)**.

- *A GPU energy consumption model in Cloud computing*. This model aims to automatically estimate the energy consumed by GPU applications executed on heterogeneous Cloud computing infrastructures. This model has considered different types of NVIDIA GPU generations and has addressed the second proposed research question **(Q.2)**.
- *A GPU end-to-end energy prediction framework in Cloud computing.* This framework aims to directly estimate the energy consumed by GPU applications executed on heterogeneous Cloud infrastructures without performing experiments to collect the features of GPU applications that are used to train the energy model. This framework includes various mathematical models for addressing the proposed research question **(Q.3)**.
- *An energy-aware scheduling policy*. This policy communicates with the end-to-end framework to obtain the predicted energy and execution time of GPU applications. This policy aims to reduce energy consumption by scheduling GPU applications to the appropriate Virtual Machine (VM) in heterogeneous Cloud computing infrastructures. The policy has been evaluated through computational modelling. The trade-off between performance and energy consumption whilst maintaining the QoS requirements has been identified to address the research question **(Q.4)**.
- *An energy and deadline aware scheduling policy.* This policy has interacted with the end-to-end framework to obtain the predicted energy and execution time of GPU applications. This policy has aimed to balance the trade-off between performance and energy consumption whilst maintaining the QoS requirements when allocating GPU applications to different VMs on heterogeneous Cloud infrastructure. This policy has been evaluated via computational modelling, and the policy's results have been compared with different scheduling algorithms, which addressed the research question **(Q.5)**.

7.4 Limitations

The conducted evaluations of this work through direct experiments for the developed models and simulation for the designed scheduling policies have shown promising results to enable energy awareness to manage heterogeneous resources of Cloud computing. However, the research has a few limitations:

- Each VM in Cloud computing consists of different virtual resources, such as CPUs, memory and network traffics. These resources affect the total amount of VM's energy consumption. Although the workload has been executed by the GPU, the developed energy consumption models in this thesis have only considered the energy consumed by the GPU and neglected the other resources.
- Cloud computing is known for the diversity and heterogeneity of its resources, and Cloud computing infrastructure hosts a great number of VMs and PMs. However, this work has only considered two VMs and two types of NVIDIA GPU architectures in Cloud computing. This is because of the high cost of performing experiments on a real Cloud testbed containing a large number of heterogeneous GPUs. Simulations tools that support heterogeneous GPUs architectures in Cloud computing environments have also not been available.

7.5 Future Work

Some directions are identified to further extend the presented work in this thesis. The suggested directions to extend this work are:

- The energy model presented in this work has merely considered the energy consumed by the GPU when a GPU application is executed on a VM in Cloud computing. A promising extension to this model is by considering other resources including CPU, memory and network traffic to estimate the total energy consumption of the VM.
- The work presented in this thesis has aimed to enable energy awareness to manage heterogeneous resources in VM instances in Cloud computing, which are based on the hypervisors virtualisation type. A potential extension of this work is to enable energy awareness to manage

heterogeneous resources in container-based virtualisation, a different evolving virtualisation technology. This extension will be beneficial to provide energy awareness in different types of virtualisation techniques in Cloud computing.

- The presented work in this thesis is a part of the proposed architecture discussed in section 4.3 to manage GPU applications during the deployment and operation time in Cloud computing. This work has only considered managing GPU applications in a Cloud computing environment at deployment time. Another potential extension of this work is, therefore to manage GPU applications during the run time once they are deployed on hardware resources. This extension will aim to fulfil the QoS requirements by preventing performance degradation and reduce energy consumption. This can be achieved by developing a self-adaptive system using the MAPE-K [190] (Monitor, Analyse, Plan, Execute and Knowledge) approach. Therefore, by developing this adaptive system, the life cycle of the GPU application from the deployment time to the run time executed on a Cloud computing environment will be covered.
- The presented work in this thesis has investigated the features of two different types of NVIDIA GPU architectures (Fermi and Kepler) to enable energy awareness for managing heterogeneous Cloud computing resources. Another potential extension of this work is to increase the heterogeneity of GPU architectures, e.g. through the consideration of Maxwell, Pascal and Volta NVIDIA architectures running together in Cloud computing environments. This will be beneficial for the representation of Cloud computing scalability and the development of agnostic energy models that can represent different GPU architectures.

Appendix A

Table A.1 lists shared hardware performance counters between Fermi and Kepler GPUs and performance counters specific for Kepler GPU.

Table A.1: Shared Hardware Performance Counters between Fermi and Kepler and specific for Kepler GPU

Counter Name	Description	GPU
sm_efficiency	The percentage value of time at least one warp is active on a single multiprocessor	Fermi+Kepler
sm_efficiency_instance	The percentage value of time at least one warp is active on a specific multiprocessor	Fermi+Kepler
achieved_occupancy	The ratio of the average of active warps per active cycle	Fermi+Kepler
issue_slot_utilization	Percentage of issue slots which issued at least for a single instruction	Fermi+Kepler
inst_executed	The instructions number executed	Fermi+Kepler
inst_issued	The instructions number issued	Fermi+Kepler
issue_slots	The number of issue slots utilised	Fermi+Kepler
executed_ipc	Instructions number executed per cycle	Fermi+Kepler
issued_ipc	Instructions number issued per cycle	Fermi+Kepler
ipc_instance	Instructions executed per cycle on one multiprocessor	Fermi+Kepler
inst_per_warp	The average number of instructions run by every warp	Fermi+Kepler
cf_issued	The issued control-flow instructions number	Fermi+Kepler
cf_executed	The executed control-flow instructions number	Fermi+Kepler
ldst_issued	issued load and store instructions number	Fermi+Kepler
ldst_executed	The executed load and storeinstructions number	Fermi+Kepler
branch_efficiency	The ratio of non-divergent branches	Fermi+Kepler
warp_execution_efficiency	The ratio of the average of active threads on every warp	Fermi+Kepler
inst_replay_overhead	The average of replays on every instruction executed	Fermi+Kepler

Counter Name	Description	GPU
shared_replay_overhead	The average of replays due to shared memory conflicts	Fermi+Kepler
global_cache_replay_overhead	The average of replays on global memory cache misses on every instruction executed	Fermi+Kepler
local_replay_overhead	The average of replays because of local memory accesses on every instruction executed	Fermi+Kepler
gld_efficiency	The ratio of the requested global memory load throughput	Fermi+Kepler
gst_efficiency	The ratio of requested global memory store throughput to required global memory store throughput	Fermi+Kepler
gld_transactions	The global memory load transactions number	Fermi+Kepler
gst_transactions	The global memory store transactions number	Fermi+Kepler
gld_transactions_per_request	The average of global memory load transactions conducted for every global memory load	Fermi+Kepler
gst_transactions_per_request	The average of global memory store transactions conducted on every global memory store	Fermi+Kepler
gld_throughput	The global memory load throughput number	Fermi+Kepler
gst_throughput	The global memory store throughput number	Fermi+Kepler
gld_requested_throughput	The number of the requested global memory load throughput	Fermi+Kepler
gst_requested_throughput	The global memory store throughput requested number	Fermi+Kepler
local_load_transactions	The number of the local memory load transactions	Fermi+Kepler
local_store_transactions	The local memory store transactions number	Fermi+Kepler
local_load_transactions_per_ request	The average number of local memory load transactions that conducted for every local memory load	Fermi+Kepler
local_store_transactions_per_ request	The average number of the local memory store transactions conducted for every local memory store	Fermi+Kepler
local_load_throughput	The Local memory load throughput number	Fermi+Kepler

Counter Name	Description	GPU
local_store_throughput	The Local memory store throughput number	Fermi+Kepler
shared_load_transactions	The shared memory load transactions number	Fermi+Kepler
shared_store_transactions	The shared memory store transactions number	Fermi+Kepler
shared_load_transactions_per_ request	The average number of shared memory load transactions conducted for every shared memory load operation	Fermi+Kepler
shared_store_transactions_per_ request	The average number of shared memory store transactions conducted for every shared memory store operation	Fermi+Kepler
shared_load_throughput	The shared memory load throughput number	Fermi+Kepler
shared_store_throughput	The shared memory store throughput number	Fermi+Kepler
shared_efficiency	The ratio of requested shared memory throughput number	Fermi+Kepler
dram_read_transactions	The device memory read transactions number	Fermi+Kepler
dram_write_transactions	The device memory write transactions number	Fermi+Kepler
dram_read_throughput	The device memory read throughput number	Fermi+Kepler
dram_write_throughput	The device memory write throughput number	Fermi+Kepler
sysmem_read_transactions	The number of system memory read transactions	Fermi+Kepler
sysmem_write_transactions	The number of system memory write transactions	Fermi+Kepler
sysmem_read_throughput	The number of system memory read throughput	Fermi+Kepler
sysmem_write_throughput	The number of system memory write throughput	Fermi+Kepler
l1_cache_global_hit_rate	The hit rate in L1 cache memory for the global loads number	Fermi+Kepler
l1_cache_local_hit_rate	The hit rate in L1 cache memory for local loads and stores number	Fermi+Kepler
tex_cache_hit_rate	The texture cache hit rate number	Fermi+Kepler
tex_cache_transactions	The texture cache read transactions number	Fermi+Kepler

Counter Name	Description	GPU
tex_cache_throughput	The texture cache throughput number	Fermi+Kepler
l2_read_transactions	Memory read transactions seen at L2 cache for all read requests	Fermi+Kepler
l2_write_transactions	The memory write transactions number found in the L2 cache	Fermi+Kepler
l2_read_throughput	The memory read throughput number found in the L2 cache	Fermi+Kepler
l2_write_throughput	The memory write throughput number found in the L2 cache	Fermi+Kepler
l2_l1_read_hit_rate	The hit rate number in the L2 cache for the entire read requests from the L1 cache	Fermi+Kepler
l2_l1_read_throughput	The memory read throughput number found for the L2 cache for reading requests from the L1 cache	Fermi+Kepler
l2_texture_read_hit_rate	The hit rate number in the L2 cache for all read requests from the texture cache	Fermi+Kepler
l2_texure_read_throughput	The memory read throughput number found in the L2 cache for reading requests from the texture cache	Fermi+Kepler
local_memory_overhead	The ratio of the local memory traffic number to the total memory traffic number between the L1 and L2 caches	Fermi+Kepler
l1_shared_utilization	The L1/shared memory utilisation level	Fermi+Kepler
l2_utilization	The L2 cache utilisation level	Fermi+Kepler
tex_utilization	The texture cache utilisation level	Fermi+Kepler
dram_utilization	The device memory utilisation level	Fermi+Kepler
sysmem_utilization	The system memory utilisation level	Fermi+Kepler
ldst_fu_utilization	The multiprocessor function units which run global, local and shared memory instructions utilisation level	Fermi+Kepler
cf_fu_utilization	The multiprocessor function units which run control-flow instructions utilisation level	Fermi+Kepler
tex_fu_utilization	The utilisation level of the multiprocessor function units which run the texture instructions	Fermi+Kepler

Counter Name	Description	GPU
alu_fu_utilization	The utilisation level of the multiprocessor function units which run the integer and floating-point arithmetic instructions	Fermi+Kepler
inst_fp_32	The number of single-precision floating-point instructions run by the non-predicated threads	Fermi+Kepler
inst_fp_64	The number of double-precision floating-point instructions run by non-predicated threads	Fermi+Kepler
inst_integer	The integer instructions number run by non-predicated threads	Fermi+Kepler
inst_bit_convert	The bit-conversion instructions number run by non-predicated threads	Fermi+Kepler
inst_control	The number of control-flow instructions run by non-predicated threads	Fermi+Kepler
inst_compute_ld_st	the compute load/store instructions number run by non-predicated threads	Fermi+Kepler
inst_inter_thread_communication	The inter-thread communication instructions number run by non-predicated threads	Fermi+Kepler
inst_misc	The miscellaneous instructions number executed by non-predicated threads	Fermi+Kepler
flops_sp	The number of single-precision floating-point operations run	Fermi+Kepler
flops_sp_add	The number of single-precision floating-point add operations run	Fermi+Kepler
flops_sp_mul	The number of executed single-precision floating-point multiply operations	Fermi+Kepler
flops_sp_fma	The number of executed single-precision floating-point multiply-accumulate operations	Fermi+Kepler
flops_dp	The number of executed double-precision floating-point operations	Fermi+Kepler
flops_dp_add	The number of executed double-precision floating-point add operations	Fermi+Kepler
flops_dp_mul	The number of executed double-precision floating-point multiply operations	Fermi+Kepler
flops_dp_fma	The number of executed double-precision floating-point multiply-accumulate operations	Fermi+Kepler
flops_sp_special	The number of executed single-precision floating-point special operations	Fermi+Kepler

Counter Name	Description	GPU
stall_exec_dependency	The Percentage number of stalls occurring due to an input requested by the instruction is not available	Fermi+Kepler
stall_data_request	The percentage number of stalls occurring due to a memory operation cannot be conducted	Fermi+Kepler
stall_sync	The percentage number of stalls occurring due to the warp is closed	Fermi+Kepler
stall_texture	The percentage number of stalls happening due to the texture sub-system is fully occupied	Fermi+Kepler
stall_other	The percentage number of stalls happening due to miscellaneous reasons	Fermi+Kepler
ipc	The instructions number issued per cycle	Kepler Only
warp_nonpred_execution_efficiency	The ratio number of the average active threads per warp running on non-predicated instructions to the highest number of threads per warp in a multiprocessor	Kepler Only
global_replay_overhead	The average number of replays because of the global memory cache faults	Kepler Only
nc_l2_read_throughput	The memory read throughput number for non coherent global read needs found in the L2 cache	Kepler Only
nc_l2_read_transactions	The memory read transactions number found in the L2 cache for non coherent global read needs	Kepler Only
nc_cache_global_hit_rate	The hit rate number in non coherent cache in global loads transactions	Kepler Only
nc_gld_throughput	The throughput of non coherent global memory loads	Kepler Only
nc_gld_requested_throughput	The requested throughput number on the global memory loaded by non-coherent cache	Kepler Only
nc_gld_efficiency	The ratio number of the required non coherent global memory load throughput	Kepler Only
l2_atomic_throughput	The memory read throughput number found in the L2 cache for atomic requests	Kepler Only
atomic_replay_overhead	The average number of replays for atomic and reduction bank conflicts for every executed instruction	Kepler Only
atomic_transactions	The global memory reduction and atomic transactions number	Kepler Only
atomic_transactions_per_request	The global memory reduction and atomic average number of reduction transactions performed for each reduction and atomic instruction	Kepler Only

References

[1] “Consumer cloud computing users worldwide 2018 | Statista.” [Online]. Available: https://www.statista.com/statistics/321215/global-consumer-cloud-computing-users/. [Accessed: 25-Nov-2020].

[2] CISCO, “Cisco Annual Internet Report 2018–2023 White Paper,” 2020.

[3] “Public cloud computing market size 2022 | Statista.” [Online]. Available: https://www.statista.com/statistics/273818/global-revenue-generated-with-cloud-computing-since-2009/. [Accessed: 25-Nov-2020].

[4] “Amazon EC2 P3 – Ideal for Machine Learning and HPC - AWS.” [Online]. Available: https://aws.amazon.com/ec2/instance-types/p3/. [Accessed: 12-Nov-2020].

[5] “Applications that scale using GPU Compute.” [Online]. Available: https://channel9.msdn.com/Events/Microsoft-Azure/AzureCon-2015/ACON303. [Accessed: 22-Feb-2016].

[6] “Graphics Processing Units (GPU) | Google Cloud Platform.” [Online]. Available: https://cloud.google.com/gpu/. [Accessed: 20-Aug-2017].

[7] A. Beloglazov and R. Buyya, “Energy efficient resource management in virtualized cloud data centers,” in *CCGrid 2010 - 10th IEEE/ACM International Conference on Cluster, Cloud, and Grid Computing*, 2010, pp. 826–831.

[8] “Artificial Intelligence Computing Leadership from NVIDIA.” [Online]. Available: https://www.nvidia.com/en-gb/. [Accessed: 30-Nov-2020].

[9] C. M. Wittenbrink, E. Kilgariff, and A. Prabhu, “Fermi GF100 GPU architecture,” *IEEE Micro*, vol. 31, no. 2, pp. 50–59, 2011.

[10] T. Fastest, “Kepler GK110.”

[11] T. Mukherjee, K. Dasgupta, S. Gujar, G. Jung, and H. Lee, “An economic model for green cloud,” in *Proceedings of the 10th International Workshop on Middleware for Grids, Clouds and e-Science, MGC 2012*, 2012, no. December.

[12] J. Hamilton, “Cooperative expendable micro-slice servers (CEMS): Low cost, low power servers for internet-scale services,” in *Conference on Innovative Data Systems Research (CIDR’09)*, 2009, pp. 1–8.

[13] J. Shuja, A. Gani, S. Shamshirband, R. W. Ahmad, and K. Bilal, “Sustainable Cloud Data Centers: A survey of enabling techniques and technologies,” *Renew. Sustain. Energy Rev.*, vol. 62, pp. 195–214, 2016.

[14] A. Shehabi *et al.*, “United States Data Center Energy Usage Report,” Berkeley, California, 2016.

[15] N. Jones, “The information factories,” *Nature*, vol. 561, no. 7722, pp. 163–166, 2018.

[16] A. Andrae and T. Edler, “On Global Electricity Usage of Communication Technology: Trends to 2030,” *Challenges*, vol. 6, no. 1, pp. 117–157, 2015.

[17] B. Jennings and R. Stadler, “Resource Management in Clouds: Survey and

Research Challenges," *J. Netw. Syst. Manag.*, pp. 1–53, 2014.

[18] S. Crago *et al.*, "Heterogeneous cloud computing," in *Proceedings - IEEE International Conference on Cluster Computing, ICCC*, 2011, pp. 378–385.

[19] V. T. Ravi, M. Becchi, G. Agrawal, and S. Chakradhar, "Supporting GPU sharing in cloud environments with a transparent runtime consolidation framework," in *Proceedings of the 20th international symposium on High performance distributed computing - HPDC '11*, 2011, p. 217.

[20] V. Gupta *et al.*, "GViM: GPU-accelerated Virtual Machines," in *the 3rd ACM Workshop on System-level Virtualization for High Performance Computing*, 2009, pp. 1–8.

[21] Z. Qi, J. Yao, C. Zhang, M. Yu, Z. Yang, and H. Guan, "VGRIS: Virtualized GPU Resource Isolation and Scheduling in Cloud Gaming," *ACM Trans. Archit. Code Optim.*, vol. 11, no. 2, pp. 1–25, 2014.

[22] J. Oh and Y. Kim, "Job placement using reinforcement learning in GPU virtualization environment," *Cluster Comput.*, vol. 7, 2020.

[23] A. Siavashi and M. Momtazpour, "GPUCloudSim: an extension of CloudSim for modeling and simulation of GPUs in cloud data centers," *J. Supercomput.*, vol. 75, no. 5, pp. 2535–2561, 2019.

[24] S. Ilager, R. Muralidhar, K. Rammohanrao, and R. Buyya, "A Data-Driven Frequency Scaling Approach for Deadline-aware Energy Efficient Scheduling on Graphics Processing Units (GPUs)," *arXiv Prepr. arXiv2004.08177*, 2020.

[25] J. W. Creswell, *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*. Sage publications, 2013.

[26] V. Adhinarayanan, B. Subramaniam, and W. Feng, "Online Power Estimation of Graphics Processing Units," in *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2016, no. May.

[27] G. Wu, J. L. Greathouse, A. Lyashevsky, N. Jayasena, and D. Chiou, "GPGPU performance and power estimation using machine learning," in *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, 2015, pp. 564–576.

[28] R. N. Calheiros, R. Ranjany, A. Beloglazov, C. a. F. De Rose, and R. Buyya, "CloudSim: a toolkit formodeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw. - Pract. Exp.*, vol. 41(1), no. 7, pp. 23–50, 2011.

[29] S. Hong and H. Kim, "An integrated GPU power and performance model," in *Proceedings of the 37th annual international symposium on Computer architecture*, 2010, pp. 280–289.

[30] Y. Arafa, A. H. A. Badawy, G. Chennupati, N. Santhi, and S. Eidenbenz, "PPT-GPU: Scalable GPU Performance Modeling," *IEEE Comput. Archit. Lett.*, vol. 18, no. 1, pp. 55–58, 2019.

[31] J. Leng *et al.*, "GPUWattch: Enabling Energy Optimizations in GPGPUs," *ACM SIGARCH Comput. Archit. News*, vol. 41, no. 3, pp. 487–498, 2013.

[32] R. E. Kavanagh, “Negotiated Resource Brokering for Quality of Service Provision of Grid Applications,” 2013.

[33] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, “cloud computing: Vision, hype, and reality of delivering computing as the 5th utility,” *Futur. Gener. Comput. Syst.*, vol. 25, no. 6, pp. 599–616, 2009.

[34] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, “A break in the clouds: towards a cloud definition,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 50–55, 2008.

[35] P. Mell and T. Grance, “The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology,” p. 7, 2011.

[36] L. Wang, G. Von Laszewski, M. Kunze, and J. Tao, “Cloud computing: A Perspective study,” *Proc. Grid Comput. Environ. Work.*, no. November, pp. 1–11, 2008.

[37] Q. Zhang, L. Cheng, and R. Boutaba, “Cloud computing: state-of-the-art and research challenges,” *J. Internet Serv. Appl.*, vol. 1, no. 1, pp. 7–18, 2010.

[38] S. Nanda and T. Chiueh, “A Survey on Virtualization Technologies,” in *RPE Report*, 2005, pp. 1–42.

[39] P. Barham *et al.*, “Xen and the Art of Virtualization,” *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 164–177, 2003.

[40] “KVM.” [Online]. Available: http://www.linux-kvm.org/page/Main_Page. [Accessed: 21-Feb-2016].

[41] “VMware Virtualization for Desktop & Server, Application, Public & Hybrid Clouds.” [Online]. Available: http://www.vmware.com/. [Accessed: 21-Feb-2016].

[42] “OpenNebula.” [Online]. Available: https://opennebula.org/. [Accessed: 26-Oct-2017].

[43] “Apache CloudStack: Open Source Cloud Computing.” [Online]. Available: https://cloudstack.apache.org/. [Accessed: 12-Sep-2018].

[44] “Open source software for creating private and public clouds.” [Online]. Available: https://www.openstack.org/. [Accessed: 12-Sep-2018].

[45] R. Morabito, V. Cozzolino, A. Y. Ding, N. Beijar, and J. Ott, “Consolidate IoT Edge Computing with Lightweight Virtualization,” *IEEE Softw.*, vol. 32, no. 1, pp. 102–111, 2015.

[46] R. Morabito, J. Kjällman, and M. Komu, “Hypervisors vs. lightweight virtualization: A performance comparison,” in *In 2015 IEEE International Conference on Cloud Engineering*, 2015, pp. 386–393.

[47] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, “An updated performance comparison of virtual machines and linux containers,” in *2015 IEEE international symposium on performance analysis of systems and software (ISPASS)*, 2015, pp. 171–172.

[48] “Empowering App Development for Developers | Docker.” [Online]. Available: https://www.docker.com/. [Accessed: 03-Nov-2020].

[49] “Linux Containers.” [Online]. Available: https://linuxcontainers.org/. [Accessed: 03-Nov-2020].

[50] “GitHub - cloudfoundry-attic/warden: Cloud Foundry - the open platform as a service project.” [Online]. Available: https://github.com/cloudfoundry-attic/warden. [Accessed: 03-Nov-2020].

[51] “NGC: GPU-Optimized Software for DL, ML and HPC Workflows | NVIDIA.” [Online]. Available: https://www.nvidia.com/en-us/gpu-cloud/. [Accessed: 03-Nov-2020].

[52] “GitHub - NVIDIA/nvidia-docker: Build and run Docker containers leveraging NVIDIA GPUs.” [Online]. Available: https://github.com/NVIDIA/nvidia-docker. [Accessed: 03-Nov-2020].

[53] J. P. Walters *et al.*, “GPU Passthrough Performance: A Comparison of KVM, Xen, VMWare ESXi, and LXC for CUDA and OpenCL Applications,” in *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*, 2014, pp. 636–643.

[54] “Schedule GPUs | Kubernetes.” [Online]. Available: https://kubernetes.io/docs/tasks/manage-gpus/scheduling-gpus/. [Accessed: 03-Nov-2020].

[55] D. Luebke and G. Humphreys, “How GPUs work,” *Computer (Long. Beach. Calif).*, vol. 40, no. 2, pp. 96–100, 2007.

[56] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, “NVIDIA Tesla: A unified graphics and computing architecture,” *IEEE Micro*, vol. 28, no. 2, pp. 39–55, 2008.

[57] M. J. Flynn, “Some computer organization and their effectiveness,” *IEEE Trans. Comput.*, vol. C–21, no. 9, pp. 948–960, 1972.

[58] NVIDIA, “NVIDIA GeForce GTX 980 Featuring Maxwell, The Most Advanced GPU Ever Made.,” 2014.

[59] NVIDIA Corporation, “NVIDIA Tesla P100 Whitepaper,” 2016.

[60] “Artificial Intelligence Architecture | NVIDIA Volta.” [Online]. Available: https://www.nvidia.com/en-us/data-center/volta-gpu-architecture/. [Accessed: 02-Aug-2018].

[61] “CUDA C Programming Guide.”

[62] N. Rath *et al.*, “High-speed, multi-input, multi-output control using GPU processing in the HBT-EP tokamak,” *Fusion Eng. Des.*, vol. 87, no. 12, pp. 1895–1899, 2012.

[63] B. He *et al.*, “Relational Joins on Graphics Processors,” in *ACM SIGMOD international conference on Management of data*, 2008, pp. 511–524.

[64] M. Silberstein, B. Ford, I. Keidar, and E. Witchel, “GPUfs: Integrating a file system with GPUs,” in *Proceedings of the 18th international conference on*

Architectural Support for Programming Languages and Operating Systems, 2013, vol. 32, no. 1, pp. 485–498.

[65] M. McNaughton, C. Urmson, J. M. Dolan, and J. W. Lee, “Motion planning for autonomous driving with a conformal spatiotemporal lattice,” in *Proceedings - IEEE International Conference on Robotics and Automation*, 2011, vol. 1, pp. 4889–4895.

[66] S. Han and K. Park, “PacketShader : A GPU-Accelerated Software Router,” in *ACM SIGCOMM Computer Communication Review*, 2010, vol. 40, no. 4, pp. 195–206.

[67] K. Jang, S. S. Han, S. Moon, and K. Park, “SSLShader : Cheap SSL Acceleration with Commodity Processors,” in *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation*, 2011, p. 1.

[68] J. Owens and M. Houston, “GPU computing,” *Proc. IEEE*, vol. 96, no. 5, pp. 879–899, 2008.

[69] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, “Inception-v4, inception-ResNet and the impact of residual connections on learning,” in *arXiv preprint arXiv:1602.07261.*, 2016.

[70] “OpenCL - The open standard for parallel programming of heterogeneous systems.” [Online]. Available: https://www.khronos.org/opencl/. [Accessed: 21-Feb-2016].

[71] L. Silene, “OpenCL: A parallel programming standard for heterogeneous computing systems,” vol. 86, no. 3, pp. 354–366, 1999.

[72] “Parallel Programming and Computing Platform | CUDA | NVIDIA|NVIDIA.” [Online]. Available: http://www.nvidia.com/object/cuda_home_new.html. [Accessed: 21-Feb-2016].

[73] J. Nickolls, I. Buck, M. Garland, and K. Skadron, “Scalable parallel programming with CUDA,” *ACM SIGGRAPH 2008 classes - SIGGRAPH ’08*, no. April, p. 1, 2008.

[74] J. Diaz, C. Munoz-Caro, and a Nino, “A Survey of Parallel Programming Models and Tools in the Multi and Many-Core Era,” *Parallel Distrib. Syst. IEEE Trans.*, vol. 23, no. 8, pp. 1369–1386, 2012.

[75] Y. Suzuki, S. Kato, H. Yamada, and K. Kono, “GPUvm: GPU Virtualization at the Hypervisor,” *IEEE Trans. Comput.*, vol. 65, no. 9, pp. 2752–2766, 2016.

[76] L. Shi, H. Chen, J. Sun, and K. Li, “vCUDA: GPU-Accelerated High-Performance Computing in Virtual Machines,” *IEEE Trans. Comput.*, vol. 61, no. 6, pp. 804–816, 2012.

[77] R. Montella, G. Coviello, G. Giunta, G. Laccetti, F. Isaila, and J. G. Blas, “A general-purpose virtualization service for HPC on cloud computing : an application to GPUs,” in *Parallel Processing and Applied Mathematics*, 2012, pp. 740–749.

[78] J. Duato, A. J. Peña, F. Silla, R. Mayo, and E. S. Quintana-Ort, “rCUDA: Reducing the number of GPU-based accelerators in high performance

clusters," in *Proceedings of the 2010 International Conference on High Performance Computing and Simulation, HPCS 2010*, 2010, pp. 224–231.

[79] "XenServer - Server Virtualization and Consolidation - Citrix - Citrix." [Online]. Available: https://www.citrix.co.uk/products/xenserver/. [Accessed: 17-Jul-2018].

[80] I. Conference and C. C. Technology, "On Implementation of GPU Virtualization Using PCI Pass-Through," in *Proceedings of IEEE CloudCom 2012*, 2012, pp. 578–580.

[81] M. Dowty and J. Sugerman, "GPU virtualization on VMware's hosted I/O architecture," in *ACM SIGOPS Operating Systems Review*, 2009, vol. 43, no. 3, p. 73.

[82] M. Gottschlag, M. Hillenbrand, J. Kehne, J. Stoess, and F. Bellosa, "LoGV: Low-overhead GPGPU Virtualization," in *Proceedings of the 4th International Workshop on Frontiers of Heterogeneous Computing*, 2013, pp. 1721–1726.

[83] K. Tian, Y. Dong, and D. Cowperthwaite, "A Full GPU Virtualization Solution with Mediated Pass-Through," in *2014 USENIX Annual Technical Conference*, 2014, pp. 121–132.

[84] "June 2018 | TOP500 Supercomputer Sites." [Online]. Available: https://www.top500.org/lists/2018/06/. [Accessed: 12-Jul-2018].

[85] S. P. Crago, J. P. Walters, and S. California, "Heterogeneous Cloud Computing : The Way Forward," 2015.

[86] "AWS | High Performance Computing - HPC Cloud Computing." [Online]. Available: http://aws.amazon.com/hpc/. [Accessed: 21-Feb-2016].

[87] "IBM Bluemix - GPUs Cloud Computing - More processing power." [Online]. Available: https://www.ibm.com/cloud-computing/bluemix/gpu-computing. [Accessed: 20-Aug-2017].

[88] "Nimbix: High Performance Computing & Supercomputing Platform." [Online]. Available: https://www.nimbix.net/. [Accessed: 20-Aug-2017].

[89] "Elastic GPU Service: Powerful Computing Capabilities for Deep Learning - Alibaba Cloud." [Online]. Available: https://www.alibabacloud.com/product/gpu. [Accessed: 03-Nov-2020].

[90] S. Crago *et al.*, "Heterogeneous cloud computing," 2011, no. September.

[91] Y. Suzuki, S. Kato, H. Yamada, and K. Kono, "GPUvm : Why Not Virtualizing GPUs at the Hypervisor?," in *USENIX Annual Technical Conference (USENIX ATC)*, 2014, pp. 109–210.

[92] T. Mitchell, *Machine Learning*. New York: McGraw Hill, 1997.

[93] S. Marsland, *Machine Learning: An Algorithmic Perspective*. CRC press, 2015.

[94] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge: MIT press, 2016.

[95] L. Deng and D. Yu, "Deep learning: Methods and applications," *Found. Trends Signal Process.*, vol. 7, no. 3–4, pp. 197–387, 2014.

[96] G. Hinton *et al.*, “Deep neural network for acoustic modoling in speech recognition :The shared views of four research groups,” *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[97] Y. Lecun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[98] M. Dayarathna, Y. Wen, and R. Fan, “Data Center Energy Consumption Modeling: A Survey,” *IEEE Commun. Surv. Tutorials*, vol. 18, no. 1, pp. 732–794, 2016.

[99] R. Buyya, V. Christian, and S. Thamarai, *Mastering Cloud Computing Foundations and Applications Programming*. Boston: Morgan Kaufmann, 2013.

[100] Y. Gao, H. Guan, Z. Qi, B. Wang, and L. Liu, “Quality of service aware power management for virtualized data centers,” *J. Syst. Archit.*, vol. 59, no. 4–5, pp. 245–259, 2013.

[101] Global e-Sustainability Initiative, “GeSI SMARTer 2020 : The Role of ICT in Driving a Sustainable Future,” Global e- sustiainibility initiative, Brussels, Belgium, 2012.

[102] G. Cook, “How clean is your cloud?,” Greenpeace International, 2012.

[103] A. Strategy, “#SMARTer2030-ICT Solutions for 21st Century Challenges,” The Global eSustainability Initiativessels, Brussels-Capital Region, Belgium, Tech. Rep., 2015.

[104] X. Ma and L. Zhong, “Statistical Power Consumption Analysis and Modeling for GPU-based Computing,” in *Proceedings of the SOSP Workshop on Power Aware Computing and Systems (HotPower ’09)*, 2009, pp. 115–122.

[105] H. Nagasaka, N. Maruyama, A. Nukada, T. Endo, and S. Matsuoka, “Statistical power modeling of GPU kernels using performance counters,” in *International Conference on Green Computing, Green Comp*, 2010, pp. 115–122.

[106] S. Ghosh, S. Chandrasekaran, and B. M. Chapman, “Statistical Modeling of Power/Energy of Scientific Kernels on a Multi-GPU system,” in *2013 International Green Computing Conference Proceedings*, 2013, pp. 1–6.

[107] S. Song, C. Su, B. Rountree, and K. W. Cameron, “A simplified and accurate model of power-performance efficiency on emergent GPU architectures,” in *Proceedings - IEEE 27th International Parallel and Distributed Processing Symposium, IPDPS 2013*, 2013, pp. 673–686.

[108] H. Wang and Q. Chen, “Power estimating model and analysis of general programming on GPU,” *J. Softw.*, vol. 7, no. 5, pp. 1164–1170, 2012.

[109] Y. Abe, H. Sasaki, S. Kato, K. Inoue, M. Edahiro, and M. Peres, “Power and performance characterization and modeling of GPU-accelerated systems,” in *Proceedings of the International Parallel and Distributed Processing Symposium, IPDPS*, 2014, pp. 113–122.

[110] D. Zhao and Q. Chen, “Current Prediction Model of GPU Oriented to General

Purpose Computing," *IEEE Access*, vol. 7, pp. 127920–127931, 2019.

[111] L. Braun, S. Nikas, C. Song, V. Heuveline, and H. Fröning, "A Simple Model for Portable and Fast Prediction of Execution Time and Power Consumption of GPU Kernels," *arXiv Prepr. arXiv2001.07104*, 2020.

[112] K. Kasichayanula, D. Terpstra, P. Luszczek, S. Tomov, S. Moore, and G. D. Peterson, "Power aware computing on GPUs," *Symp. Appl. Accel. High-Performance Comput.*, pp. 64–73, 2012.

[113] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," *Proc. Int. Symp. Microarchitecture*, no. c, pp. 469–480, 2009.

[114] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," in *ISPASS 2009 - International Symposium on Performance Analysis of Systems and Software*, 2009, pp. 163–174.

[115] J. Lucas, S. Lal, M. Andersch, M. Alvarez-Mesa, and B. Juurlink, "How a single chip causes massive power bills GPUSimPow: A GPGPU power simulator," in *ISPASS 2013 - IEEE International Symposium on Performance Analysis of Systems and Software*, 2013, pp. 97–106.

[116] B. Dutta, V. Adhinarayanan, and W. C. Feng, "GPU power prediction via ensemble machine learning for DVFS space exploration," in *Proceedings of the 15th ACM International Conference on Computing Frontiers*, 2018, pp. 240–243.

[117] A. Ilic, N. Roma, and P. Tom, "GPGPU Power Modeling for Multi-Domain Voltage-Frequency Scaling," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018, pp. 789–800.

[118] J. Guerreiro, A. Ilic, N. Roma, and P. Tomas, "Modeling and Decoupling the GPU Power Consumption for Cross-Domain DVFS," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 11, pp. 2494–2506, 2019.

[119] J. Guerreiro, A. Ilic, N. Roma, and P. Tomas, "GPU Static Modeling Using PTX and Deep Structured Learning," *IEEE Access*, vol. 7, pp. 159150–159161, 2019.

[120] A. Herrera, "NVIDIA GRID: Graphics accelerated VDI with the visual performance of a workstation," *Nvidia Corp*, no. May, pp. 1–18, 2014.

[121] H. Guan, J. Yao, Z. Qi, and R. Wang, "Energy-Efficient SLA Guarantees for Virtualized GPU in Cloud Gaming," *IEEE Trans. Parallel Distrib. Syst.*, vol. 9219, no. c, pp. 1–1, 2015.

[122] NVIDIA Corporation, "NVIDIA GRID K1 and K2 Graphics-Accelerated Virtual Desktops and Applications," 2013.

[123] Q. Xie, T. Huang, Z. Zou, L. Xia, Y. Zhu, and J. Jiang, "An accurate power model for GPU processors," in *Proceedings - 2012 7th International Conference on Computing and Convergence Technology (ICCIT, ICEI and ICACT), ICCCT 2012*, 2012, pp. 1141–1146.

[124] D. Boughzala *et al.*, “Predicting the energy consumption of CUDA kernels using SimGrid,” in *SBAC-PAD-IEEE International Symposium on Computer Archi- tecture and High Performance ComputingInternational Symposium on Computer Archi- tecture and High Performance Computing*, pp. 1–8.

[125] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, “Versatile, scalable, and accurate simulation of distributed applications and platforms,” *J. Parallel Distrib. Comput.*, vol. 74, no. 10, pp. 2899–2917, 2014.

[126] A. C. B, A. Acosta, F. Almeida, and V. Blanco, “GPU Power Modeling of HPC Applications for the Simulation of Heterogeneous Clouds,” in *International Conference on Parallel Processing and Applied Mathematics*, 2017, pp. 91–101.

[127] K. M. Giannoutakis, A. T. Makaratzis, D. Tzovaras, C. K. Filelis-Papadopoulos, and G. A. Gravvanis, “On the power consumption modeling for the simulation of heterogeneous HPC clouds,” in *Proceedings of the 1st International Workshop on Next generation of Cloud Architectures*, 2017, pp. 1–6.

[128] S. Hong and H. Kim, “An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness,” in *ACM SIGARCH Computer Architecture News*, 2009, vol. 37, no. 3, p. 152.

[129] S. Baghsorkhi and M. Delahaye, “Analytical performance prediction for evaluation and tuning of GPGPU applications,” in *Workshop on EPHAM2009, in Conjunction with CGO, Citeseer*, 2009.

[130] K. Kothapalli, R. Mukherjee, M. Suhail Rehman, S. Patidar, P. J. Narayanan, and K. Srinathan, “A performance prediction model for the CUDA GPGPU platform,” in *16th International Conference on High Performance Computing, HiPC 2009 - Proceedings*, 2009, no. December, pp. 463–472.

[131] W. Jia, K. A. Shaw, and M. Martonosi, “Stargazer: Automated regression-based GPU design space exploration,” in *ISPASS 2012 - IEEE International Symposium on Performance Analysis of Systems and Software*, 2012, pp. 2–13.

[132] A. Karami, F. Khunjush, and S. A. Mirsoleimani, “A statistical performance analyzer framework for OpenCL kernels on Nvidia GPUs,” *J. Supercomput.*, vol. 71, no. 8, pp. 2900–2921, 2015.

[133] E. Konstantinidis and Y. Cotronis, “A practical Performance model for compute and memory bound GPU kernels,” in *Proceedings - 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2015*, 2015, pp. 651–658.

[134] M. H. Shafiabadi, H. Pedram, M. Reshadi, and A. Reza, “Comprehensive regression-based model to predict performance of general-purpose graphics processing unit,” *Cluster Comput.*, vol. 2, 2019.

[135] R. Ubal, P. Mistry, D. Schaa, H. Ave, and D. Kaeli, “Multi2Sim : A Simulation Framework for CPU-GPU Computing,” in *Proceedings of the PACT ’12*, 2012, pp. 335–344.

[136] X. Wang, K. Huang, A. Knoll, and X. Qian, “A hybrid framework for fast and accurate GPU performance estimation through source-level analysis and trace-based simulation,” in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019, pp. 506–518.

[137] Q. Wang and X. Chu, “GPGPU Performance Estimation with Core and Memory Frequency Scaling,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 12, pp. 2865–2881, 2020.

[138] J. J. Prevost, K. Nagothu, M. Jamshidi, and B. Kelley, “Optimal calculation overhead for energy efficient cloud workload prediction,” in *2014 World Automation Congress (WAC)*, 2014, pp. 741–747.

[139] E. Caron, F. Desprez, and A. Muresan, “Forecasting for grid and cloud computing on-demand resources based on pattern matching,” in *Proceedings - 2nd IEEE International Conference on Cloud Computing Technology and Science, CloudCom 2010*, 2010, pp. 456–463.

[140] Z. Gong, X. Gu, and J. Wilkes, “PRESS: PRedictive Elastic reSource Scaling for cloud systems,” in *Proceedings of the 2010 International Conference on Network and Service Management, CNSM 2010*, 2010, no. Vm, pp. 9–16.

[141] N. Roy, A. Dubey, and A. Gokhale, “Efficient autoscaling in the cloud using predictive models for workload forecasting,” in *Proceedings - 2011 IEEE 4th International Conference on Cloud Computing, CLOUD 2011*, 2011, pp. 500–507.

[142] A. Khan, X. Yan, S. Tao, and N. Anerousis, “Workload characterization and prediction in the cloud: A multiple time series approach,” in *Proceedings of the 2012 IEEE Network Operations and Management Symposium, NOMS 2012*, 2012, pp. 1287–1294.

[143] S. Islam, J. Keung, K. Lee, and A. Liu, “Empirical prediction models for adaptive resource provisioning in the cloud,” *Futur. Gener. Comput. Syst.*, vol. 28, no. 1, pp. 155–162, 2012.

[144] D. A. Menascé, “TPC-W: A benchmark for e-commerce,” *IEEE Internet Comput.*, vol. 6, no. 3, pp. 83–87, 2002.

[145] L. Zhang, Y. Zhang, P. Jamshidi, L. Xu, and C. Pahl, “Service workload patterns for Qos-driven cloud resource management,” *J. Cloud Comput.*, vol. 4, no. 1, pp. 1–21, 2015.

[146] J. Yang, C. Liu, Y. Shang, Z. Mao, and J. Chen, “Workload Predicting-Based Automatic Scaling in Service Clouds,” in *2013 IEEE Sixth International Conference on Cloud Computing*, 2013, pp. 810–815.

[147] W. Fang, Z. H. Lu, J. Wu, and Z. Y. Cao, “RPPS: A novel resource prediction and provisioning scheme in cloud data center,” in *Proceedings - 2012 IEEE 9th International Conference on Services Computing, SCC 2012*, 2012, pp. 609–616.

[148] Y. Han, J. Chan, and C. Leckie, “Analysing Virtual Machine Usage in Cloud Computing,” *Services (SERVICES), 2013 IEEE Ninth World Congress on*. pp. 370–377, 2013.

[149] Q. Huang, S. Su, S. Xu, J. Li, P. Xu, and K. Shuang, “Migration-based elastic consolidation scheduling in cloud data center,” in *Proceedings - International Conference on Distributed Computing Systems*, 2013, pp. 93–97.

[150] R. N. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya, “Workload prediction using ARIMA model and its impact on cloud applications’ QoS,” *IEEE Trans. Cloud Comput.*, vol. 3, no. 4, pp. 449–458, 2015.

[151] Q. Huang, K. Shuang, P. Xu, J. Li, X. Liu, and S. Su, “Prediction-based dynamic resource scheduling for virtualized cloud systems,” *J. Networks*, vol. 9, no. 2, pp. 375–383, 2014.

[152] Q. Zhang, H. Chen, and Z. Yin, “PRMRAP: A Proactive Virtual Resource Management Framework in Cloud,” in *Proceedings - 2017 IEEE 1st International Conference on Edge Computing, EDGE 2017*, 2017, pp. 120–127.

[153] F. Farahnakian, T. Pahikkala, P. Liljeberg, J. Plosila, N. T. Hieu, and H. Tenhunen, “Energy-aware VM consolidation in cloud data centers using utilization prediction model,” *IEEE Trans. Cloud Comput.*, vol. 7, no. 2, pp. 524–536, 2019.

[154] S. S. Wagle, M. Guzek, and P. Bouvry, “Service performance pattern analysis and prediction of commercially available cloud providers,” in *Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom*, 2016, vol. 0, pp. 26–34.

[155] A. A. Z. A. Ibrahim, “PRESEnCE: A Framework for Monitoring, Modelling and Evaluating the Performance of Cloud SaaS Web Services,” in *Proceedings - 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops, DSN-W 2018*, 2018, pp. 83–86.

[156] J. Wang, C. Huang, K. He, X. Wang, X. Chen, and K. Qin, “An Energy-Aware Resource Allocation Heuristics for VM Scheduling in Cloud,” in *High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference on*, 2013, pp. 587–594.

[157] F. Farahnakian, P. Liljeberg, and J. Plosila, “LiRCUP: Linear regression based CPU usage prediction algorithm for live migration of virtual machines in data centers,” in *Proceedings - 39th Euromicro Conference Series on Software Engineering and Advanced Applications, SEAA 2013*, 2013, no. June 2014, pp. 357–364.

[158] J. Subirats and J. Guitart, “Assessing and forecasting energy efficiency on Cloud computing platforms,” *Futur. Gener. Comput. Syst.*, vol. 45, pp. 70–94, 2015.

[159] J. V. Kistowski, M. Deffner, and S. Kounev, “Run-time prediction of power consumption for component deployments,” in *Proceedings - 15th IEEE International Conference on Autonomic Computing, ICAC 2018*, 2018, pp. 151–156.

[160] M. Aldossary, K. Djemame, I. Alzamil, A. Kostopoulos, A. Dimakis, and E.

Agiatzidou, “Energy-aware cost prediction and pricing of virtual machines in cloud computing environments,” *Futur. Gener. Comput. Syst.*, vol. 93, pp. 442–459, 2019.

[161] M. Xue *et al.*, “GScale: Scaling up GPU virtualization with dynamic sharing of graphics memory space,” in *Proceedings of the 2016 USENIX Annual Technical Conference, USENIX ATC 2016*, 2016, pp. 579–590.

[162] X. Zhao, J. Yao, P. Gao, and H. Guan, “Efficient sharing and fine-grained scheduling of virtualized GPU resources,” in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, 2018, vol. 2018-July, pp. 742–752.

[163] “Intel® Graphics Virtualization Technology (Intel® GVT).” [Online]. Available: https://01.org/zh/igvt-g. [Accessed: 13-Oct-2020].

[164] Q. Lu, J. Yao, H. Guan, and P. Gao, “GQoS: A QoS-Oriented GPU Virtualization with Adaptive Capacity Sharing,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 4, pp. 843–855, 2020.

[165] V. Gupta, K. Schwan, N. Tolia, V. Talwar, and P. Ranganathan, “Pegasus: Coordinated Scheduling for Virtualized Accelerator-Based Systems,” in *2011 USENIX Annual Technical Conference (USENIX ATC’11)*, 2011, p. 31.

[166] “ICE financial application.” [Online]. Available: https://www.theice.com/technology/instant-message. [Accessed: 26-May-2016].

[167] G. Giunta, R. Montella, G. Agrillo, and G. Coviello, “A GPGPU transparent virtualization component for high performance computing clouds,” in *In Euro-Par 2010-Parallel Processing*, 2010, pp. 379–391.

[168] S. Srinivasan, S. Krishnamoorthy, and P. Sadayappan, “A Robust Scheduling Strategy for Moldable Scheduling of Parallel Jobs . A Robust Scheduling Strategy for Moldable Scheduling of Parallel Jobs,” 2016, no. JANUARY 2003.

[169] H. Yadav and B. Annappa, “Adaptive GPU resource scheduling on virtualized servers in cloud gaming,” in *2017 Conference on Information and Communication Technology, CICT 2017*, 2018, vol. 2018-April, no. i, pp. 1–6.

[170] H. E. Dinaki, S. Shirmohammadi, and M. R. Hashemi, “Boosted Metaheuristic Algorithms for QoE-Aware Server Selection in Multiplayer Cloud Gaming,” *IEEE Access*, vol. 8, no. July 2016, pp. 60468–60483, 2020.

[171] S. Kato, M. Mcthrow, C. Maltzahn, and S. Brandt, “Gdev : First-Class GPU Resource Management in the Operating System,” in *USENIX Annual Technical Conference (USENIX ATC)*, 2012, pp. 401–412.

[172] C. H. Hong, I. Spence, and D. S. Nikolopoulos, “FairGV: Fair and fast GPU virtualization,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 12, pp. 3472–3485, 2017.

[173] Y. Hu, S. Rallapalli, B. Ko, and R. Govindan, “Olympian: Scheduling GPU usage in a deep neural network model serving system,” in *Proceedings of the 19th International Middleware Conference*, 2018, pp. 53–65.

[174] “Tensorflow website,” 2019. [Online]. Available: https://www.tensorflow.org/. [Accessed: 16-Oct-2020].

[175] S. Ilager, R. Wankar, R. Kune, and R. Buyya, “GPU PaaS Computation Model in Aneka Cloud Computing Environments,” in *Smart Data: State-of-the-Art Perspectives in Computing and Applications*, 2019, p. 19.

[176] C. Vecchiola, X. Chu, and R. Buyya, “Aneka: A software platform for.NET based cloud compputing,” *High Speed Large Scale Sci. Comput.*, vol. 18, pp. 267–295, 2009.

[177] L. Yu, “Multilevel Interference-aware Scheduling on Modern GPUs,” 2019.

[178] C. Zhang, J. Yao, Z. Qi, M. Yu, and H. Guan, “vGASA: Adaptive Scheduling Algorithm of Virtualized GPU Resource in Cloud Gaming,” *Parallel Distrib. Syst. IEEE Trans.*, vol. 25, no. 11, pp. 3036–3045, 2014.

[179] “VMware Horizon,” *VMware*. [Online]. Available: https://www.vmware.com/products/horizon.html. [Accessed: 20-Oct-2020].

[180] “What is vSphere?” [Online]. Available: https://www.vmware.com/products/vsphere.html. [Accessed: 20-Oct-2020].

[181] A. Guleria, J. Lakshmi, and C. Padala, “QuADD: QUantifying accelerator disaggregated datacenter efficiency,” in *IEEE International Conference on Cloud Computing, CLOUD*, 2019, vol. 2019-July, pp. 349–357.

[182] C. S. Li, H. Franke, C. Parris, B. Abali, M. Kesavan, and V. Chang, “Composable architecture for rack scale big data computing,” *Futur. Gener. Comput. Syst.*, vol. 67, pp. 180–193, 2017.

[183] F. Grid, “Grid5000 : Home.” [Online]. Available: https://www.grid5000.fr/w/Grid5000:Home. [Accessed: 22-Oct-2020].

[184] X. Zhang, J. Lu, and X. Qin, “BFEPM: Best fit energy prediction modeling based on CPU utilization,” in *Proceedings - 2013 IEEE 8th International Conference on Networking, Architecture and Storage, NAS 2013*, 2013, pp. 41–49.

[185] D. Economou, S. Rivoire, C. Kozyrakis, and P. Ranganathan, “Full-System Power Analysis and Modeling for Server Environments,” in *Workshop on Modeling, Benchmarking and Simulation (MoBS)*, 2006, no. 3, pp. 807–812.

[186] M. Armbrust *et al.*, “A view of cloud computing,” *Commun. ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[187] J. Ricardo, P. Orellana, and A. B. Caminero, “Proposals for Efficient Management of FPGAs within Cloud Computing Environments,” 2017.

[188] H. J. Choi, D. O. Son, S. G. Kang, J. M. Kim, H.-H. Lee, and C. H. Kim, “An efficient scheduling scheme using estimated execution time for heterogeneous computing systems,” *J. Supercomput.*, vol. 65, no. 2, pp. 886–902, 2013.

[189] C.-T. Yang, J.-C. Liu, H.-Y. Wang, and C.-H. Hsu, “Implementation of GPU Virtualization Using PCI Pass-through Mechanism,” *J. Supercomput.*, vol. 68, no. 1, pp. 183–213, 2014.

[190] IBM, “An architectural blueprint for autonomic computing,” 2006.

[191] “CUDA Occupancy Calculator - Nvidia.” [Online]. Available: developer.download.nvidia.com/compute/cuda/CUDA_Occupancy_calculator.xls. [Accessed: 23-Oct-2017].

[192] S. Mittal and J. S. Vetter, “A Survey of Methods For Analyzing and Improving GPU Energy Efficiency,” *ACM Comput. Surv.*, vol. 47, no. 2, pp. 1–23, 2014.

[193] “nvidia-smi.” [Online]. Available: http://developer.download.nvidia.com/compute/DCGM/docs/nvidia-smi-367.38.pdf. [Accessed: 20-Oct-2017].

[194] “Profiler User’s Guide.” [Online]. Available: http://docs.nvidia.com/cuda/profiler-users-guide/index.html#gpu-trace-and-api-trace-modes. [Accessed: 20-Oct-2017].

[195] S. Che *et al.*, “Rodinia: A Benchmark Suite for Heterogeneous Computing,” in *IEEE International Symposium on Workload Characterization*, 2009, pp. 44–54.

[196] J. a. Stratton *et al.*, “Parboil: A Revised Benchmark Suite for Scientific and Commercial Throughput Computing,” *Cent. Reliab. High-Performance Comput.*, p. 127, 2012.

[197] “CUDA Code Samples | NVIDIA Developer.” [Online]. Available: https://developer.nvidia.com/cuda-code-samples. [Accessed: 23-May-2016].

[198] “Profiler User’s Guide.” [Online]. Available: https://docs.nvidia.com/cuda/profiler-users-guide/index.html. [Accessed: 21-Jan-2020].

[199] “CUDA PROFILER USER’S GUIDE.” [Online]. Available: https://wrf.ecse.rpi.edu/wiki/ParallelComputingSpring2014/nvidia/cuda6doc/pdf/CUDA_Profiler_Users_Guide.pdf. [Accessed: 21-Jan-2020].

[200] M. Curtis-Maury, A. Shah, F. Blagojevic, D. S. Nikolopoulos, B. R. De Supinski, and M. Schulz, “Prediction models for multi-dimensional power-performance optimization on many cores,” in *Parallel Architectures and Compilation Techniques - Conference Proceedings, PACT*, 2008, no. 2008, pp. 250–259.

[201] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, “Effective straggler mitigation: Attack of the Clones,” in *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2013*, 2013, pp. 185–198.

[202] M. Zaharia, A. Konwinski, A. Joseph, R. Katz, and I Stoica, “Improving MapReduce performance in heterogeneous environments.,” *OSDI*, vol. 8, no. 4, p. 7, 2008.

[203] D. Fall, T. Okuda, Y. Kadobayashi, and S. Yamaguchi, “Risk adaptive authorization mechanism (RAdAM) for cloud computing,” *J. Inf. Process.*, vol. 24, no. 2, pp. 371–380, 2016.

[204] G. Shao and J. Chen, “A load balancing strategy based on data correlation in

cloud computing," in *Proceedings - 9th IEEE/ACM International Conference on Utility and Cloud Computing, UCC 2016*, 2016, pp. 364–368.

[205] S. Lek, M. Delacoste, P. Baran, I. Dimopoulos, J. Lauga, and S. Aulagnier, "Application of neural networks to modelling nonlinear relationships in ecology," *Ecol. Modell.*, vol. 90, no. 1, pp. 39–52, 1996.

[206] S. Mukhopadhyay and S. Mukhopadhyay, *Deep Learning and Neural Networks*. 2015.

[207] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Adv. Neural Inf. Process. Syst.*, pp. 1–9, 2012.

[208] J. Schmidt-Hieber, "Nonparametric regression using deep neural networks with ReLU activation function," 2017.

[209] R. G. Mantovani, A. L. D. Rossi, J. Vanschoren, B. Bischl, and A. C. P. L. F. Carvalho, "To tune or not to tune: Recommending when to adjust SVM hyper-parameters via meta-learning," in *Proceedings of the International Joint Conference on Neural Networks*, 2015, pp. 1–8.

[210] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," *Adv. Neural Inf. Process. Syst.*, pp. 1–9, 2011.

[211] K. Greff, R. K. Srivastava, J. Koutnik, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A Search Space Odyssey," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 28, no. 10, pp. 2222–2232, 2017.

[212] M. Borovcnik, H.-J. Bentz, and R. Kapadia, *Machine Learning:A Probabilistic Perspective*. MIT press, 2012.

[213] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[214] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *the thirteenth international conference on artificial intelligence and statistics*, 2010, vol. 9, pp. 249–256.

[215] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv Prepr. arXiv*, vol. 1412.6980, 2014.

[216] J. C. Duchi, P. L. Bartlett, and M. J. Wainwright, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization," *J. ofMachine Learn. Res.*, vol. 12, pp. 2121–2159, 2011.

[217] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," *ACM SIGARCH Comput. Archit. News*, vol. 35, no. 2, p. 13, 2007.

[218] F. Chollet, "Home - Keras Documentation," 2015. [Online]. Available: https://keras.io/. [Accessed: 27-Feb-2020].

[219] Z. Liu *et al.*, "Renewable and cooling aware workload management for sustainable data centers," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 40, no. 1, pp. 175–186, 2012.

[220] S. S. Gill and R. Buyya, “Sustainable Cloud Computing Realization for Different Applications: A Manifesto,” *Digit. Bus.*, pp. 95–117, 2019.

[221] J. V. Kistowski, H. Block, J. Beckett, K. D. Lange, J. A. Arnold, and S. Kounev, “Analysis of the influences on server power consumption and energy efficiency for CPU-intensive workloads,” in *ICPE 2015 - Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, 2015, pp. 223–234.

[222] N. Bobroff, A. Kochut, and K. Beaty, “Placement of Virtual Machines for Managing SLA Violations,” in *In 2007 10th IFIP/IEEE International Symposium on Integrated Network Management*, 2007, vol. 5, pp. 119–128.

[223] S.-Y. Jing, S. Ali, K. She, and Y. Zhong, “State-of-the-art research study for green cloud computing,” *J. Supercomput.*, vol. 65, no. 1, pp. 445–468, 2013.

[224] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani, “An approach for QoS-aware service composition on algorithms,” in *GECCO 2005 - Genetic and Evolutionary Computation Conference*, 2005, pp. 1069–1075.

[225] R. W. Hall, *Queueing methods: for services and manufacturing*. Pearson College Div, 1991.

[226] V. Sundarapandian, *Probability, Statistics and Queueing Theory*. PHI Learning, 2009.

[227] B. Wickremasinghe, R. N. Calheiros, and R. Buyya, “CloudAnalyst: A cloudsim-based visual modeller for analysing cloud computing environments and applications,” in *Proceedings - International Conference on Advanced Information Networking and Applications, AINA*, 2010, pp. 446–452.

[228] A. NNúñezez, J. L. Vázquez-Poletti, A. C. Caminero, G. G. Castañé, J. Carretero, and I. M. Llorente, “ICanCloud: A Flexible and Scalable Cloud Infrastructure Simulator,” *J. Grid Comput.*, vol. 10, no. 1, pp. 185–209, 2012.

[229] S. K. Garg and R. Buyya, “NetworkCloudSim: Modelling parallel applications in cloud simulations,” in *Proceedings - 2011 4th IEEE International Conference on Utility and Cloud Computing, UCC 2011*, 2011, pp. 105–113.

[230] Calheiros, Rodrigo N., Netto, Marco a.S., De Rose, César a.F., and R. Buyya, “EMUSIM: an integrated emulation and simulation environment for modeling, evaluation, and validation of performance of cloud computing applications,” *Softw. - Pract. Exp.*, vol. 43, no. 5, pp. 595–612, 2012.

[231] L. Liu *et al.*, “GreenCloud: A New Architecture for Green Data Center,” in *Proceedings of the 6th International Conference Industry Session on Autonomic Computing and Communications Industry Session*, 2009, pp. 29–38.

[232] I. Sriram, “SPECI, a simulation tool exploring cloud-scale data centres,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 5931 LNCS, pp. 381–392, 2009.

[233] S. Collange, M. Daumas, D. Defour, and D. Parello, “Barra: A parallel functional simulator for GPGPU,” in *Proceedings - 18th Annual IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer*

and Telecommunication Systems, MASCOTS 2010, 2010, pp. 351–360.

[234] G. F. Diamos, A. R. Kerr, S. Yalamanchili, and N. Clark, “Ocelot: a Dynamic Optimization framework for bulk-synchronous applications in heterogeneous systems,” in *Proceedings of the 19th international conference on Parallel architectures and compilation techniques - PACT ’10*, 2010, p. 353.

[235] V. Zakharenko, “FusionSim : Characterizing the Performance Benefits of Fused CPU / GPU Systems FusionSim : Characterizing the Performance Benefits of Fused CPU / GPU Systems,” 2012.

[236] Y. Kessaci *et al.*, “A Pareto-based Metaheuristic for Scheduling HPC Applications on a Geographically Distributed Cloud Federation,” 2012.