Power Analysis and Prediction for Heterogeneous Computation

Bishwajit Dutta

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Engineering

Wu-chun Feng, Chair
Dongyoon Lee
Haibo Zeng

December 12, 2017
Blacksburg, Virginia

Keywords: GPU, Power and Performance, Machine Learning

Power Analysis and Prediction for Heterogeneous Computation

Bishwajit Dutta

ABSTRACT

Power, performance, and cost dictate the procurement and operation of high-performance computing (HPC) systems. These systems use graphics processing units (GPUs) for performance boost. In order to identify inexpensive-to-acquire and inexpensive-to-operate systems, it is important to do a systematic comparison of such systems with respect to power, performance and energy characteristics with the end use applications. Additionally, the chosen systems must often achieve performance objectives without exceeding their respective power budgets, a task that is usually borne by a software-based power management system. Accurately predicting the power consumption of an application at different DVFS levels (or more generally, different processor configurations) is paramount for the efficient functioning of such a management system.

This thesis intends to apply the latest in the state-of-the-art in green computing research to optimize the total cost of acquisition and ownership of heterogeneous computing systems. To achieve this we take a two-fold approach. First, we explore the issue of *greener* device selection by characterizing device power and performance. For this, we explore previously untapped opportunities arising from a special type of graphics processor — the low-power integrated GPU — which is commonly available in commodity systems. We compare the *greenness* (power, energy, and energy-delay product $\rightarrow$ EDP) of the integrated GPU against a CPU running at different frequencies for the specific application domain of scientific visualization. Second, we explore the problem of predicting the power consumption of a GPU at different DVFS states via machine-learning techniques. Specifically, we perform statistically rigorous experiments to uncover the strengths and weaknesses of eight different machine-learning techniques (namely, *ZeroR*, *simple linear regression*, *KNN*, *bagging*, *random forest*, *SMO regression*, *decision tree*, and *neural networks*) in predicting GPU power consumption at different frequencies. Our study shows that a support vector machine-aided regression model (i.e., *SMO regression*) achieves the highest accuracy with a mean absolute error (MAE) of 4.5%. We also observe that the *random forest* method produces the most consistent results with a reasonable overall MAE of 7.4%. Our results also show that different models operate best in distinct regions of the application space. We, therefore, develop a novel, ensemble technique drawing the best characteristics of the various algorithms, which reduces the MAE to 3.5% and maximum error to 11% from 20% for *SMO regression*.

Power Analysis and Prediction for Heterogeneous Computation

Bishwajit Dutta

GENERAL AUDIENCE ABSTRACT

High-performance computing (HPC) systems or supercomputers, and data centers consume immense amount of power. Power consumption of a supercomputer generally exceeds the capacity of a small power plant. In fact, turning on a supercomputer caused a brown-out in a neighboring town once. These systems increasingly use graphics processing units (GPUs) for performance boost. For example the November 2017 top 500 supercomputers ranking has 101 GPU accelerated systems. Power, performance, and cost dictate the procurement and operation of these systems. In this thesis we intend to apply the latest in the state-of-the-art in green computing research to optimize the total cost of acquisition and ownership of heterogeneous CPU-GPU computing systems.

In order to identify inexpensive-to-acquire and inexpensive-to-operate systems, it is important to do a systematic comparison of such systems with respect to power, performance and energy characteristics with the end user applications. Additionally, the chosen systems must often achieve performance objectives without exceeding their respective power budgets, a task that is usually borne by a software-based power management system. Accurately predicting the power consumption of an application at different processor configurations is paramount for the efficient functioning of such a management system.

To achieve this we take a two-fold approach. First, we explore the issue of *greener* device selection by characterizing device power and performance. For this, we explore previously untapped opportunities arising from a special type of graphics processor — the low-power integrated GPU — which is commonly available in commodity systems. We compare the *greenness* (power, energy, and energy-delay product $\rightarrow$ EDP) of the integrated GPU against a CPU running at different frequencies for the specific application domain of scientific visualization. Second, we explore the problem of predicting the power consumption of a GPU at different configurations via machine-learning techniques. Specifically, we perform statistically rigorous experiments to uncover the strengths and weaknesses of different machine-learning techniques in predicting GPU power consumption at different frequencies. Our study shows that a support vector machine-aided regression model which fits a complex curve achieves the highest accuracy with a mean absolute error of 4.5%. Our results also show that different models operate best in distinct regions of the application space. We, therefore, develop a novel, ensemble technique drawing the best characteristics of the various algorithms, which reduces the MAE to 3.5% and maximum error to 11% from 20%.

Acknowledgments

I would like to use this page to express my deepest gratitude to Dr. Wu-chun Feng, my research advisor and committee chair for his constant guidance and useful remarks on my research and thesis work. His deep knowledge, personality and his gentle way of guidance have and will remain a source of inspiration for me.

Furthermore, I want to express my heartfelt gratitude to my collaborator Vignesh Adhinarayanan for giving directions and innumerable help throughout my research and thesis work. His tireless work ethic has inspired me to do the best possible research.

I am grateful to Dr. Dongyoon Lee and Dr. Haibo Zhang for being my thesis committee members.

I want to thank Dr. Mark Gardener and other friends and colleagues for all the help in getting me settled down and feel comfortable in the Virginia Tech SyNeRGy lab.

Lastly, I can never thank enough my wife Sudeshna Podder for motivating me to pursue a master degree. Her love and tremendous support made life comfortable for me as I worked to complete this degree.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Energy efficiency and performance are top design criteria for designing high-performance computing (HPC) systems. These systems consume a significant amount of power and energy. For instance, with today's technology, building an exascale machine would consume up to 350 megawatts (MW) which is sufficient to power 250,000 houses [24]. Increasingly these systems are reliant on GPUs [3] for applications such as scientific visualization. This trend is shown in figure 1.1. The latest November 2017 ranking has 101 GPU accelerated system with 86 of them being from NVIDIA [3].

Apart from the problem of the cost of power or energy, there is the problem of exceeding power budgets and corresponding heating up of systems. The increasing prevalence of *dark*

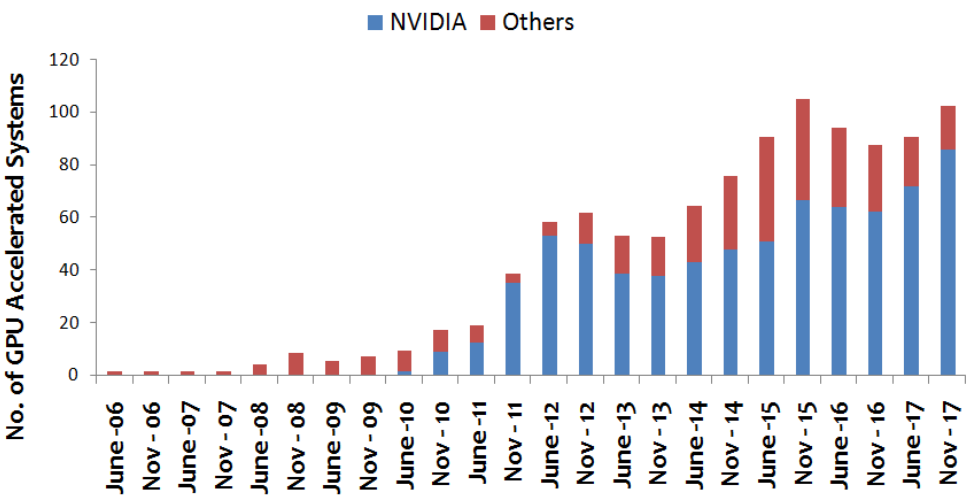


Figure 1.1: Top 500 Supercomputers as per Top500.org's November 2017 Rankings

silicon[1] [18] and the emerging *hardware-overprovisioned* supercomputers[2] [37] have made it harder to safely operate these systems under their respective power budgets. This has necessitated the introduction of power *management* systems such as Intel's Running Average Power Limit (RAPL) [16] and several research prototypes [19, 32, 14, 43] which are capable of enforcing strict *power caps*. Central to the functioning of such a power management system is the ability to predict the power consumption of an application at different processor configurations (usually meaning different operating frequencies or DVFS states, but not necessarily limited to it). The idea is to configure the system to its "highest" configuration point where it is guaranteed that the *power caps* are not violated.

To make these systems energy-efficient, deliver high performance, and operate within power

[1]In a processor, the sheer number of transistors can result in a situation where the processor can draw more power than what can be safely dissipated as heat. This means several transistors must remain off or operate at a low-power mode resulting in "dark" areas on the silicon

[2]In an overprovisioned supercomputer, more compute nodes are added than what the nameplate capacity can support

budgets, we must not only be able to select proper processing elements but also be able to predict power and performance of different applications at different hardware configurations. Device selection involves evaluating their performance, power, and energy characteristics when running frequently-executed applications.

Firstly In this work, we study power and performance characteristics with one such class of applications, namely, scientific visualization. We use the EDP metric to compare the systems and visualization tasks for the combined criteria of dynamic energy and performance. Additionally, for overall system cost reduction, it is important to explore commodity components such as integrated GPUs.

The task of choosing between a CPU and a GPU for running scientific visualization workloads may seem straightforward, as GPUs are specifically designed for graphics processing. However, if cost is an issue, datacenters may restrict (or reduce) the availability of discrete GPUs for application scientists. In the era of post-processing visualization, where the simulation runs on a cluster and the visualization takes place on a different (and separate) node, this may not be an issue. In this case, only the visualization node needed graphics processing capabilities, thus using a more expensive discrete GPU for it is a viable and cost-effective option. For in-situ visualization, the visualization task runs on the same nodes as the simulation. Thus, graphics processing capabilities must be provided on *each* node. In such a situation, the system designer and the users have three options: (i) use CPUs that are optimized for graphics processing [2] for both simulation and visualization, (ii) deploy an expensive discrete GPU into *each* node, or (iii) use smaller integrated GPUs that are

soldered onto the motherboard. In this thesis, we explore the first and third options.

The motherboard-integrated GPU has a few drawbacks. It only has a few MB of RAM of its own and relies primarily on system memory instead. Many of these integrated GPUs, including the Matrox G200 GPU used in our study, do not expose dynamic voltage and frequency scaling (DVFS) settings to the user, and therefore, their operating frequency cannot be tuned to optimize for power and energy. However, they can provide a few benefits. First, they operate at a fraction of the power envelope than other devices do. Second, they are significantly cheaper. While the power savings can be significant for this integrated CPU device, their impact on other greenness metrics such as energy or the energy-delay product (EDP) is not clearly understood. Therefore, we systematically study the *greenness* of the integrated GPU for scientific visualization tasks and compare it against CPUs operating at different frequencies. Ultimately, this work benefits the visualization scientists in making greener and less expensive simulations in the context of high-performance computing (HPC). A similar study can be done for other application domains.

Second, in this thesis, we address the problem of power prediction and we take a machine learning based approach to study the same. Although several models for DVFS-based power prediction have been previously proposed [17]. Recent interest and advances in machine-learning techniques (including neural networks) has necessitated a re-examination of data-driven modeling techniques in many areas of computing, including system design. As such, we investigate the applicability of several machine-learning techniques in predicting the power consumed by a GPU at different voltage-frequency settings (or P-states).

We address the problem of predicting the power consumption of an application at different system configurations (namely, DVFS states). Accurately predicting the power consumption of a computing system is useful in several scenarios, some of which are described below.

Use Cases

Power Capping. Modern computing systems can draw more power than they can safely sustain. At the node level, the processor may end up consuming more power than its rated thermal design power (TDP) for some applications [18]. At the supercomputer level, machines may be built which can draw more power than its nameplate capacity (i.e., hardware overprovisioned) [37].

To illustrate the need for power capping, we present the power profile of an application on a reference processor operating at three different frequencies f_1, f_2, and f_3 in Figure 1.2. Note that this application consists of three distinct phases: a low power phase which is followed by a high power phase which in turn is followed by another low power phase. Let's assume that a power cap of 120W is enforced on this system. A conservative approach would set the machine at f_2 (or lower) where it can be guaranteed that the power cap is never violated. However, if a power predictor is able to accurately predict the power consumption at different frequencies, the power management system may choose to operate at f_1 for phase I, f_2 for phase II, and back to f_1 for phase III.

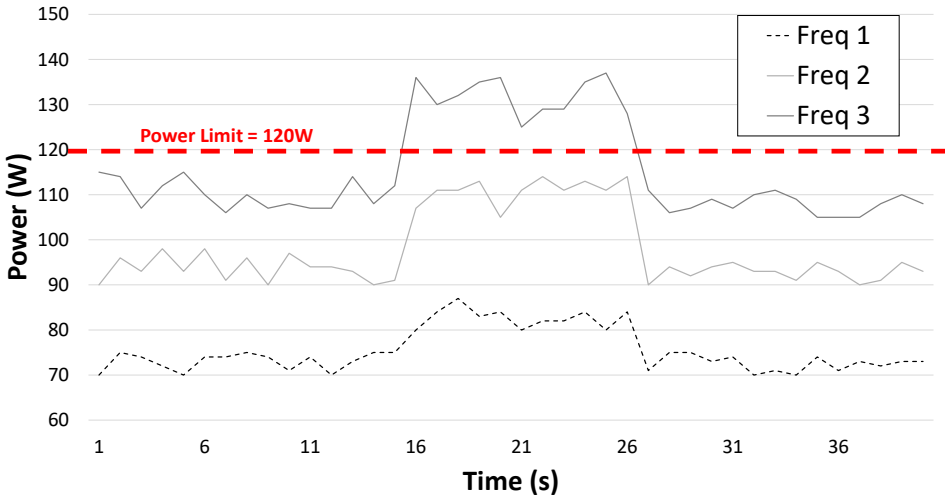


Figure 1.2: Power consumption of an application at different frequencies

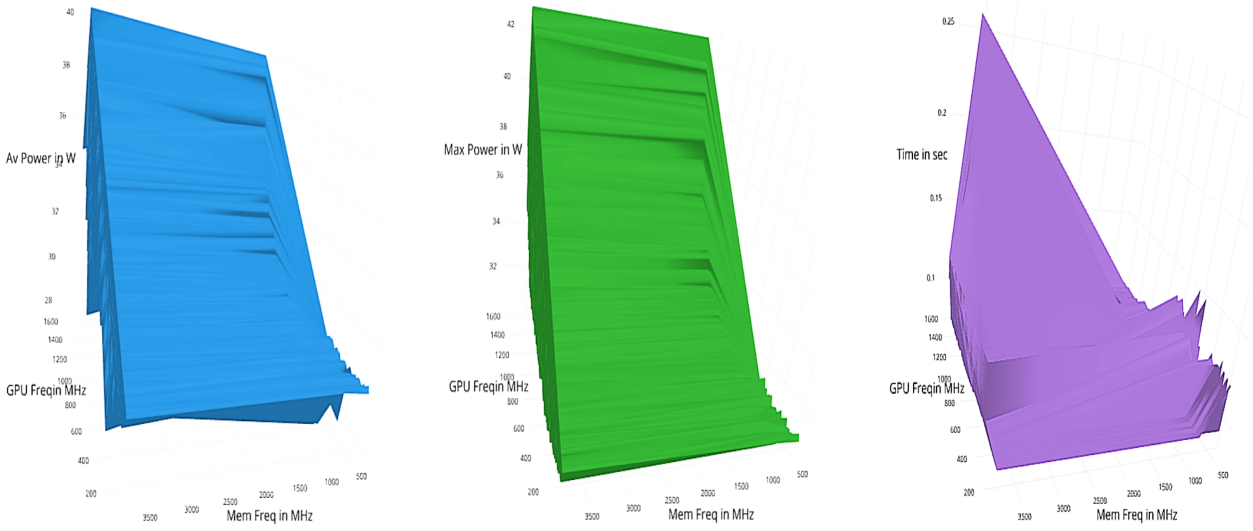
Energy-Performance Trade-off Analysis. The energy cost of operating today's high-end computing systems is around 25% of their acquisition cost [23]. For computing systems built with older technology components, the energy cost may even exceed the operating cost. Reliable prediction of power consumption and performance can help in judicious trade-offs between power and energy to keep the total cost of operation (TCO) low. An alternative approach to operating an application at an ideal frequency is to exhaustively explore the configuration space. However, with numerous configuration options available in today's computing system (e.g., different DVFS knobs for the different components of a processor), the energy cost of finding the correct configuration itself could exceed the energy cost of running the application. Therefore, predicting the power consumption at different configuration points automatically via modeling becomes indispensable.

Machines without Built-in Power Meters. Even in scenarios where exhaustive exploration is viable (which can happen in a mission-oriented computing system where only a handful of applications are run), a model-based exploration is useful in many cases. Let's consider the example of a popular GPU platform used in computer vision – Nvidia's Tegra TK1. The TK1 platform lacks in-built GPU power meters (so is the case with several consumer-grade GPUs). Direct techniques to measure the GPU power consumption on these platforms can be intrusive and time-consuming. On the other hand, a model-based approach requires instrumentation only once for collecting the training data set after which commonly available profiling tools such as `nvprof` can be used to estimate the power consumption.

Design space exploration: The ability to accurately estimate power and performance at different hardware configurations can help in faster design space exploration. An example is estimating the power consumption when level 2 cache size is doubled (and hence utilization level becomes half as before, for an application) while keeping all other architectural aspects same. Low-level power estimation techniques tend to be more accurate but are generally slow enough for fast design space pruning. Therefore research into off-line high-level prediction techniques can lead to cheaper and quicker design space exploration.

Challenges in Power Prediction

Previous works that attempted to model GPU power consumption at different DVFS states via machine learning operated at the GPU kernel level [42]. While there are several differ-



(a) Av. power scaling surface (b) Max power scaling surface (c) Performance scaling surface

Figure 1.3: Scaling surfaces for the SHOC Triad application

ences in our approach (including the parameters used for modeling, the breadth of techniques, and the target metrics modeled), at the problem formulation-level there is a fundamental difference which makes our modeling task more challenging. While previous works attempted to predict the power of individual GPU kernels, we attempt to predict the characteristics of the application as a whole. The individual kernels can have irregular scaling properties with respect to frequencies [42]. This scaling behavior gets more complex when an application has several kernels each of which is scheduled in parallel and have different computational and memory bounds.

For instance, Figure 1.3 shows the power and performance scaling surface for the triad application from the SHOC benchmark suite. We observe that the power and performance scaling is not linearly dependent on frequencies but is a complex interaction of frequencies and resources utilization levels. Therefore, it becomes harder to find line, curves, or models

which fit the data. This is one of the reasons that warrant a new look at the GPU power prediction problem.

1.2 Related Work

1.2.1 Power Analysis

The problem of improving the *greenness* of scientific visualization pipelines has received recent attention due to the increasing energy cost of data movement [40]. The solutions previously explored can be classified into four categories.

In-situ Techniques: These techniques reduce data movement by processing most of the data when it resides in memory rather than allowing it to *move* to the storage subsystem. Specific examples include image-based approach to scientific visualization [4], in-situ sampling [6], in-situ analytics [21], and data compression [12].

Work Distribution: The basic premise here is to carefully redistribute the simulation and visualization tasks across the various resources (e.g., nodes or cores) to minimize data movement [39]. For instance, the two tasks may be distributed to dedicated nodes or cores or the same set of resources may be used for the two tasks, but in a time-shared manner which affects intra and inter-node data movement and hence affects energy.

Memory Optimizations: New types of memory such as NVRAM exhibit lower data access energy. Some research papers have looked at optimizing the memory hierarchy to reduce the

energy consumption of scientific visualization [22, 25].

DVFS-based Techniques: The frequency of the cores and the other components of a compute node are varied to tune power and performance. Labasan et al. have looked at both DVFS [30] and techniques built atop DVFS such as power sloshing [31].

Unlike the above works, we explore the *underutilized* compute components (namely, integrated GPU) inside a node and compare its efficacy with classical green computing techniques such as DVFS.

1.2.2 Power Prediction

Table 1.1: Related Work in Power Prediction

Authors	**Device**	**Analysis Type**	**Where**	**Technique**	**Error**
G. Wu et al. [42]	GPU	Prediction	Offline	ML	10% Max
V. Adhinarayanan et al. [7]	GPU	Estimation	Online	Statistical	6% MAE
B. Subramaniam et al. [41]	CPU	Prediction	Offline	Statistical	5.6% MAE
W. Jia et al. [28]	GPU	Analysis	Offline	Statistical	-
This work	GPU	Prediction	Offline	ML	3.5% MAE, 11% Max

Various statistical and machine learning techniques have been explored in the past for power prediction. A survey by Eeckhout on power and performance evaluation [17] gives an excellent overview but these are mostly focused on CPUs. GPU power and performance modeling techniques range from low-level modeling to statistical and machine learning based methods. Table 1.1 summarizes some of the closely related work and their prediction accuracy as compared to ours.

Wenhao Jia et al. [28] proposed the starchart optimization technique which relies on the experimental determination of power and performance of the whole design space with all possible combinations of system parameter settings. A similar approach is taken in Stargazer [27]. We study multiple approaches for power prediction. In our study, we predict power consumption of a new application based only on its hardware resource utilization metrics.

Hsu et al. [26] have proposed a power-aware algorithm which achieves power reduction by automatically adapting voltage and frequency settings. Bailey et al. [10] have examined multivariate regression and clustering techniques. Subramaniam et al. [41] and Ma et al. [34] have explored regression-based techniques for power and performance modeling. However, these are with non- machine learning based techniques and are evaluated either on CPUs or earlier generations of the GPU which was simpler. The prediction MAE of the statistical technique by Subramaniam et al. [41] was 5.6% compared to 4.5% in our case.

Adhinarayanan et al. [7] constructed on-line power model with MAE of 6% based on correlation analysis and built-in GPU power meter. Unlike their work, we make off-line predictions for a new application for *different DVFS settings*. Luo and Suda [33] and Baghsorkhi et al. [9] use memory instruction analysis and workflow graphs to estimate performance and energy but our technique is based on hardware resource utilization levels.

Arian et al. [35] have explored the area of GPU benchmarking and gave suggestions on optimizing the power and performance but it was mostly based on evolutionary algorithms.

Gene Wu et al. [42] used K-means algorithm and neural network to predict power and

performance from a given set of parameters. But this was at the kernel level with a maximum error of 10% and our goal is a prediction for the whole application which can consist of several kernels. Additionally, our goal is to predict power without measuring the power consumption at a base frequency.

Xinxin et al. [36] have captured the application of existing DVFS based techniques to GPU power optimization. However, we focus on DVFS as well as hardware resource utilization levels for predicting GPU power consumption.

1.3 Contributions

Power and Performance Analysis

For power and performance analysis we systematically study the *greenness* of the integrated GPU for a specific application domain i.e. scientific visualization tasks and compare it against CPUs operating at different frequencies. This work benefits the visualization scientists in making greener and less expensive simulations in the context of high-performance computing (HPC).

Our major contributions and findings in this area can be summarized as follows:

- We perform experiments and provide data to facilitate a direct comparison between the CPU and integrated GPU with respect to performance, power, energy, and EDP for multiple visualization filters.

- We present our findings that highlight the scenarios in which the integrated GPU performs better than the CPU and vice versa. For larger data sets and complex visualization operations, the integrated GPU delivers significant benefits (up to a 26% better EDP) whereas for smaller data sets and relatively simpler visualization tasks the CPU is a better choice.

- We run the visualization filters on the CPU at different frequencies to help the CPU deliver better characteristics. We identify cases where the DVFS technique helped and where it turned out to be counter-productive.

Power Prediction

Our contributions on GPU power prediction in this thesis include the following:

- **Accurate power prediction of a GPU at different DVFS states.** We explore several prediction techniques including *ZeroR*, *simple linear regression*, *SMO regression*, *bagging*, *random forests*, *decision trees*, *KNN*, and *neural networks* for predicting the GPU's power consumption at different DVFS states. To the best of our knowledge, the breadth of this study is wider than any other power prediction study including those carried out for CPUs.

- **Statistically rigorous comparison of different machine-learning techniques**. Unlike previous studies that *only* compared the *mean error* of a few modeling tech-

niques, we employ *Tukey's HSD* test for comparing multiple approaches in a statistically rigorous manner at once.

- **Design of an ensemble approach for GPU power prediction**. Based on our observations we create different ensemble designs and evaluate them to make the best use of the relative strengths of each technique. While such ensemble models are popular in other domain areas, we are among the first to use it for DVFS prediction.

Our statistically rigorous experiments have resulted in several findings. Some of our major findings are highlighted below.

- *SMO regression*, *simple linear regression*, *KNN*, and *REPTree* can be statistically verified to have a better prediction accuracy than the baseline *ZeroR* method. While the other methods (excluding *neural networks*) exhibit a lower mean error than the baseline, these results were found to be statistically insignificant.

- *Neural networks* consistently shows lower accuracy than the other methods. We hypothesize that this is due to the few data points being used in the training set used for modeling. *C*omposite methods such as random forests, which combine results from several regression trees, produce consistent results (i.e., about the same error for different configurations). Therefore, we believe these methods are more appropriate for power management systems that are required to provide guarantees.

- Our ensemble method, which combines the results of *SMO regression*, *simple linear*

regression, and *REPTree* shows the best accuracy. The average error for this method is 3.5% with a maximum error of 11%.

1.4 Document Overview

The rest of the thesis is organized as follows. Chapter 2 presents background information about (i) GPU computing, (ii) DVFS, (iii) EDP and (ii) Systems and Setup. In Chapter 3, Section 3.1 we provide background on visualization filters and data sets used for power and performance analysis between CPU and the integrated GPU. We present the methodology and results in Sections 3.2 and 3.3 respectively.

In Chapter 4, Section 4.1 we briefly introduce the various machine learning techniques used in this thesis. An overview of our methodology is presented in Section 4.2. We present our experimental results and analysis in Section 4.3. Based on our analysis, we design an ensemble method for GPU power prediction which is presented in Section 4.4. Finally in Section 5.1 we summarize the thesis and describe some future directions.

Chapter 2

Heterogeneous Computation

In this chapter, we first describe an overview of GPU based heterogeneous computing followed by DVFS for power and performance control and the EDP metric for power and performance analysis. Finally we describe the hardware and software we used for this work.

2.1 Graphics Processing Units

Heterogeneous computation involves using multiple kinds of processors in order to gain power and performance benefits. GPUs are computation systems originally meant for graphics processing and which consist of multiple compute cores allowing highly parallel computation. GPU comprise multiple compute units with each compute unit running multiple threads which run in a lock-stepped fashion. GPU heterogeneous computation involves both the CPU and the GPU for processing. As compared to CPU which uses a hierarchy of memory

to speed up memory access, a GPU aggressively performs context switching to hide memory access latency. Thus the GPU specializes in data parallel applications whereas the CPU is best for general purpose computation. Figure 2.1 shows a heterogeneous CPU-GPU system. The GPU is connected to the CPU cores via the PCI bus. The motherboard integrated GPUs tend to share the main memory with the CPU where a discrete GPU comes with its own memory. The discrete GPU is generally much more expensive than an integrated GPU.

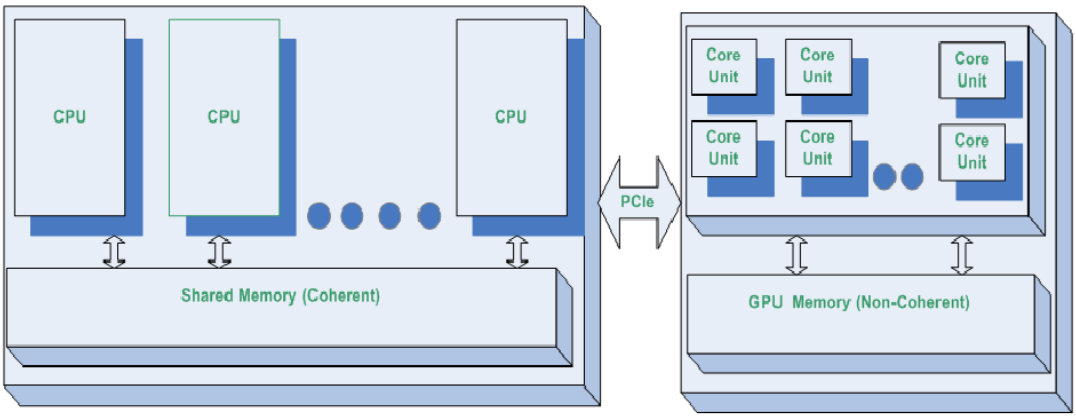


Figure 2.1: CPU-GPU System

GPU heterogeneous computation platforms employ increasing amount of specialization to realize higher performance while consuming a lower amount of power. Apart from involving multi-core CPU and multi-core GPU combinations these systems have a multitude of memory hierarchies and an accelerating larger number of resource combinations. The resulting numerous and subtle hardware and software interaction is leading to several challenges for software developers seeking to optimize power and performance for a particular hardware-software platform. This thesis intends to tackle GPU power and performance issues in systematic ways and uncover newer insights.

2.2 Dynamic Voltage Frequency Scaling

In this section, we describe the technique of Dynamic Voltage Frequency Scaling (DVFS) which is useful for controlling the trade-off between power and performance in the context of heterogeneous computation. CMOS gates are fundamental low-level components in modern semiconductors and power consumption for CMOS gates is proportional to the frequency. The below equation shows the dependency of the dynamic power with CMOS gate voltage and the frequency.

$$Dynamic\ power\ \propto\ capacitive\ load\ *\ voltage^2\ *\ frequency \tag{2.1}$$

Figure 2.2 shows the variation of power and runtime with GPU frequency for the FFT application from SHOC benchmark suite. With the increase in GPU core frequency, power consumption increases but the runtime or delay decreases. Therefore by controlling the frequency, trade-off between power and performance can be made. However, this scaling can get complex in case we vary the frequency of multiple subsystems in addition to the GPU, such as the DRAM. Additionally, scaling complexity can vary with the application as we discussed earlier. We use the DVFS technique to find the optimum frequency for power and performance for the CPU in our study on power and performance analysis.

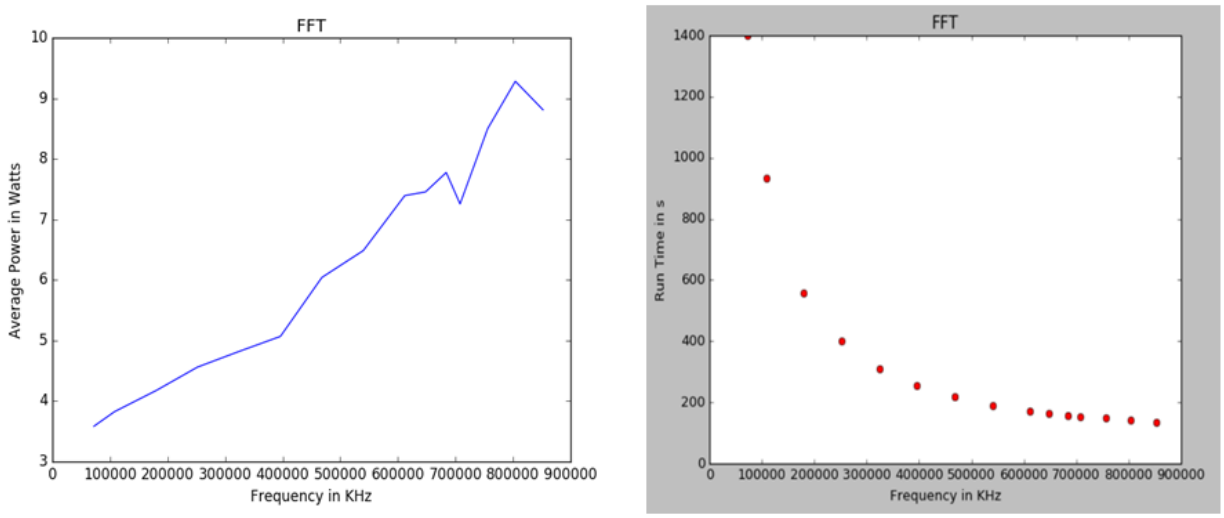


Figure 2.2: DVFS for FFT application

2.3 The Energy Delay Product Metric

In this section, we describe the energy-delay product (EDP) metric which is useful to understand the trade-off between power and performance in the context of heterogeneous computation. The EDP for an application at a particular hardware configuration is:

$$Energy\ Delay\ Product\ = Energy\ *\ Time\ = (Power\ *\ Time)\ *\ Time \tag{2.2}$$

Figure 2.3 shows the typical variation of power and performance with frequency for a dummy application. We can see that with the increase in CPU/GPU core frequency, power consumption increases but the runtime or delay decreases. Therefore there is an optimum frequency to run the application at which the EDP is minimum. Running the application at this frequency helps in the trade-off between power and performance. In our work, we

have used the EDP metric for power and performance analysis between the CPU and the integrated GPU.

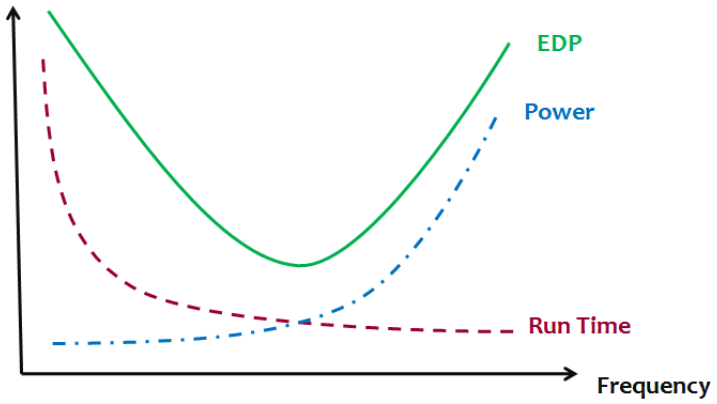


Figure 2.3: EDP variation with frequency

2.4 Systems And Setups

In this section, we present the hardware and software we used for power and performance comparison/analysis and for power and performance prediction.

Power and Performance Analysis

We perform our experiments on a SuperMicro® 7047GR-TR workstation equipped with two 8-core Intel® Xeon® E5-2665 processors and 64 GB of system memory. The processors operate at a frequency of 2.4 GHz and have a TDP of 115 W each. The operating frequency of the processors can be reduced to up to 1.2 GHz in steps of 0.2 GHz to save power. We also run our experiments on the motherboard-integrated Matrox G200 graphics processor

operating at 84 MHz and having a TDP of 4 W. The integrated GPU is connected to the CPU via the PCI bus. Specifically, the Matrox GPU used in our study has two separate 64-bit buses to transfer data in each direction instead of a wider 128-bit single bus. This decreases data-transfer latency, and thus, boosts bus performance. Consequently, large data set visualization tasks become more efficient as data movement becomes less of an issue. For CPU power and performance analysis, we use the classical technique of dynamic voltage and frequency scaling (DVFS). We tune the frequency of the compute node cores and study the power and performance characteristics.

We used the ParaView v5.0.1 visualization tool compiled with MESA libraries for the CPU and OpenGL libraries for the integrated GPU. We use the WattsUp Pro® power meter and Intel® Power-Governor® tool. We repeated all experiments thrice and reported their median value in subsequent sections.

Power Prediction

For power prediction we use the Nvidia® Quadro P4000® graphics hardware platform for our study as it allows high resolution power measurement. This platform is designed for HPC systems and is based on Pascal® architecture. The peak power consumption is 105W. Table 2.1 presents the hardware details. We installed this graphics card on an Intel Xeon® based ubuntu 16.04 system. Both the GPU frequency and the memory frequency are configurable for this platform. We use CUDA 9.0 driver.

Table 2.1: NVIDIA Quadro P4000 Hardware Details

H/W Type	**H/W Detail**
GPU Architecture	Pascal
Number of GPU Cores	1792
GPU Frequencies (MHz)	1708 - 139
Memory Type	GDDR5
GPU Memory	8 GB
Memory Frequencies (MHz)	3802, 810 and 405
Memory Bandwidth (GB/sec)	Up to 243
Peak Power	105 W

Profiling. To measure power, we use the nvprof® tool which has a resolution of up to 5 ms. For collecting power readings of applications with short execution time, we run them in a loop. We use the nvprof® profiler for collecting the application metrics. The profiled metrics are the various GPU resource utilization levels as shown in Table 2.2. Power consumption is directly related to the hardware resource utilization levels and frequencies. We observe that these utilization levels of different hardware blocks are independent of each other and with these, we can obtain average prediction accuracy of 4.5%. This accuracy compares well with other techniques which we discuss in the section 1.2. Therefore we use these resource utilization levels along with frequencies to predict power consumption. Direct correlation analysis may not uncover the best variables in this case because of the complexities of applications in using the various hardware resources. Additionally, using these limited metrics helps in limiting the data dimensions, as due to the curse of dimensionality [11], data which is too thinly spread compared to the data set size, is not suitable for machine learning algorithms. Also, limiting the number of profiled metrics saves profiling time.

Table 2.2: Resource Utilization Metrics

H/W Metric	**Description**
dram_utilization	Utilization level of the DRAM.
issue_slot_utilization	Percentage of issue slots that issued at least one instruction, averaged across all cycles.
l2_utilization	Utilization level of l2 cache.
tex_utilization	Utilization level of texture cache relative to the peak utilization.
tex_fu_utilization	Utilization level of the multiprocessor function units that execute texture instructions.
special_fu_utilization	Utilization level of the multiprocessor function units that execute sin, cos, ex2, popc, flo, and similar instructions.
ldst_fu_utilization	Utilization level of the multiprocessor function units that execute global, local and shared memory instructions.
cf_fu_utilization	Utilization level of the multiprocessor function units that execute control-flow instructions.
single_prec_fu_util.	Utilization level of the multiprocessor function units that execute single-precision floating-point and integer instructions.
double_prec_fu_util.	Utilization level of the multiprocessor function units that execute double-precision floating-point and integer instructions.

Training and Test Applications. In order to train and compare the accuracy of different models, we execute 23 CUDA applications from the scalable heterogeneous computing (SHOC) [15] benchmark suite, the Rodinia benchmark suite [13] and CUDA 9.0 sample applications. We use different applications for training and test. The list of applications is shown in Table 2.3. We select workloads which exhibit a variety of computational and communication patterns and work without throwing errors with the CUDA 9.0 release. We particularly choose CUDA applications as CUDA is known to be well optimized for NVIDIA GPUs as compared to the OpenCL® framework.

Table 2.3: Training and Test Applications

Training App (Benchmark Suite)	**Description**
DeviceMemory (S)	Measures the device memory bandwidth for different memory regions including global, local, constant, texture.
Scan (S)	Measures the performance of the parallel prefix sum algorithm on a large data array of floating point numbers.
Sort (S)	Measures performance for fast radix sort algorithm on single precision floating point key-value pairs.
Spmv (S)	Measures performance of sparse matrix vector multiplication. This is memory bound.
Reduction (S)	Measures the performance of sum reduction operation. The algorithm recursively folds single precision floating values together to generate a single result value from the elements of an array.
Huffman (R)	Measures the performance of the Huffman lossless data compression algorithm.
Hotspot (R)	Simulates the average temperature value of different areas of a chip given the initial temperature and power consumption.
mat.MulDyn.JIT (C)	Measures the performance of matrix multiplication operation.
Transpose (C)	Measures the performance of matrix transpose operation.
Interval (C)	Implements interval arithmetic operations.
radixSortThrust (C)	Implements very fast and efficient parallel radix sort using thrust library and warp-synchronous programming.
Interval (C)	Implements interval arithmetic operations.
lineOfSight (C)	Measures the performance of the simple line-of-sight algorithm which computes all the points along a ray that are visible from an observation point.
mergeSort (C)	Measures the performance of the merge sort algorithm.
Test Apps	**Description**
FFT (S)	Measures the speed of a single and double-precision fast Fourier transform.
MaxFlops (S)	Measures the max floating point capability of a GPU using a large number of floating point operations.
BFS (S)	Measures the performance of the breadth-first search algorithm on an undirected graph.
Triad (S)	For two random 64-bit floating-point value vectors, b and c, this computes a = b + a c where a is a scalar.
Stream Cluster (R)	Measures clustering performance by finding the medians so that each point is assigned to its nearest center.
Gaussian (R)	Solves systems of linear equations using the Gaussian elimination method.
cdpQuadtree (C)	Measures the performance of the quad tree algorithm implemented using CUDA dynamic parallelism.
scalarProd (C)	Measures the performance of scalar products of a given set of input vector pairs.
seg.TreeThrust (C)	Measures the performance of the image segmentation algorithm.
Eigenvalues (C)	Measures the performance of parallel implementation of bisection algorithm for the computation of eigen values.

(S) = SHOC, (R) = Rodinia, (C) = CUDA Samples

Chapter 3

Power and Performance Analysis

3.1 Filters And Datasets

In this chapter, we first discuss the ParaView rendering pipeline. Then we discuss the filters and the data sets that we used for CPU- integrated GPU power and performance comparison and analysis. Finally, we discuss the results.

The Paraview Rendering Pipeline

The basic steps for visualizing data are reading, filtering, and rendering. Figure 3.1 presents the ParaView visualization pipeline. First, data is read into ParaView and next, a number of filters are applied which process the data to generate, extract, or derive features from the data. Finally, a viewable image is rendered from the data. The figure 3.1 describes the

ParaView visualization pipeline. Below is the description of the visualization pipelines that we use.

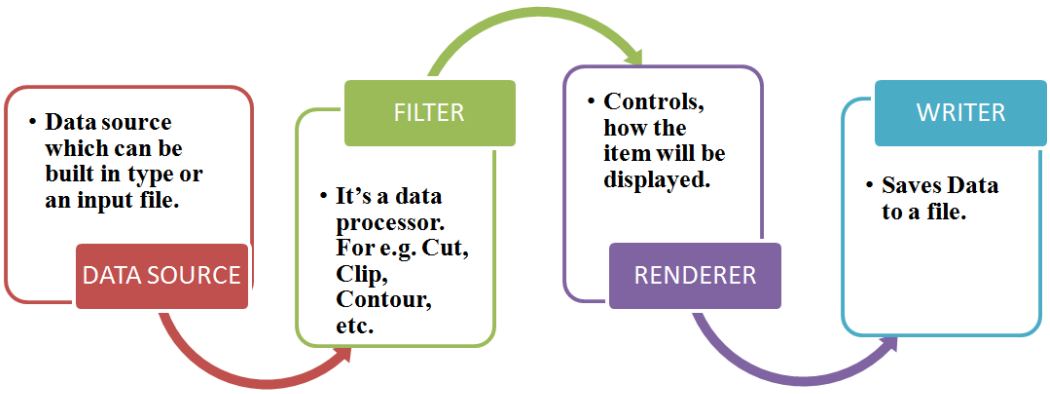


Figure 3.1: The Paraview visualization pipeline components - reader, filter, renderer and writer.

- **Sources:** The ways to get data into ParaView are reading from a data file or generating data with a built-in source object. For MAPS and Particulate Ensemble data sets we read the data from the respective data file and for Cone and Sphere objects we use the Paraview built-in source.

- **Filter Operations:** After data is read, multiple filter operations can be applied to discover much more about the data. These filters are functional units that process the input data in order to generate, extract, or derive features from the data. There are many filters available in ParaView and the ones we investigate are described in the sections on filters.

- **Rendering:** Rendering or image synthesis is the process of generating an image from

a 2D or 3D model. ParaView supports data representation with multiple techniques like surface rendering, volume rendering etc. In this thesis, we do volume rendering for built-in data types and particulate ensemble data set and surface rendering for MPAS data set. With volume rendering, a solid mesh is rendered as a translucent solid with the scalar field determining the color and density at every point and this allows to see the features all the way through a volume. The final output of rendering operation is a digital image or raster graphics image file.

- **Writers:** The final output files are written to disk by writers. Writers provide attributes for e.g.: file format to write ASCII or binary files etc.

The Paraview Filters

Filters connect to form the ParaView visualization pipeline. They process the input data in order to generate, extract, or derive features from the data [8]. Below is the description of some filters that we investigate.

- **Clip:** The Clip filter operates on all types of input data sets and cuts away a portion of the input data set using an implicit plane without reducing the data set dimensionality. The output data type produced by this filter is an unstructured grid.

- **Contour:** The Contour filter operates on any type of input data set and generates isolines or isosurfaces using point scalars. It does this by using a selected point-centered

scalar array. This filter requires input data to have at least one point-centered scalar array. The output data type produced by this filter is polygonal.

- **Glyph:** The Glyph filter operates on any type of input data set and generates a glyph at each point in the input dataset. Examples of glyphs are, an arrow, cone, cube, cylinder, line, sphere, or 2D glyph and they can be oriented and scaled by the input point-centered scalars and vectors. The output data type produced by this filter is polygonal.

- **Shrink:** The Shrink filter operates on any type of input data set and it shrinks each input cell pulling them away from their neighbors and thus causing the individual cells of a dataset to break apart from each other. In this, each point in a cell moves toward the centroid of all the points in the cell. The output data type produced by this filter is unstructured grid output. The main input to this filter is the shrink factor, which determines how far the points will move towards the centroid. A value of 1 keeps the points at their original position and a value of 0 positions the points at the centroid of the cell. We use the Paraview defaults.

- **Slice:** The Slice filter takes any data set as input and slices it with a plane. Similar to a contour, it creates lines from surfaces and surfaces from volumes, extracting the portion of the input data set lying along the sliced plane. The output data type produced by this filter is polygonal.

- **StreamTracer:** The StreamTracer filter operates on any type of input data set having

point-centered vectors and generates streamlines in a vector field from a collection of seed points. Production of streamlines terminates on crossing the exterior boundary of the input data set or on hitting the maximum number of steps, terminal speed, and maximum propagation properties. The output of this filter is polygonal data containing polylines.

- **WrapByVector:** The Warp by vector filter operates on polygonal, curvilinear, and unstructured grid input data sets and translates the points of the dataset along a vector by a distance determined by the specified scalars. The output data set type for this filter is the same as input as in this operation only the positions of the input points are changed.

The Simulation Data sets

The data sets that we use for our analysis are described below. Cone and Sphere are the ParaView built-in data sets. Model for Prediction Across Scales-Ocean (MPAS-O) and Particulate Ensemble are real-world data sets.

- **Built In Data Sets:** We use the ParaView built in three-dimensional sphere and cone data sets. Figure 3.2 shows different ParaView filters applied to the sphere data set.

- **MPAS Ocean:** The MPAS is a collaborative project between climate modeling group

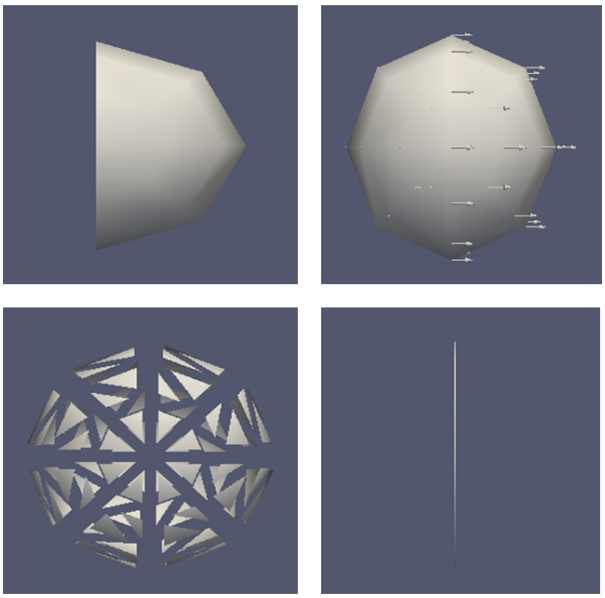


Figure 3.2: Different ParaView filters applied to the sphere data set

at Los Alamos National Laboratory and the National Center for Atmospheric Research. It develops atmosphere, ocean and other earth-system simulation components for use in climate, regional climate and weather studies. MPAS-Ocean can simulate the ocean system in spatial scales from sub 1 km to global circulations and in time scales of months to millennia. Many MPAS model components use the unstructured Voronoi meshes and C-grid discretization used as the basis. We use a scaled down, 1.00*1.00 data set for our analysis. The figure 3.3 shows the glyph filter applied to the MPAS-O data set.

- **Particular Ensembles:** This data set is taken from Scientific Visualization Contest 2016 and it corresponds to the simulation of a solid body of salt dissolving in a cylindrical flow domain that contains water. The salt sits at the top of the cylinder and the top is modeled by a corresponding boundary condition. Due to the transient nature of

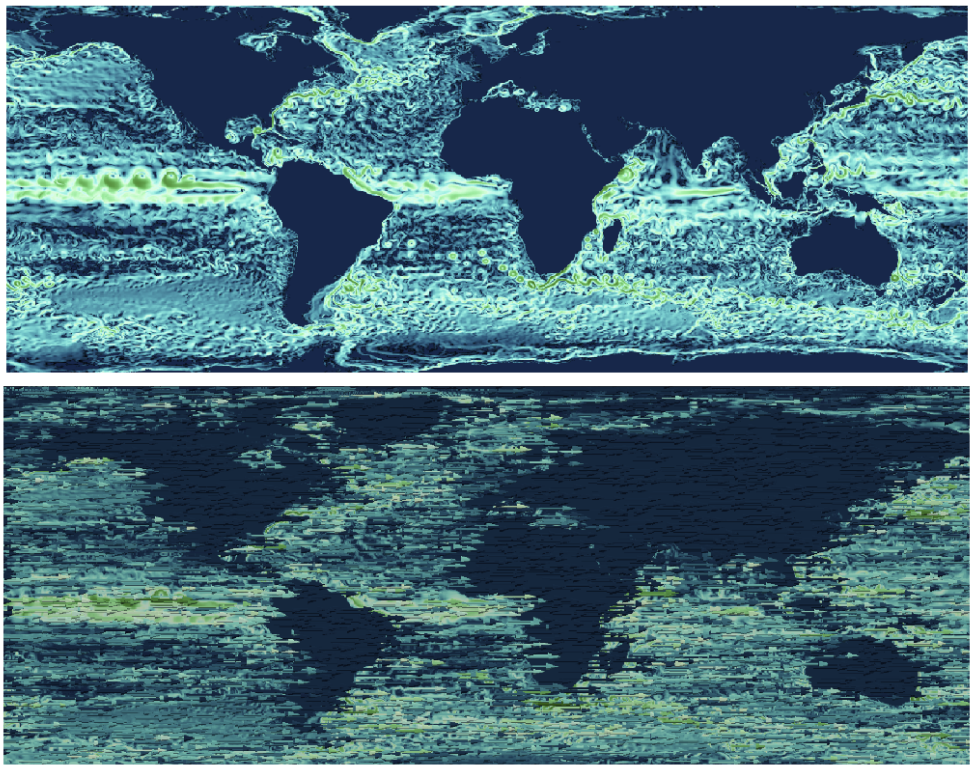


Figure 3.3: The ParaView glyph filter applied to the MPAS-O data set

the solution process, simulation is processed in multiple time steps. For our analysis, we take the data set corresponding to a single run of the simulation.

3.2 Methodology

We measure the impact of techniques such as dynamic voltage and frequency scaling (DVFS) and integrated GPU usage on the *greenness* for certain visualization tasks shown in Table 3.1. *Greenness* may refer to power, energy, or the energy-delay product (EDP).

Table 3.1: Summary of Experiments

Data set/Task	Cone	Sphere	MPAS-O	PE
Clip	X	X	X	X
Contour			X	X
Glyph	X	X	X	X
Shrink	X	X		
Slice	X	X	X	X
Stream Tracer				X
Wrap By Vector				X

3.3 Results

Table 3.2: Normalized execution time, dynamic power, energy and EDP for integrated GPU.

Viz. Task	Data set	Time	Power	Energy	EDP
Clip	Cone	6.71	0.31	2.08	13.98
	Sphere	6.67	0.29	1.92	12.84
	MPAS-O	0.37	0.05	0.02	0.01
	PE	4.61	0.10	0.48	2.20
Contour	MPAS-O	4.21	0.09	0.37	1.57
	PE	5.37	0.24	1.27	6.79
Glyph	Cone	6.26	0.21	1.33	8.29
	Sphere	5.64	0.28	1.55	8.76
	MPAS-O	1.15	0.10	0.11	0.13
	PE	1.01	0.06	0.01	0.01
Shrink	Cone	7.15	0.12	0.89	6.36
	Sphere	6.62	0.19	1.24	8.17
Slice	Cone	6.00	0.26	1.57	9.41
	Sphere	7.00	0.47	3.12	21.78
	MPAS-O	4.62	0.08	0.37	1.65
	PE	4.76	0.15	0.69	3.29
Stream Tracer	PE	3.65	0.17	0.57	2.09
Wrap By Vector	PE	4.87	0.14	0.66	3.23

Table 3.2 presents the execution time, dynamic power, dynamic energy, and EDP for the integrated GPU for various visualization tasks and data sets. The values presented in this table are normalized with respect to the CPU, i.e., normalized execution time = execution time on GPU/execution time on CPU).

We can clearly see that there exists a trade-off between performance and power for the two devices. While the CPU exhibits better performance for nearly all of our visualization tasks, the GPU consistently consumes less power. This trade-off results in widely varying energy and EDP characteristics for the two devices. To shed light on the nature of the *greenness* characteristics of the two devices, we summarize the EDP by data set and visualization tasks in Fig. 3.4 and Fig. 3.5 respectively.

Data Set Results: Figure 3.4 shows the EDP for the CPU and GPU for different data sets. Note that the data sets are arranged in increasing order of data size. The cone and sphere are on the order of kilobytes (KB) while the PE and MPAS-O data sets consume megabytes (MB) and gigabytes (GB), respectively. The EDP increases with increase in data size, but the rate of increase differs for the two devices. The cone and sphere show similar EDP profiles; that is, the CPU consumes significantly less EDP in both the cases, as they are similar in size. However, as the data size increases, the percentage difference in EDP shrinks significantly as in the case of PE, and when the data size exceeds a gigabyte (MPAS-O), the EDP of the GPU becomes lower (i.e., better) than the CPU.

Visualization Task Results: Figure 3.5 presents the mean EDP for the CPU and the GPU for different filters. We find that the modern CPU is well optimized for visual processing tasks and delivers a better EDP for nearly all the tasks. The average EDP for the CPU is 21% better than that of the GPU. One major exception is the glyph filter, where the integrated GPU does better with an EDP that is only 89% of the CPU's EDP. This is because the glyph operation is computationally more intensive than the other operations.

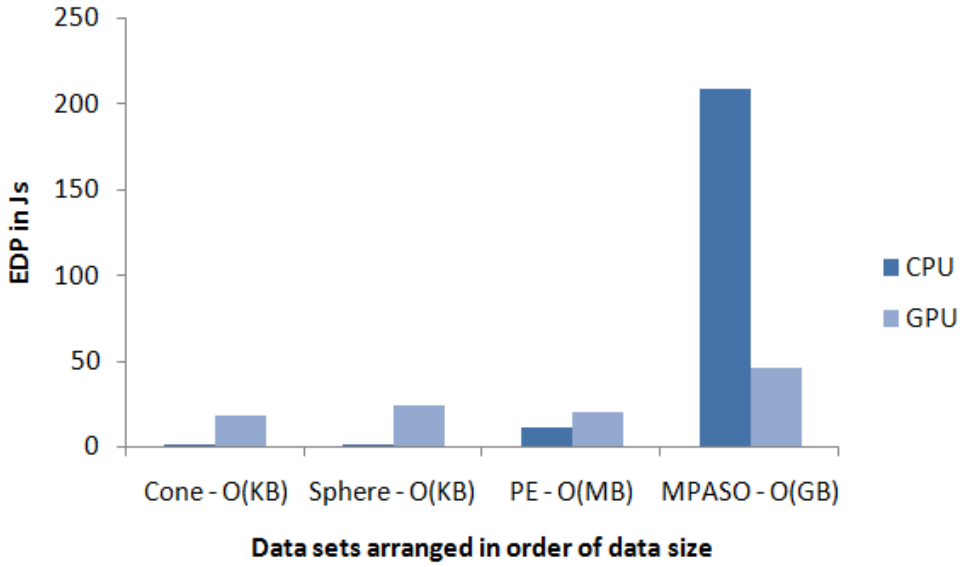


Figure 3.4: Mean EDP for the GPU and the CPU for different data sets

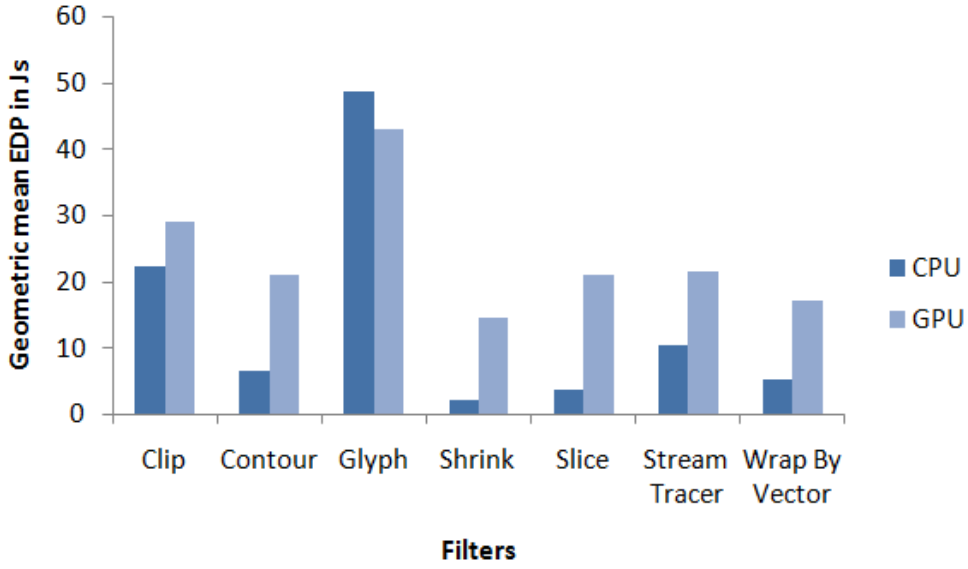
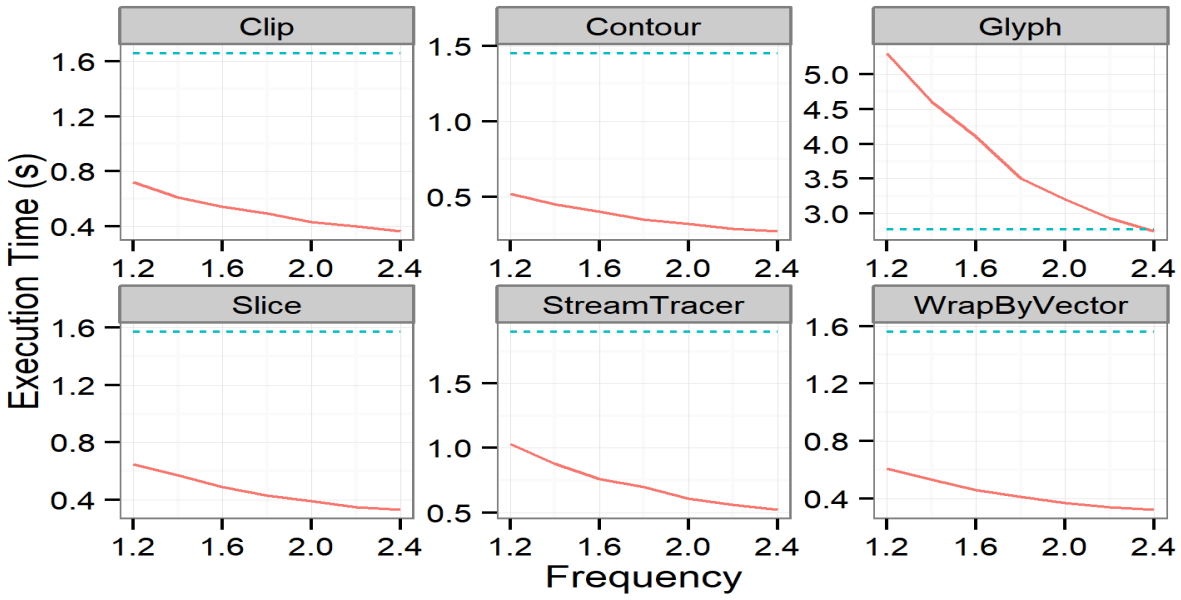


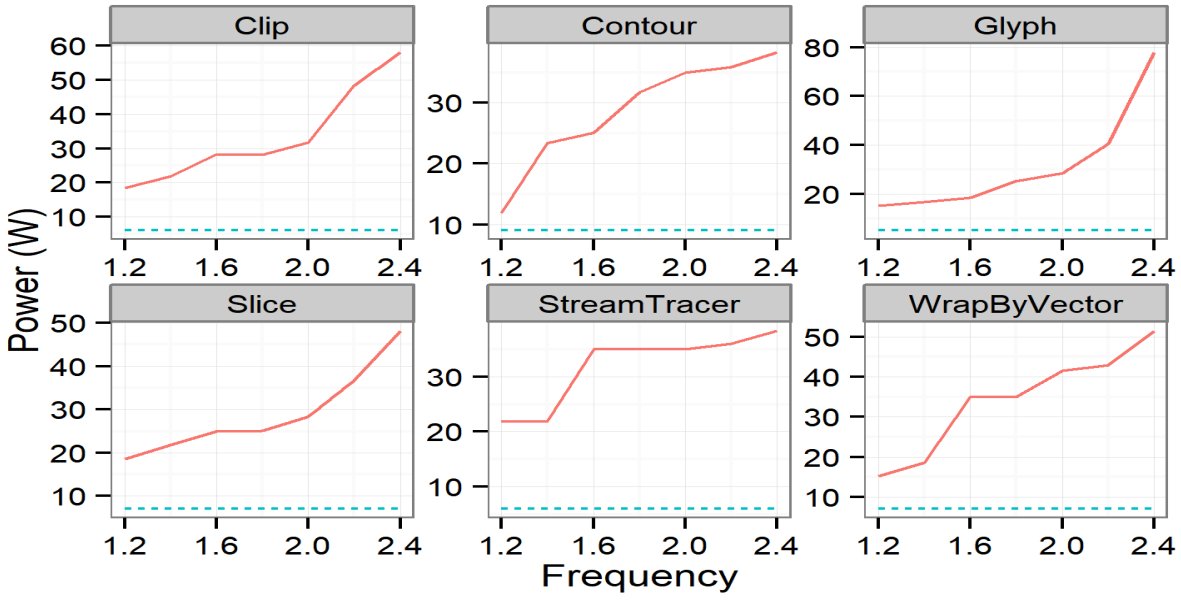
Figure 3.5: Mean EDP for the GPU and the CPU for different filters

Reducing Power Consumption with DVFS

Here we investigate whether the power consumed by the CPU can be reduced to a level below the GPU via DVFS. We also seek to understand what the impact of DVFS would

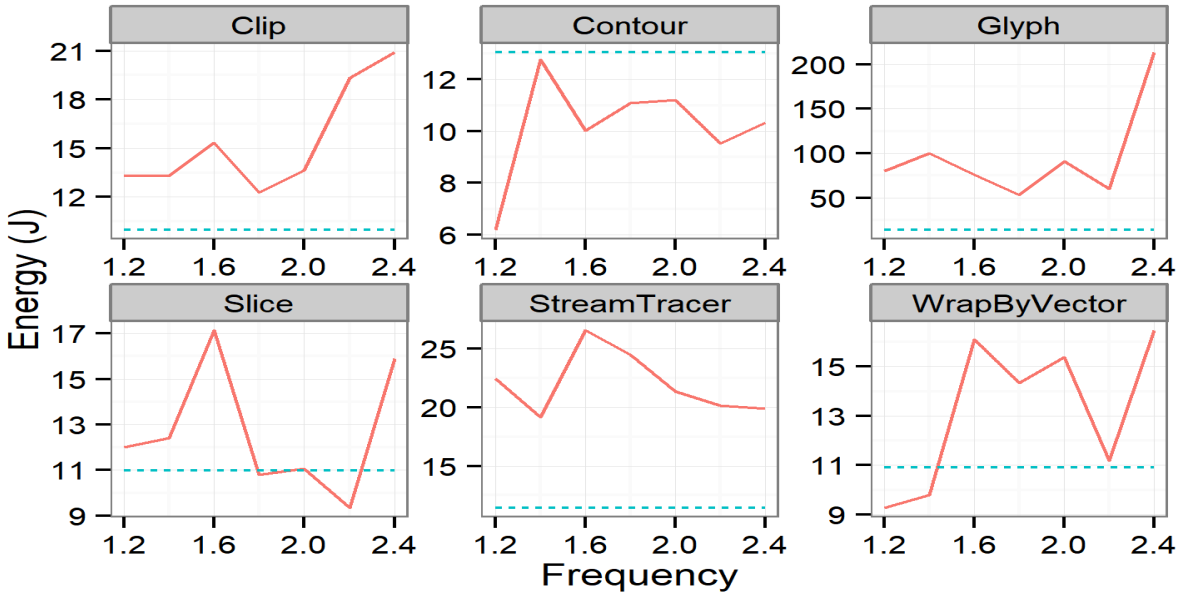


(a) Performance

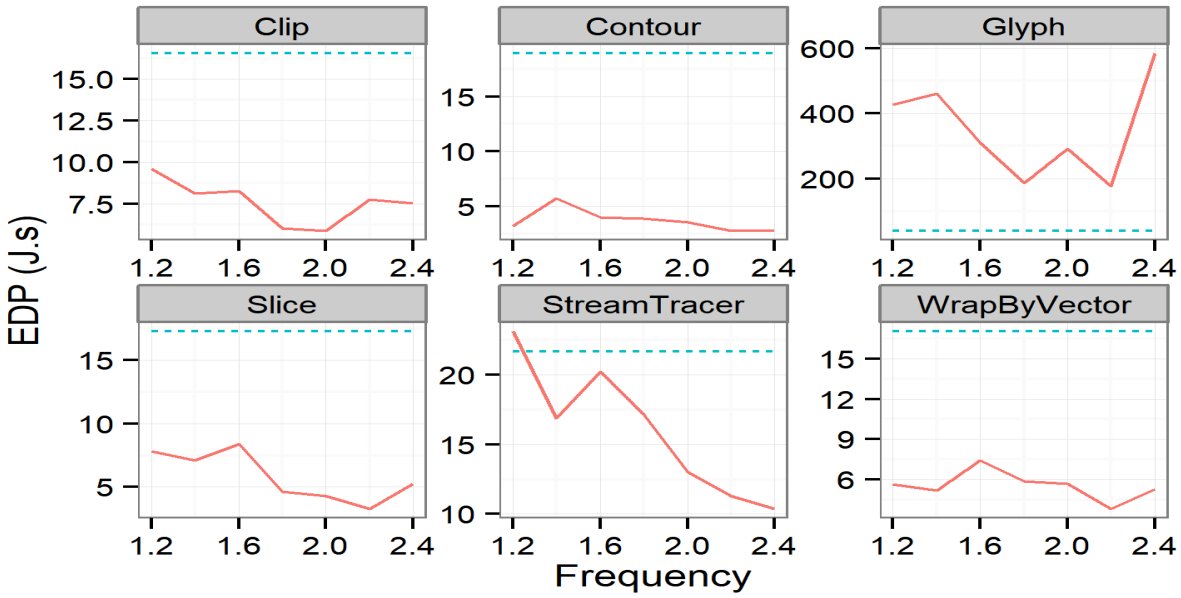


(b) Power

be on other relevant metrics such as performance, energy, and EDP. For brevity, we only present the results for the PE data set.



(a) Energy



(b) EDP

Figure 3.6: Greenness results for PE data set at different CPU frequencies. Solid line = CPU, Dotted Line=GPU

Fig. 3.6 presents the performance, power, energy, and EDP for the CPU using DVFS with different filters, respectively. In particular, Fig. 3.6(b) shows the power consumed by the CPU at various frequencies for different filters. The power consumed by the GPU is shown in dotted lines for reference. Note that the GPU frequency cannot be changed and is fixed in these experiments. We observe that while the CPU power consumption is reduced significantly for all six filters, in none of the cases does the power reduce to a value that is below the integrated GPU. This suggests that when power reduction is the primary criterion for optimization, one must use the integrated GPU.

Previous experiments suggested that the CPU consumed lower energy in all the cases. However, if one attempts to reduce the power consumption by reducing the operating frequency, then the corresponding slowdown is significant enough (Fig. 3.6(a)) to increase the energy consumption above the GPU as in the case of *slice* and *wrap by vector* (Fig. 3.7(a)).

We expect performance to have a greater weight in HPC systems, so we compare the EDP metric in Fig. 3.7(b). While the EDP of the CPU remains below GPU at all frequency levels *slice* and *wrap by vector*, for the *stream tracer* filter, the GPUs turn out to be *greener*. Overall, our results show a complex interplay between the various parameters in determining power and performance for a GPU but EDP metric is a great simplifier for evaluation.

Chapter 4

Power Prediction

In this chapter, we first discuss the base machine learning and statistical methods that we explored for the problem of GPU power prediction for a new application at different frequencies. We then discuss the methodology and the results. Finally, we discuss the novel ensemble method that we created for higher accuracy power prediction.

4.1 Base Methods

In this section, we describe the machine learning techniques we used and on why we choose these particular techniques. Machine learning is the technique of prediction of the value of a response variable given the value of predictor variables and a model. The model is is built using a known data set called the training data set. Figure 4.1 shows the methodology with inputs and output for our case. Machine learning methods are classified as classification

and regression techniques. We use regression methods as we have to predict the continuous variable of power. The techniques we choose cover a wide range of fundamental regression learning techniques such as straight line fitting, curve fitting, nearest neighbors, decision trees, voting, and neural networks. Additionally, these could successfully predict *continuous* values in closely related fields. In this work, we are not concerned about the performance overhead of the techniques as the intent is off-line power prediction. For the sake of consistency, we use terminology from the `weka` [20] workbench throughout the thesis. Generic names for the methods are included in the description below. Table 4.1 lists the parameter values with weka that we use for each of the methods.

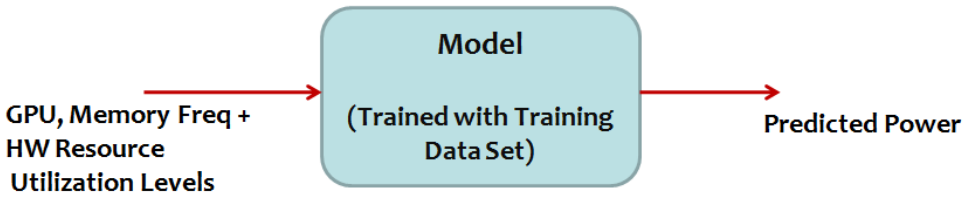


Figure 4.1: Machine Learning Based Power Prediction

ZeroR: `ZeroR` simply predicts the *mean* power consumption across all applications used in the training data set. This technique has no prediction power and is only used as a baseline to compare the prediction power of the other techniques explored in the thesis.

Simple Linear Regression (SLR): SLR is a widely-used line-fitting technique to predict the power consumption of a processor. As shown in Fig. 4.2, the task is to find a straight line that predicts the *response* variable Y as a function of *predictor* variable X. More concretely,

the technique estimates the slope of the line and the intercept which minimizes the *residuals* (i.e., the distance between the data points and the fitted line). While Figure 4.2 predicts the value of Y based on a line (i.e., a single predictor variable), a hyperplane is also commonly used in a related technique known as multi-linear regression.

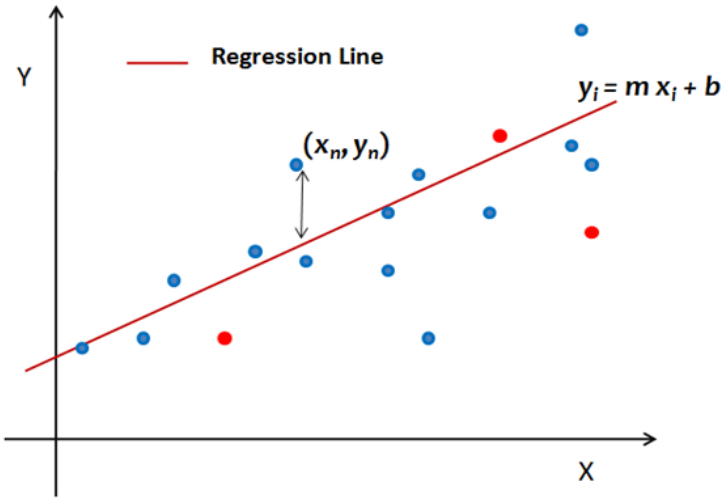


Figure 4.2: Simple linear regression, fitted line

Sequential Minimal Optimization Regression (SMOreg): Informally, `SMOreg` finds a curve (unlike `SLR` which finds a straight line) that explains the relationship between the predictors and the response variable as shown in 4.3. While any learning technique can be used to find the curve, here we rely upon support vector machine (SVM) [29] to accomplish the task of optimizing the prediction bounds for a given regression.

The model can be stated as follows: the i^{th} data point, with weights w and bias b, can be expressed as $y_i = (w^T x_i + b)$. To find the values of the unknown weights, we solve the following optimization problem:

$$\begin{aligned} \underset{w,\xi_i,\xi_i^*}{\text{minimize}} \quad & \frac{1}{2}||w||^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*) \\ \text{subject to} \quad & y_i - (w^T x_i + b) \leq \epsilon + \xi_i \\ & (w^T x_i + b) - y_i \leq \epsilon + \xi_i^* \\ & \xi_i, \xi_i^* \geq 0 \end{aligned} \tag{4.1}$$

Here ξ_i and ξ_i^* are slack variables that specify the upper and the lower training errors subject to an error tolerance . C is a positive constant that determines the degree of penalized loss when a training error occurs.

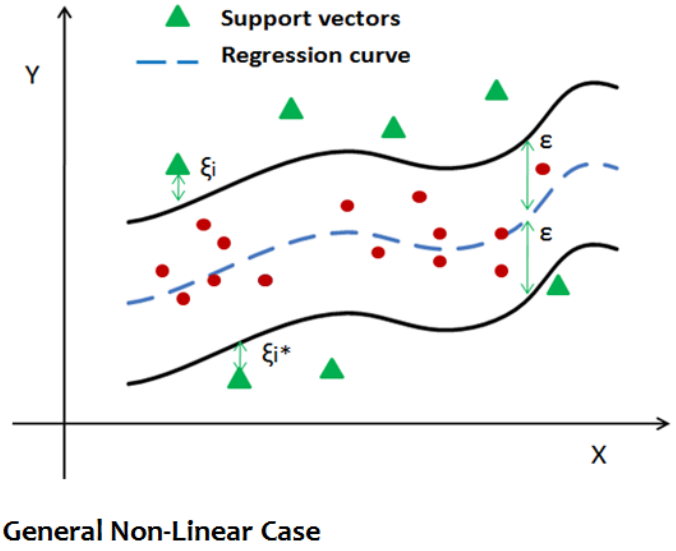


Figure 4.3: Support vector machine for regression

K-Nearest Neighbor (KNN): `KNN` is a popular technique which predicts based upon the k nearest neighbors of the prediction data point as shown in figure 4.4. To predict the power consumed by a new *test* application, `KNN` uses application data points showing similar

characteristics. The number of neighbors to use for estimation is determined using training set cross-validation, and in our case, it turns out to be *three*. We use distance-weighted average `KNN` to estimate the power consumed by the new *test* application.

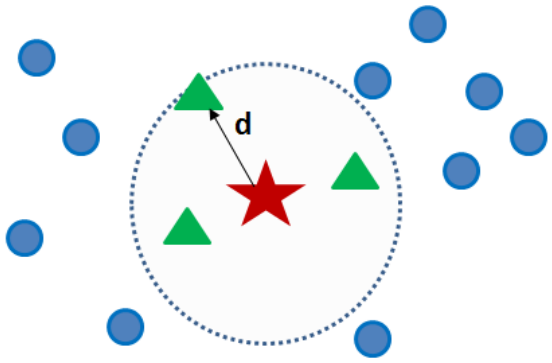


Figure 4.4: K-Nearest Neighbor

Reduced Error Pruning (REP) Tree: `REPTree` [38] predicts based on decision trees and it uses information gain/variance for decision splits at tree nodes. Reduced-error pruning is used to avoid overfitting. Figure 4.5 shows the decision tree we built using the training dataset.

Bagging: Figure 4.6 presents the bagging algorithm. In this multiple decision trees/models are created by splitting the training data. Each tree/model predicts individually and the final prediction is made by voting mechanisms.

Random Forest (RandomForest): Predicts using votes from a number of random decision trees as shown in figure 4.7. The individual random decision trees of the forest are

```
tex_utilization < 0.84 : 58.34
tex_utilization >= 0.84
|   Frequency < 1486.5
|   |   l2_utilization < 2.05
|   |   |   single_precision_fu_utilization < 1.1
|   |   |   |   issue_slot_utilization < 13.63
|   |   |   |   |   issue_slot_utilization < 8.89
|   |   |   |   |   |   Frequency < 980.5 : 31.28
|   |   |   |   |   |   Frequency >= 980.5 : 32.94
|   |   |   |   |   issue_slot_utilization >= 8.89 : 31.05
|   |   |   |   issue_slot_utilization >= 13.63 : 35.58
|   |   |   single_precision_fu_utilization >= 1.1
|   |   |   |   l2_utilization < 1.98
|   |   |   |   |   Frequency < 980.5 : 30.22
|   |   |   |   |   Frequency >= 980.5 : 31.79
|   |   |   |   l2_utilization >= 1.98 : 25.75
|   |   l2_utilization >= 2.05
|   |   |   special_fu_utilization < 0.52
|   |   |   |   l2_utilization < 2.29 : 34.55
|   |   |   |   l2_utilization >= 2.29
|   |   |   |   |   l2_utilization < 2.84 : 31.51
|   |   |   |   |   l2_utilization >= 2.84 : 29.73
|   |   |   special_fu_utilization >= 0.52
|   |   |   |   double_precision_fu_utilization < 1.84
|   |   |   |   |   tex_utilization < 1.1 : 45.31
|   |   |   |   |   tex_utilization >= 1.1 : 42.3
|   |   |   |   double_precision_fu_utilization >= 1.84 : 33.21
|   Frequency >= 1486.5 : 38.81
```

Figure 4.5: Trained decision tree to predict the power consumption.

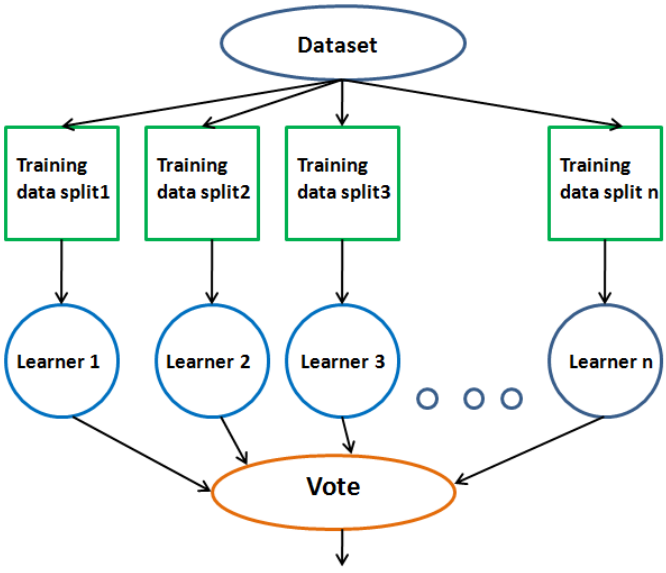


Figure 4.6: Bagging

constructed by randomly selecting the splitter attributes of the nodes of the decision trees.

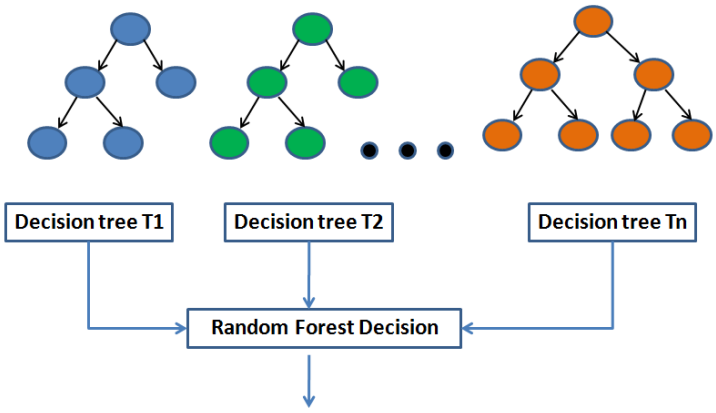


Figure 4.7: Random Forest

Neural Network/NeuralNet/MultilayerPerceptron: Figure 4.8 presents the neural network that we use. Neural network predicts using biological brain-inspired algorithmic model. It consists of several layers of artificial neurons with each neuron taking an input of the form $Y = (w^T X + b)$ where w is the weight vector and b is the bias. The neural node output is sigmoid, however, the final layer output in our case is in un-thresholded linear units as ours is a regression problem. The weights of the network are learned in the training phase through the technique of gradient descent which tries to iteratively minimize the error with training data set.

4.2 Methodology

In this section, we describe the data collection, data preparation, and our evaluation methodology. Our goal is to predict the power consumption of a new application at different GPU

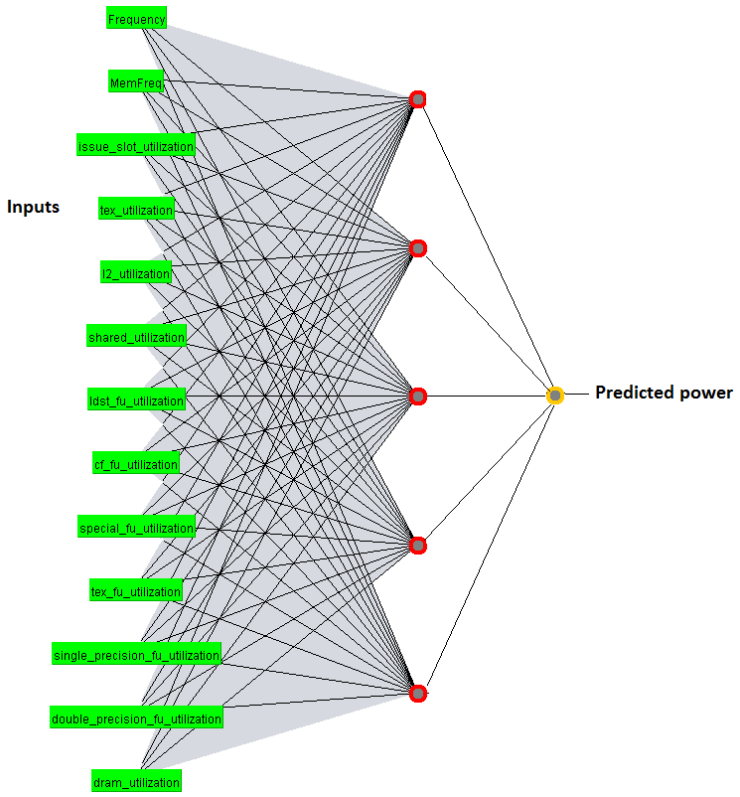


Figure 4.8: Neural Network

Table 4.1: Parameter Values of ML/Statistical Techniques

Method	**Parameters**
ZeroR	None
SLR	None
SMOreg	C=1.0, PolyKernel, RegSMOImproved
KNN	KNN=5, CrossVal=true, 1/distance weighting Algorithm=LinearNNSearch
REPTree	maxDepth=-1, minVarianceProp=0.001
Bagging	bagSizePercent=50, Classifier = REPTree
RandomForest	bagSizePercent=50, maxDepth=0
NeuralNet	layers=3, learningRate=0.3, momentum=0.2

and memory frequencies based on a training data set and the application's resource utilization levels.

Data Collection. In order to make accurate predictions, machine learning algorithms should contain diverse samples with different parameter configurations. To collect data sets for training and test, we run multiple GPU intensive applications at different GPU and memory frequencies and collect the nvprof resource utilization metrics.

Data Aggregation. In this step, we aggregate the execution time and utilization levels (i.e., performance counters) for an application across different CUDA kernels. Since an application can be composed of several CUDA kernels, we measure an application's utilization level of a particular resource according to the below formula:

$$Application's\ utilization\ level\ of\ a\ resource = \frac{\sum_{i=1}^{n}(Resource\ utilization\ level\ of\ kernel_i * Time\ spent\ in\ kernel_i)}{\sum_{i=1}^{n} Time\ spent\ in\ kernel_i} \tag{4.2}$$

Modeling. With the training data, we create machine learning and statistical models. The KNN is an exception here as it is a lazy predictor which predicts directly using the training data. That is, there is no pre-built model with KNN which can increase the prediction overhead. However, analyzing the overheads of the techniques are beyond the scope of this thesis. For analyzing the machine learning models we use R studio and weka [20] software tools.

Model Evaluation. The evaluation metric we use for individual observations, maximum and minimum error comparison is the percentage absolute error which is defined as:

$$Absolute\ error\ \% = \frac{|predicted\ value_i - actual\ value_i| * 100}{actual\ value_i} \tag{4.3}$$

For overall comparisons, we use the percentage MAE metric defined as:

$$MAE\ \% = \frac{100}{n} * \sum_{i=1}^{n} (\frac{|predicted\ value_i - actual\ value_i|}{actual\ value_i}) \tag{4.4}$$

The ZeroR algorithm predicts the mean of the predicted variable ignoring all the predictors. Since it has no prediction power, we use it as the benchmark for evaluating all other algorithms.

$$ZeroR's\ prediction\ value = \frac{\sum_{i=1}^{n}(actual\ value_i)}{n} \tag{4.5}$$

4.3 Results

In this section, we present the accuracy of the various machine learning techniques with respect to mean absolute error (MAE) percentage. A lower MAE indicates a more accurate technique.

4.3.1 High-Level Overview of Results

Figure 4.9 shows the error distribution for all the techniques across all test applications and frequencies. For readability, we present this information in two different graphs. The first graph compares `SLR`, `SMOreg`, and `NeuralNet` against the baseline `ZeroR`. The peak of the distribution gives the most commonly seen errors. The baseline technique, `ZeroR` often produces an error around 20%. This means, if we design a *power capping* system based on this method, we need to provide a safety margin of at least 20% which can be very inefficient. The `SMOreg` and `SLR` both perform better with most common errors around 12% and 14% respectively. The `NeuralNet` method produces significantly worst results with a common error close to 30%. This is to be expected given that the neural network technique needs numerous data points to train the model which is not the case with our training dataset. Looking at the tail of the distribution, we find that the maximum prediction error is around 46% for `SMOreg`, with other models not far behind with maximum errors around 40%. Analyzing further, we found that most of these high error points come from outlier applications described in Section 4.3.2.

The second graph shown in Figure 4.9 compares the error distribution of `Bagging`, `KNN`, `RandomForest`, and `REPTree` methods. The graphs show that the `Bagging` technique performs slightly poorly compared to the three other methods. `KNN`, `RandomForest`, and `REPTree` show significant overlap in their distribution with a most common error of around 14% which is the same as `SLR` from the first graph. `RandomForest` seems to produce more uniform predictions as it contains only one peak in the distribution. The other methods

contain one peak and one or more ridge in comparison. The tail distribution is largely indistinguishable from one another.

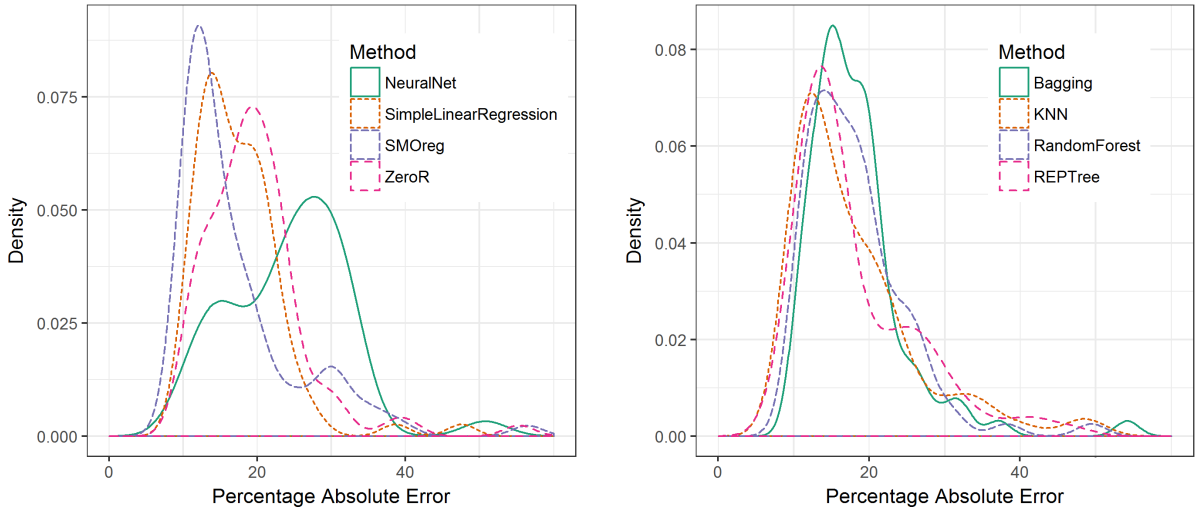


Figure 4.9: Error distribution for the various machine learning techniques

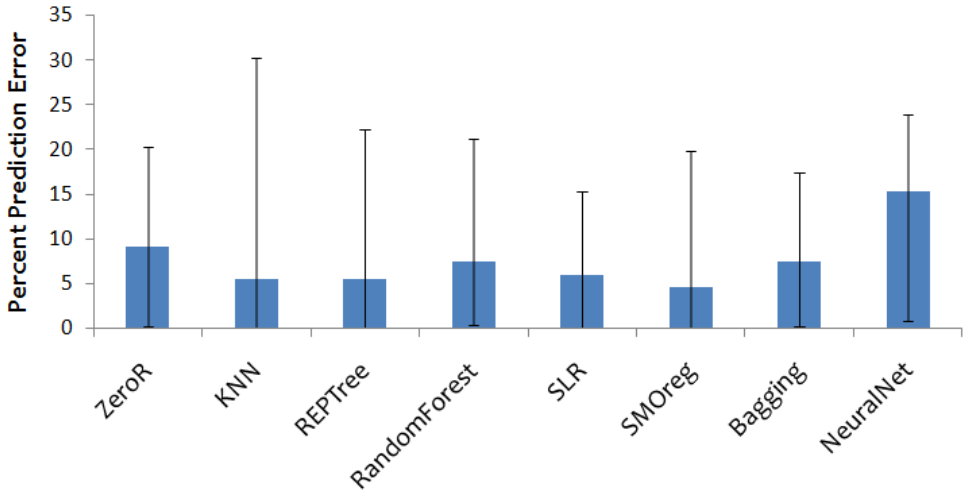


Figure 4.10: Percentage prediction errors for different algorithms/models

Figure 4.10 shows the mean absolute error (MAE) percentage and the error bars not including outlier applications described in Section 4.3.2. `SMOreg` shows the best accuracy with respect to MAE at 4.5%, followed by `REPTree` and `KNN` with MAE of 5.5%. `SLR` has MAE

of 6%, and `RandomForest` and `Bagging` have a MAE of about 7.5%. `NeuralNet` performs worst with an MAE of 15% and this is probably due to the requirement of a huge training data set for `NeuralNet` to be an effective predictor. With respect to maximum error `SLR` is best with 15% error, followed by `Bagging` at 17% and `SMOreg` at 20%. `KNN` gives worst results as it only considers nearest points for prediction which can be inaccurate in case of complex relationship between the predictors and response variables.

4.3.2 Results in Detail

Next, we delve into application-wise results and the accuracy of the different methods at different GPU configurations.

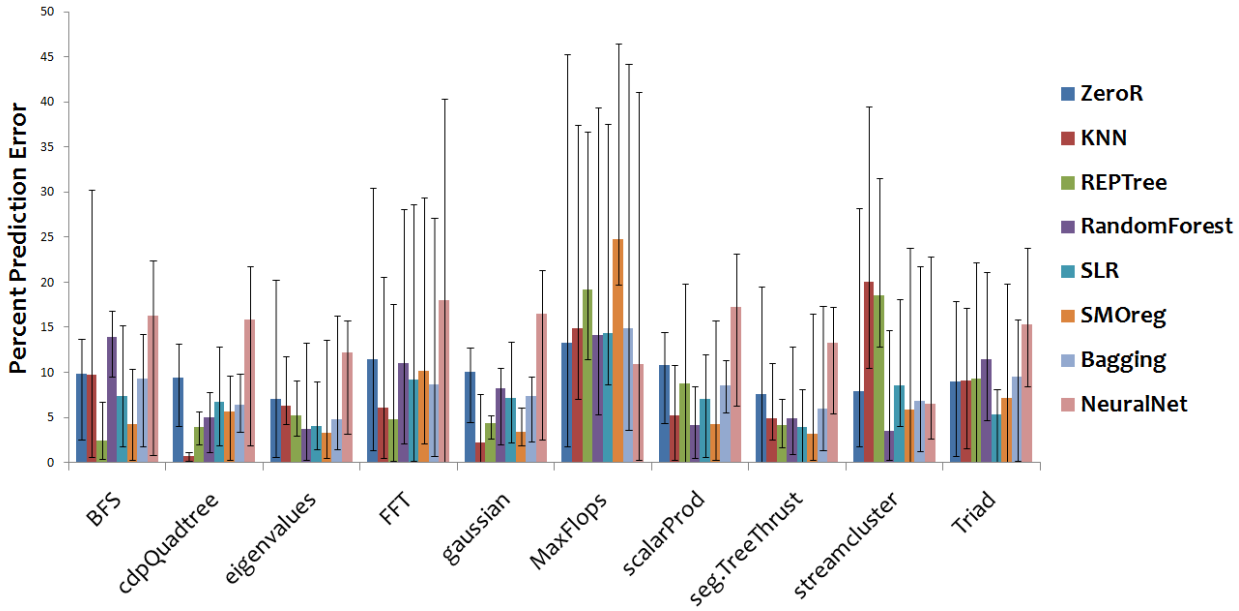


Figure 4.11: Mean absolute error (MAE) percentage for test applications

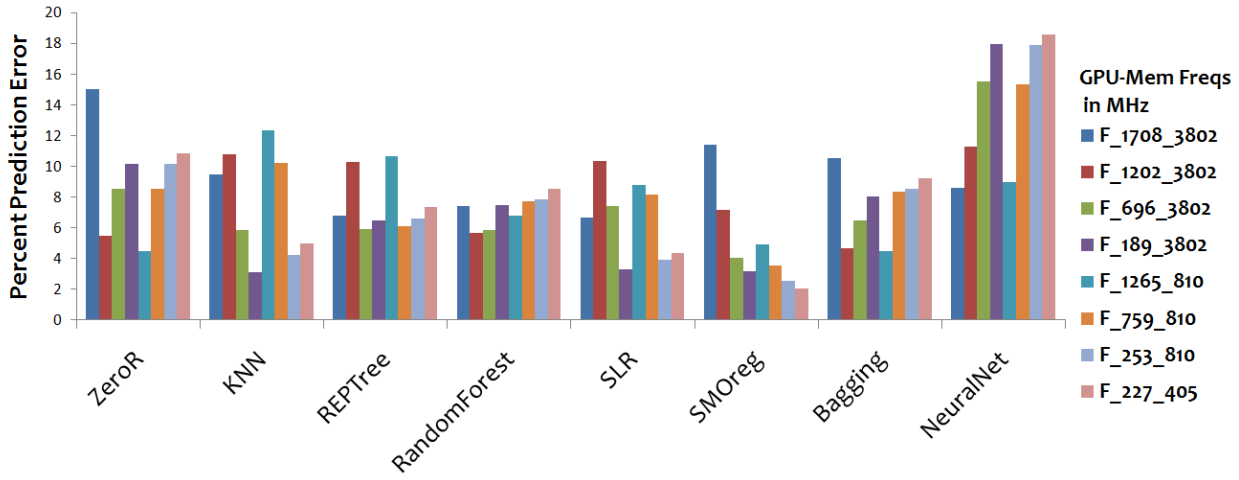


Figure 4.12: Mean absolute error (MAE) percentage for different GPU configurations

Application Results Figure 4.11 shows the minimum, mean, and maximum errors for the various test applications and machine learning techniques. We observe that both the MAE and maximum errors are higher for applications which stress the hardware resources to the maximum – `MaxFlops` is a compute-bound application [15], `FFT` stresses the memory and requires higher memory bandwidth [15], `StreamCluster` is bounded by the PCIe bandwidth [13]. While the relatively poor accuracy seen for these applications is due to the lack of suitable hardware counters (for instance, a performance counter of PCIe utilization is lacking in modern GPUs), most of the relatively less accurate results can be explained by coverage (or lack thereof) of the test applications. We believe a methodological approach towards selecting test applications for modeling [5] can address this in the future. We also believe that equipping future generations of GPUs with more appropriate counters can help in several power management techniques. For subsequent aggregate analyses, we ignore these outlier applications to provide more meaningful insights.

Frequency sensitivity Figure 4.12 shows the MAE percentage for the various machine learning techniques at different GPU configurations. We observe that `NeuralNet` gives the worst predictions consistently across different GPU-memory frequency combinations. The MAE for this method exceeds 16% for *four* out of the *eight* combinations. We also observe that `ZeroR` gives worst results at the extreme frequencies with MAE of around 15% and 11%. This is because it always predicts the mean of the training data set which will correspond to the power consumption at the middle frequencies. Other interesting observations include the following. `RandomForest` proves to be a reliable predictor showing an average error of around 7.5% consistently for all the configurations. This is because this method bases its predictions from multiple decision trees each of which is specially optimized for the various subspaces of the GPU configuration space. Another observation is that `SMOreg` shows a greater accuracy at lower frequencies. While its average MAE is 11% at the highest frequency combination, it drops significantly to only 2% at the lowest frequency combination. Extending the idea of `RandomForest` (which combines different decision trees together), we believe we can improve the prediction accuracy by combining the different methods together. This aspect is explored in Section 4.4.

4.3.3 Statistical Significance of Results

In order to statistically compare the prediction accuracy of the different algorithms, we use `Tukey's HSD` (honest significant difference) test. This test gives us the pairwise statistical difference between mean error value predicted by different algorithms.

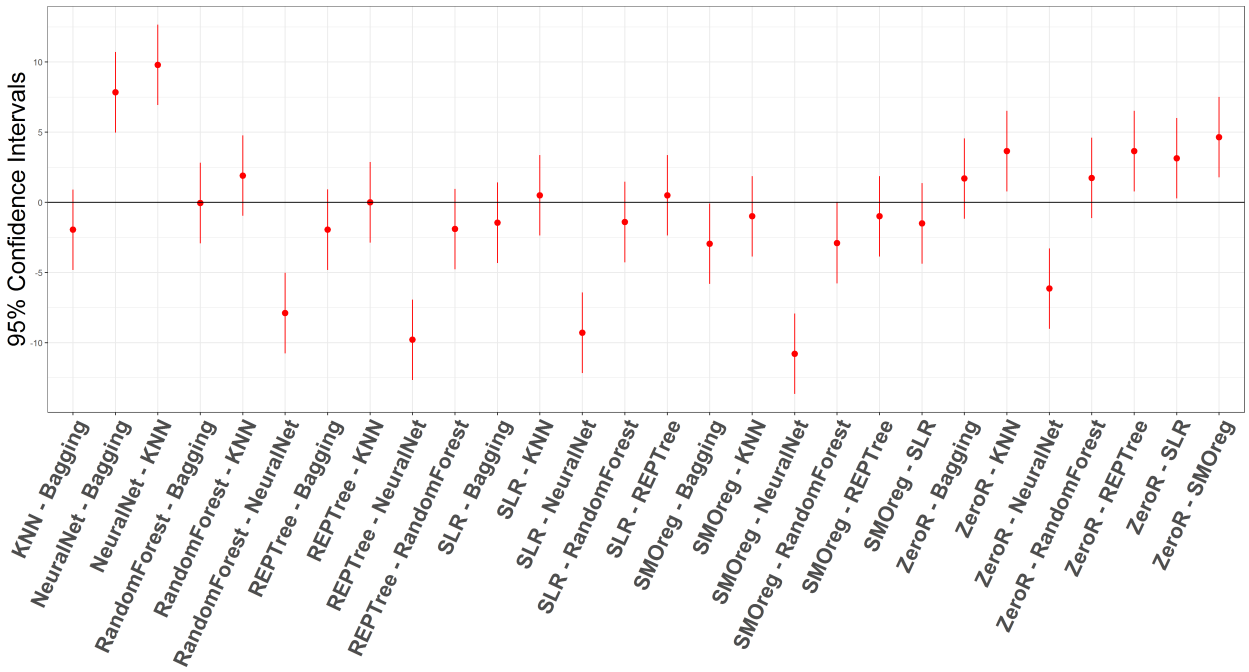


Figure 4.13: Tukey HSD Confidence Intervals for MAE difference between algorithm pairs

Figure 4.13 presents the results of `Tukey's HSD` test. The bars show the 95% confidence interval between lowest estimate and highest estimate of the difference between percentage MAE of algorithm pairs. For example for the comparison between `ZeroR` and `SMOreg`, we can say with 95% confidence that the MAE difference between `ZeroR` and `SMOreg` is 2% at the least and 7.5% at the most. When the lines in this graph cross *zero* axis, it means the difference in the prediction error is not significant enough to say that one method is better than the other (i.e., the result is not statistically significant). From the figure 4.13, we observe that *SMOreg*, *simple linear regression*, *KNN*, and *REPTree* exhibit better accuracy than the baseline `ZeroR`, statistically. We can also conclusively say that `NeuralNet` performs worse than every other technique. Overall, our experiments thus far have shown that only certain methods are better with all other comparisons with baseline proving to be inconclusive.

4.4 An Ensemble Method for GPU Power Prediction

4.4.1 Approach

In order to improve the prediction accuracy, we investigate various algorithm ensembles. The algorithms for an ensemble can be selected based upon a variety of criteria such as prediction accuracy with respect to MAE and with respect to the maximum error, and accuracy at different frequencies. Additionally, different weights can be given to predictions from different algorithms. In our case, we evaluate multiple ensembles created from algorithms based upon prediction accuracies and frequency response. We provide results from some of the more accurate predictor ensembles and ensemble created from all the methods. We use inverse MAE weights as compared to direct accuracy weights, as in our case they have more distinct values. The formula for the same is given below.

$Ensemble's\ Prediction =$

$$\frac{\sum_{i=1}^{n}(Algorithm_i\ Prediction * \frac{1}{Algorithm_i Percentage MAE})}{\sum_{i=1}^{n}\frac{1}{Algorithm_i Percentage MAE}} \tag{4.6}$$

We obtain a considerable gain in prediction accuracy with ensemble methods as shown in the results section. Figure 4.14 shows the top 3 methods inverse MAE weighted average ensemble that we created for power prediction.

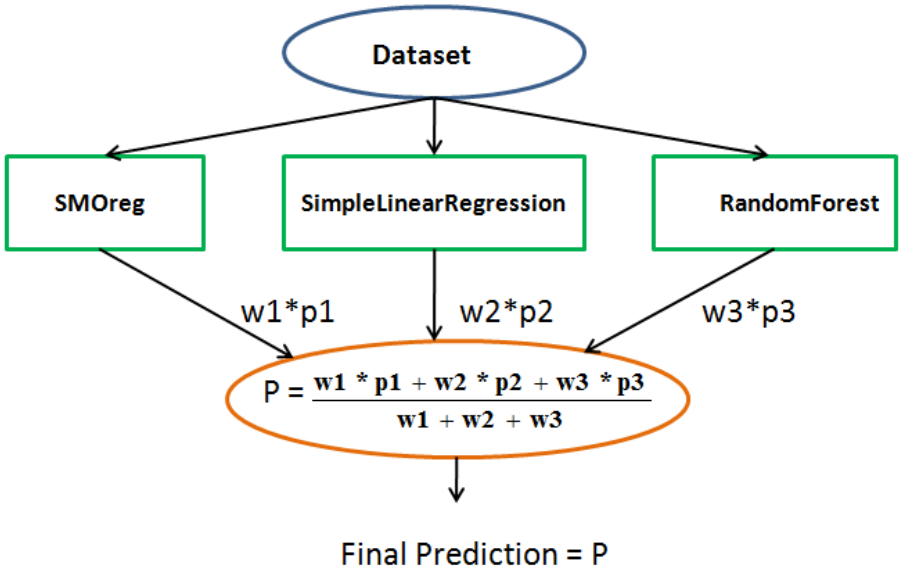
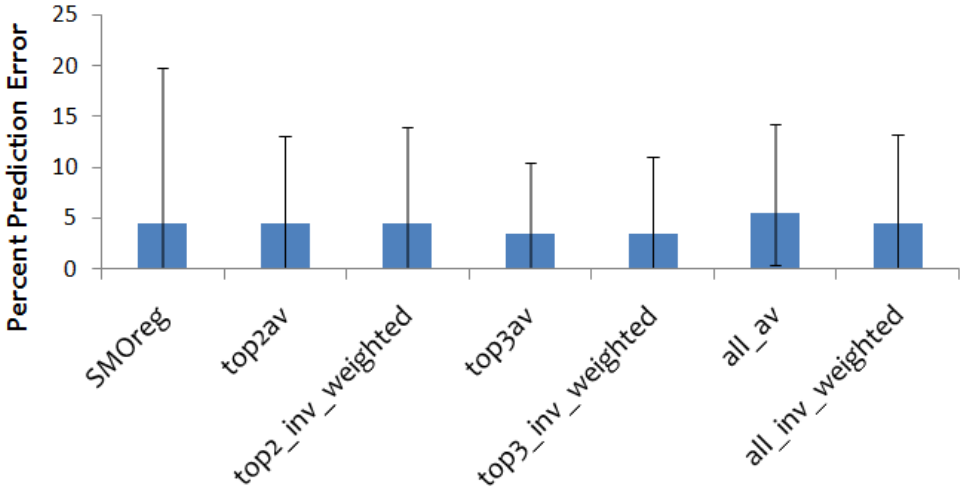


Figure 4.14: Ensemble



Note: top2 → SMOreg+SLR, top3 → SMOreg+SLR+REPTree

Figure 4.15: Percentage prediction error for various ensembles.

4.4.2 Experimental Results

Figure 4.15 shows the various ensembles we created and their prediction accuracy results. The `top2_av` ensemble averages the prediction of the 2 most accurate individual predictors

`SMOreg` for MAE, and `SLR` for max error. The `top2_inv_weighted` weights them inversely to their error percentage. For the `top3_av`, and texttttop3_inv_weighted ensembles we also incorporate `REPTree`. We choose `REPTree` as it has relatively lower MAE and max error. We also create ensembles from all the algorithms both averaged - `all_av` and inverse weighted averaged - `all_inv_weighted`. `top3_av` and `top3_inv_weighted` are the most accurate predictors having MAEs of 3.5%, and 3.4%, and max errors of 10.5%, and 11% respectively. These ensembles show similar results because the base algorithms of the ensembles have similar MAEs which results in similar weightages. The MAE for `all_av`, and `all_inv_weighted` are 5.5%, and 4.5% and maximum errors are 14.3%, and 13.2% respectively. From the results described above, we observe that on using an ensemble of the top three individual predictors, the MAE error is reduced to 3.5% and the maximum prediction error is reduced to 11%. For comparison, the best individual predictor `SMOreg` has MAE of 4.5% and a maximum error of 20% respectively. Additionally, other ensembles also deliver more accurate results compared to the individual algorithms. The results show that ensemble techniques lead to much better results compared to the individual predictions of the different methods. Ensembles techniques have consistently won a high percentage of competitions in kaggle [1]. An example to show the reason for ensembles being better predictors is, consider a correct prediction value of 80 W. The first method of the ensemble can predict 76 W, the second one 83 W, and the third one 81 W and when averaged the prediction is 80 W i.e. 0% error. This not only lowers the mean absolute prediction error but also the maximum predicted error. The downside of ensembles is that sometimes the minimum error can be a bit higher. Compared

to voting based methods of `RandomForest` which splits the attribute space, and `Bagging` which splits the training data set, our ensemble techniques takes advantage of several best performing individual methods. Our results show that significant benefits can be derived from the ensemble method with algorithms having different strengths in different regions of the application space.

Chapter 5

Summary and Future Work

5.1 Summary and Future Work

In this thesis, we have taken a two-fold approach to optimize the total cost of acquisition and ownership of heterogeneous computing systems. First is emphgreene device selection through device power and performance analysis. Second is predicting the power consumption of an application at different GPU configurations.

We have evaluated the performance and dynamic energy consumption of CPU and the integrated GPU with respect to the EDP metric for the special application domain of scientific visualization. We applied various filters to ParaView built-in data sets as well as to real-world applications. From the data, we see that modern CPU architectures such as Intel® Xeon® are highly optimized and outperform the integrated GPU when the dataset is smaller

and simpler as is the case with built-in data sets of cone and sphere. However, the integrated GPU outperforms the CPU in EDP for larger data sets and complex operations such as glyph. The performance and energy consumption for CPU and integrated GPU varies according to the dataset and the filters applied. For optimized energy and performance, visualization computation should toggle between the GPU and the CPU.

As future work, a similar analysis based on EDP can be done for other application domains with the CPU, the integrated GPU, and the discrete GPU. More extensive work can be done to characterize numerous other visualization filters. This will help in making greener high-performance architectural choices for HPC systems.

For power prediction, we explored several machine learning techniques to predict power consumption of an application at different frequencies. The prediction is based on applications resource utilization levels and the training dataset. Our evaluation demonstrated that `SMOreg` delivers best results with 4.5% MAE and `RandomForest` delivers more uniform results at different frequencies. We also demonstrated that power consumption of an application can be predicted reasonably reliably from the resource utilization level metrics of the application. We observe that more accurate prediction results can be obtained with the availability of better and more number of such metrics. Based on our evaluation of various statistical and machine learning methods we created and evaluated different ensemble predictors. The most accurate ensemble is a combination of `SMOreg`, `SLR`, and `REPTree` methods which reduced the mean absolute prediction error to 3.5% and maximum error to 11%.

A future extension of the power prediction can be to construct power and *performance*

models with a larger data set, a larger number of applications, more complex ensembles, and with better metrics as available on enterprise-class GPUs. We believe this will enable us to achieve higher accuracy in power prediction and lead to greener computation.

Bibliography

[1] Kaggle: Your home for data science. http://www.kaggle.com/.

[2] The intel xeon processor, 2017. https://www.intel.com/content/www/us/en/processors/xeon/xeon-autodesk-workstation-brief.html.

[3] TOP500 Supercomputer Site, 2017. http://www.top500.org.

[4] V. Adhinarayanan, W. chun Feng, J. Woodring, et al. On the greenness of in-situ and post-processing visualization pipelines. In *IEEE International Parallel and Distributed Processing Symposium Workshop (IPDPSW)*, 2015.

[5] V. Adhinarayanan and W. Feng. An automated framework for characterizing and subsetting gpgpu workloads. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2016.

[6] V. Adhinarayanan, W.-c. Feng, D. Rogers, et al. Characterizing and Modeling Power and Energy for Extreme-Scale In-situ Visualization. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2017.

[7] V. Adhinarayanan, B. Subramaniam, and W. Feng. Online power estimation of graphics processing units. In *16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2016.

[8] U. Ayachit. The paraview guide: A parallel visualization application, kitware isbn 978-1930934306, 2015.

[9] S. S. Baghsorkhi, M. Delahaye, S. J. Patel, W. D. Gropp, and W.-m. W. Hwu. An adaptive performance modeling tool for gpu architectures. In *ACM Sigplan Notices*, volume 45, 2010.

[10] P. E. Bailey, D. K. Lowenthal, V. Ravi, B. Rountree, M. Schulz, and B. R. de Supinski. Adaptive configuration selection for power-constrained heterogeneous systems. In *International Conference on Parallel Processing*, 2014.

[11] R. E. Bellman. *Adaptive control processes: a guided tour*. Princeton U. press, 2015.

[12] A. S. Berres, V. Adhinarayanan, T. Turton, et al. A Pipeline for Large Data Processing Using Regular Sampling for Unstructured Grids. Technical report, Los Alamos National Laboratory (LANL).

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

[14] R. Cochran, C. Hankendi, A. K. Coskun, and S. Reda. Pack & cap: adaptive dvfs and thread packing under power caps. In *Proceedings of the 44th annual IEEE/ACM international symposium on microarchitecture*, 2011.

[15] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, and J. S. Vetter. The scalable heterogeneous computing (shoc) benchmark suite. In *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, 2010.

[16] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le. Rapl: memory power estimation and capping. In *ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)*, 2010.

[17] L. Eeckhout. Computer architecture performance evaluation methods. *Synthesis Lectures on Computer Architecture*, 5(1), 2010.

[18] H. Esmaeilzadeh, E. Blem, R. St Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. In *ACM SIGARCH Computer Architecture News*, volume 39, 2011.

[19] W. Felter, K. Rajamani, T. Keller, and C. Rusu. A performance-conserving approach for reducing peak power consumption in server systems. In *Proceedings of the 19th ACM annual international conference on Supercomputing*, 2005.

[20] E. Frank, M. A. Hall, and I. H. Witten. The weka workbench. online appendix for: Data mining: Practical machine learning tools and techniques, morgan kaufmann, fourth edition, 2016.

[21] M. Gamell, I. Rodero, M. Parashar, et al. Exploring power behaviors and trade-offs of in-situ data analytics. In *IEEE/ACM International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2013.

[22] M. Gamell, I. Rodero, M. Parashar, and S. Poole. Exploring energy and performance behaviors of data-intensive scientific workflows on systems with deep memory hierarchies. In *20th IEEE International Conference on High Performance Computing (HiPC)*, 2013.

[23] N. Gholkar, F. Mueller, and B. Rountree. A power-aware cost model for hpc procurement. In *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2016.

[24] N. Gholkar, F. Mueller, and B. Rountree. Power Tuning HPC Jobs on Power-Constrained Systems. In *Proceedings of the 2016 International Conference on Parallel Architectures and Compilation (PACT)*, 2016.

[25] G. Haldeman, I. Rodero, M. Parashar, et al. Exploring Energy-Performance-Quality Tradeoffs for Scientific Workflows with In-situ Data Analyses. *Computer Science-Research and Development*, 2015.

[26] C.-h. Hsu and W. Feng. A power-aware run-time system for high-performance computing. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, 2005.

[27] W. Jia, K. A. Shaw, and M. Martonosi. Stargazer: Automated regression-based gpu design space exploration. In *IEEE International Symposium on Performance Analysis of Systems Software*, 2012.

[28] W. Jia, K. A. Shaw, and M. Martonosi. Starchart: hardware and software optimization using recursive partitioning regression trees. In *Proceedings of the 22nd IEEE international conference on Parallel architectures and compilation techniques*, 2013.

[29] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy. Improvements to platt's smo algorithm for svm classifier design. *Neural computation*, 13(3), 2001.

[30] S. Labasan, M. Larsen, and H. Childs. Exploring tradeoffs between power and performance for a scientific visualization algorithm. In *IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, 2015.

[31] S. Labasan, M. Larsen, H. Childs, and B. Rountree. PaViz: A Power-Adaptive Framework for Optimizing Visualization Performance. In *Proceedings of EuroGraphics Symposium on Parallel Graphics and Visualization (EGPGV)*, 2017.

[32] C. Lefurgy, X. Wang, and M. Ware. Power capping: a prelude to power shifting. *Cluster Computing*, 11(2), 2008.

[33] C. Luo and R. Suda. A performance and energy consumption analytical model for gpu. In *Proceedings of the IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*, 2011.

[34] K. Ma, X. Li, W. Chen, C. Zhang, and X. Wang. Greengpu: A holistic approach to energy efficiency in gpu-cpu heterogeneous architectures. In *41st IEEE International Conference on Parallel Processing (ICPP)*, 2012.

[35] A. Maghazeh, U. D. Bordoloi, P. Eles, and Z. Peng. General purpose computing on low-power embedded gpus: Has it come of age? In *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIII)*, 2013.

[36] X. Mei, Q. Wang, and X. Chu. A survey and measurement study of gpu dvfs on energy conservation. *Digital Communications and Networks*, 3(2), 2017.

[37] T. Patki, D. K. Lowenthal, B. Rountree, M. Schulz, and B. R. De Supinski. Exploring hardware overprovisioning in power-constrained, high performance computing. In *Proceedings of the 27th international ACM conference on International conference on supercomputing*, 2013.

[38] J. R. Quinlan. Simplifying decision trees. *International journal of man-machine studies*, 27(3), 1987.

[39] I. Rodero, M. Parashar, A. G. Landge, et al. Evaluation of In-situ Analysis Strategies at Scale for Power Efficiency and Scalability. In *16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2016.

[40] J. Shalf, S. Dosanjh, and J. Morrison. Exascale Computing Technology Challenges. In *International Conference on High Performance Computing for Computational Science*, 2010.

[41] B. Subramaniam and W. Feng. Statistical power and performance modeling for optimizing the energy efficiency of scientific computing. In *IEEE/ACM Int'l Conference on Green Computing and Communications (GreenCom), & Int'l Conference on Cyber, Physical and Social Computing (CPSCom)*, 2010.

[42] G. Wu, J. L. Greathouse, A. Lyashevsky, N. Jayasena, and D. Chiou. Gpgpu performance and power estimation using machine learning. In *IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, 2015.

[43] H. Zhang and H. Hoffmann. Maximizing performance under a power cap: A comparison of hardware, software, and hybrid techniques. *ACM SIGPLAN Notices*, 51(4), 2016.