Technical University of Munich

Data Innovation Lab

# Energy-Efficient Runtime in HPC Systems with Machine Learning

February 2019


| | |
|---|---|
| Authors | Gence Ozer, Sarthak Garg, Gabrielle Poerwawinata, Neda Davoudi |
| Mentors | Daniele Tafani (LRZ), Matthias Maiterth (Intel), Josef Weidendorfer (LRZ), Alessio Netti (LRZ) |
| Co-mentor | Michael Rauchensteiner (TUM) |
| Project Lead | Dr. Ricardo Acevedo Cabra (Mathematics Dept.) |
| Supervisor | Prof. Dr. Massimo Fornasier (Mathematics Dept.) |

# Contents


**1 Introduction** **2**

**2 Literature Review** **4**

**3 Project Plan** **5**

**4 Data Collection & Tools** **6**
4.1 Platform and Setup . . . 6
4.2 GEOPM . . . 6
4.2.1 What is GEOPM? . . . 6
4.2.2 Generating Training Data with GEOPMbench . . . 6
4.2.3 Test Case: GADGET . . . 9
4.3 DCDB . . . 9
4.3.1 Data Collection . . . 11
4.3.2 Data pre-processing . . . 11
4.3.3 Eliminating excessive leading points . . . 11
4.3.4 Fitting the DCDB data into GEOPM . . . 12
4.3.5 GEOPM and DCDB Final Integration Result . . . 13

**5 Data Exploration and Pre-processing** **14**
5.1 Datasets and Insights . . . 14
5.1.1 Nekbone application . . . 14
5.1.2 Random config. data - fixed frequency runs . . . 15
5.1.3 Random config. data - varying frequency runs . . . 15
5.2 Data pre-processing . . . 16
5.2.1 Outlier removal . . . 16
5.2.2 Time normalization(1 lag models) . . . 16
5.2.3 Time interpolation(multiple lag models) . . . 16
5.3 Feature generation . . . 17

**6 Models** **18**
6.1 Modelling approach 1 - one lag prediction . . . 19
6.1.1 Linear Regression on Fixed Frequency dataset . . . 19
6.1.2 Random forest regression on fixed frequency dataset . . . 20
6.1.3 Linear Regression on varying freq. dataset . . . 21
6.1.4 Nu Support vector Regression on varying freq. dataset . . . 22
6.1.5 Random Forest Regressor . . . 23
6.1.6 Region based linear regression . . . 23
6.2 Modelling approach 2 - multiple lag prediction . . . 24
6.2.1 Predicting Instructions retired . . . 25
6.2.2 The Frequency-Power lag . . . 25
6.2.3 Random forest on fixed frequency . . . 26
6.2.4 Random forest on varying frequency . . . 26
6.2.5 Merging dcdb data to the GEOPM features . . . 27

**7 Discussion & Future Work** **29**

**Abstract**

The energy efficiency problem for High Performance Computing Systems is considered to be one of the biggest obstacles in front of the exascale systems. The wasteful energy usage of these systems causes problems for the operating budgets and raises concerns for the environment. Due to the severity of the issue many people over the years, tried different approaches to tackle the problem. This project approaches the energy problem with Machine Learning point of view and implements a regression algorithm that is able to predict power and instruction retired metrics with the possible frequency settings in an online fashion using the information collected in the runtime. With the help of these metrics decision policies can be applied to select the optimal frequency setting that minimizes the energy. Our results show that we can predict these metrics on a realistic dynamic runtime setting within a relatively low error margin.

# 1 Introduction

Supercomputers are an important tool for humanity to uncover the mysteries of the universe. Some of the problems that are interest to supercomputing are protein folding, cosmological simulations, predicting climate change and even modeling the spread of swine flu. The resolution of these complex tasks requires large amounts of compute resources and, unlike our daily compute devices like desktop computers, tablets and mobile phones, supercomputers are specifically designed to fulfill the requirements of these problems. According to the recent surveys in 2018, even though our daily computers are becoming more and more capable with time, most of them still have processors comprising two or four cores[1]. On the other hand, supercomputers ranking in the Top 500 list can contain up to 10 million cores distributed on thousands of processors in the system[2, 3]. Furthermore, High Performance Computing (HPC) systems have significantly higher capacity memory size, which can reach Terra-bytes level, compared to the standard desktop computers which usually contains up to 16 GB of memory[1]. To complement the superior hardware, the HPC systems have access to premium performance optimization software developed by enterprises which can be used to enhance the efficiency of the applications.

As the hardware of HPC systems differs from the desktop computers, so does their software and applications. High Performance Computing is the practice of utilizing parallel programming to run advanced applications on the HPC systems efficiently. These applications are called High Performance Computing applications and they are specifically designed to take advantage of the parallel nature of the problem. It is typically achieved by dividing the problem into smaller independent computational components that can utilize all of the resources in the system. Each of the computational components are called Threads which can be executed simultaneously on the cores to find a small part of the final solution. Usually, all the independent results are collected in the final steps of the application to calculate the final answer. For the problems at a scale single threaded applications are not suitable whereas HPC becomes an enabler for the solution of these problems.

HPC systems are in high demand and often utilized by many scientists and researchers at the same time. Since the full capability of the HPC system is often excessive for the typical application, the available resources should be distributed among the users in an efficient manner. Therefore, these systems have a module running on top of all other applications called Scheduler that performs application queuing and resource distributing. After applications are allocated the necessary resources to start, Runtime system runs the application while performing application wide optimization.

The capability of HPC systems have high impact on the total power consumption. One of the fastest supercomputers in the world, Tianhe-2A, located in China, requires 18,482 kilowatt power to operate[3]. Running Tianhe-2A for an hour will cost as much energy as it would take to fully charge over 350 Tesla Model 3s. The Figure 1 depicts the historic power consumption data of the HPC system in Leibniz Supercomputing Centre. As it clearly shows, the power consumption of the HPC systems has been increasing yearly and it is predicted that the trend will continue in the upcoming years. As a result, as the capabilities of the systems increase further, providing the necessary power to operate these systems will become more challenging. This trend has clear implications on the feasibility of operating these systems considering that the operating costs will be a heavier burden on the institutions. On the other hand, as the world is working for a smaller carbon footprint

and more efficiency, the wasteful power usage of HPC systems concerns many in the HPC community. This phenomenon not only makes running the current systems more difficult but also makes it harder to justify the expansion of these systems' capabilities into the future exascale systems.

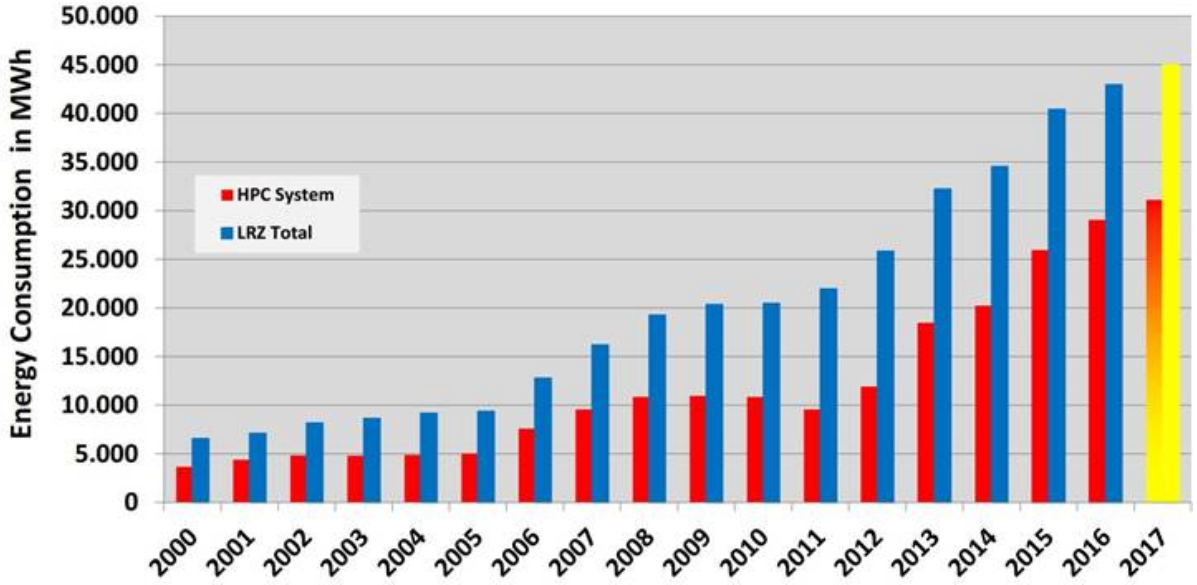


*Figure 1: Power consumption trend at LRZ over from 2000 until 2017.*

Over the past decade, the scientific and industrial communities have greatly contributed to address the challenge of energy consumption. The problem of excessive energy consumption of HPC systems can be potentially tackled at four different levels:

- **Building Infrastructure**: HPC centers can invest in better non-IT infrastructure to improve the efficiency of the systems and reduce costs. Possible examples are reduction of losses in the energy supply chain, use of better cooling solutions and reuse of the excess heat generated in the cooling process.
- **Hardware**: semiconductor companies, such as Intel and AMD, produce more power-efficient hardware, which is the result of improved process optimizations and breakthroughs in processor architectures. Data centers and HPC sites can adopt these technologies to contain energy costs.
- **Software**: energy efficiency could be further achieved by improving the software stack, that is the collection of software programs that operate and orchestrate the computational resources of the HPC system. Typical examples involves optimizations in operating systems, schedulers and runtime systems by implementing algorithms to distribute more efficiently the resources between users and decide on the optimal system configurations during application runs.
- **Application**: finally, scientific application developers could optimize their code to increase efficiency and performance, for example, by using the most efficient libraries and algorithms.

Each of the above mentioned approaches are very crucial and effective which is why they should be utilized together to achieve the maximum energy efficiency. However, when the approaches are considered individually it can be concluded that infrastructure and hardware solutions are not available to computer science community, since semiconductor companies and supercompute centers are the ones that can improve these factors. Application level solutions usually are very effective, but they require a lot of customization per application basis. The applications should usually have to be rewritten or heavily modified by an expert on the system to achieve the best performance and efficiency at the target HPC resource. On the other hand, software solutions, such as better scheduling and runtime systems, have effect on every application run on the system and allows better utilization and availability of the compute resources. As the background of project

contributors are considered, the possible research topic that we could tackle in this project was improving the software level solutions.

One of the important methods for energy optimization is CPU frequency setting, as it effects the total power directly. Setting the CPU frequency optimally can yield great benefit in terms of efficiency. A software solutions that utilizes this technique is currently being used in Leibniz Supercomputing Centre's SuperMUC is Load Leveler Energy Aware Scheduling software. When a new application is run on SuperMUC, the scheduler sets the default CPU frequency and collects statistics from application throughout the first runtime. After the first run completes, the scheduler tries to predict the optimal CPU frequency for the application using a linear model, which minimizes the energy without harming the performance severely. Afterwards, the optimal frequency can be set to increase system efficiency when next time the same application is run. The first downside of this approach is the fact that for each new application the scheduler only makes observation on the first run without any attempts for optimization. Secondly, the optimal frequency set on the later runs is static during the whole runtime and fails to capture the different characteristics within the runtime.

The goal of the project is addressing the above mentioned shortcomings by extending the GEOPM runtime system [1], which is an open-platform for co-designed energy solutions for HPC systems, to achieve better energy savings. The extension to the runtime system should be capable of changing the CPU frequency dynamically to the optimal setting by utilizing the information at the runtime. The optimal frequencies, which helps reducing the total energy cost, will be predicted by utilizing machine learning techniques. By building a model which can predict the optimal frequency in the next timestamp given the information from the previous timestamps, the energy savings can be achieved from the first runtime and the different characteristics within the application can be captured.

# 2 Literature Review

Over the last two decades, technology scaling and better performance has led to several hundreds of cores being integrated into one chip, resulting in a power wall. High power consumption increases power required for cooling and degrades the system reliability. Hence, minimizing the power consumption of HPC systems has become increasingly important for research communities for almost two decades.

Several ideas have emerged to tackle this problem. The dark silicon problem tackles it by operating only a fraction of cores on a chip at a time to avoid hotspots and reduce cooling costs [4]. But this approach becomes unviable for cost effective technology scaling [5]. Power management techniques have proven to be more promising as they limit the power to the chip/cores according to the required performance and preserve the scaling benefits. One of the most widely used method for power management is DVFS(Dynamic Voltage and Frequency Scaling), which can be used at various architecture granularities, such as node-level, chip-wide, and per core. It can be shown that optimizing the global energy consumption using DVFS is NP-hard problem[6]. The energy aware scheduling policy implemented in SuperMUC at LRZ is a static frequency scaling where a preset frequency is chosen for the whole application based on a linear regression model [7]. Lee et. al. (2015) uses dynamic control of voltage regulators with a reconfigurable power distribution network [8].

One of the most promising power management techniques use Reinforcement Learning (RL). The dynamic nature and the clear action-reward relation of online-energy optimization makes the RL a suitable candidate. Most of the methods employ either TD($\lambda$)-learning ([9], [10], [11]) or Q-learning([12], [13], [14]) as their models. Lin et. al. used TD($\lambda$)-learning to reduce server pool energy consumption while maintaining the job response time [9]. Tan et. al. used constrained Q-learning to get the best power management policy that gives the minimum power consumption without any prior information of workload [13]. Since, the state-space of the RL expands exponentially with the core count, most of the traditional RL models are limited with the single processor and small number of cores. There has further improvements on RL by reducing the complexity of the state space by applying Modular Reinforcement Learning[6]. Even though the results are promising,

[1] More detailed explanation on GEOPM can be found in Section 4.2.1

the online-training processes makes it impossible to train the models outside of a full system simulator.

An alternative approach to Reinforcement learning used in literature are Supervised Learning models. Jung et. al. used metrics such as occupancy state of a global service queue to make decisions on optimum policy using bayesian classifier [15]. Yang et. al. developed a runtime model using linear regression to map an application task on a computing resource during runtime, ensuring minimum energy consumption for a given application performance requirement[16]. Moghaddam et. al. use LSTM neural networks to estimate workload in the next control period, and select optimum frequency using DVFS to meet energy and performance constraints[17]. Another branch of research is employing time series analysis to characterize the history of an application and forecast them. The previous research show that time series analysis with Autoregressive moving average model (ARMA) and Singular Spectrum Analysis can be used with runtime traces to achieve good predictions [18]. Furthermore, more advanced models that can excel in capturing the historical data such as Long Short Term Memory (LSTM) are shown exhibit better performance in predicting the optimal frequency settings.

Although reinforcement learning has shown high potential in dynamic power management techniques, the actual implementation of such 'online' models on runtime systems are difficult and shows less promise in that direction. On the other hand, supervised learning techniques, based on runtime DVFS control and task mapping have proved to work, and it's easy to implement in an actual runtime system without too much overhead. To successfully implement the model in a runtime system, a framework is needed which can provide hardware counters and their accumulated summaries, and also provide lightweight and dynamic access to hardware settings which can be implemented over the whole application. GEOPM (Global Extensible Open Power Manager) offers such a lightweight runtime platform to implement optimal frequency policies during application execution. The motivation is to use GEOPM to implement frequency optimization policies on a 'pre-trained' supervised learning model, which can infer quick frequency decisions at runtime.

# 3 Project Plan

The project was conducted by a team of four TUM students with great contribution and help from three mentors from LRZ and one mentor from Intel. The project is supervised and lead by two representatives of the TUM Data Innovation Lab and lasted over four months of time. The guidance from Intel GEOPM developers was available to the students throughout the whole period. The project consists of the following work packages:

- Scientific literature review & choice of research methods, techniques and tools including data analysis algorithms and machine learning models.
- Acquisition, processing and analysis of application code data.
- Design of a model based on Machine Leaning / Deep Learning techniques that is capable of predicting optimal frequency settings for application runs.
- Validation of the conceived model.

The team contains members from different disciplines and the project workload is distributed according to the skills of the members. Gabrielle Poerwawinata and Gence Ozer were responsible for the HPC side of the project, specifically contributing on the collection of data from DCDB and GEOPM respectively. Sarthak Garg and Neda Davoudi took care of the statistical analysis of the data and on the development of the model through machine learning techniques.

Coordination within the group was performed mostly utilizing Slack software[19]. Furthermore, a Slack channel was setup for the project in the official GEOPM slack channel where the students could receive quick help / feedback from the GEOPM team. An project repository was created on Gitlab, where the team collaboratively worked on the application code. Furthermore, a wiki page[2] is setup to keep track of the documentation and the coordination of the group.

[2] https://gitlab.lrz.de/DILab/dil-geopm

# 4 Data Collection & Tools

## 4.1 Platform and Setup

The system used for running the applications and experiments is CoolMUC-3 provided by the Leibniz Supercomputing Centre [20].The cluster consists of 148 nodes, each of which has an Intel®Xeon Phi™CPU 7210 processor. Xeon Phi processors run at 1.3GHz nominal frequency with available frequencies ranging from 1.0GHz to 1.5GHz (when set to Turbo mode). Furthermore, it has 64 cores with 4 way hyper threading and total of 96 GB of DDR4 memory. The nodes run SUSE Linux Enterprise Server 12 SP3 as operating system and come with Intel compilers and performance libraries.

All the experiments mentioned in this paper are run on the two reserved nodes for the project with the specifications mentioned above. The C++ codes are compiled with the Intel toolchain including icc, icpc, ifort and Intel MPI (version 17.0.6). All the HPC applications are run with 62 MPI tasks to saturate the node fully. The most recent version of GEOPM at the start of the project, 0.6.0 release, was used throughout the entirety of the project. Moreover, the preprocessing and analysis codes are run with Intel Python 3.6.5 and Python Data Science Stack is used.

## 4.2 GEOPM

### 4.2.1 What is GEOPM?

The Global Extensible Open Power Manager (GEOPM) is an open source community driven software solution, initiated by Intel, that aims to address energy problem in High Performance Computing (HPC) systems. As it is stated in the GEOPM project description, the primary goal of the project is to provide an open platform for community collaboration and research on co-designed energy management solutions to close the energy efficiency gap. One of the simple use cases of GEOPM framework is reading the Hardware counters, which are special registers that tracks hardware activities, and setting the hardware controls on a particular compute node. Furthermore, GEOPM generates runtime traces populated with hardware counters' readings and reports which can be used to understand and analyze the applications' behavior. A more advanced use case of GEOPM is to utilize agents to dynamically coordinate all the compute resources used by setting dynamic power limits per node in a HPC application to achieve optimal efficiency. Experiments showed that GEOPM can achieve up to 30% improvement in time-to-solution and energy consumption metrics on production systems at scale[21, 22].

GEOPM design is based on Agents which are control algorithms that can monitor applications using hardware counters' readings during runtime and act accordingly to optimize its goal by modifying hardware controls. GEOPM version 0.6.0 has three default agents which can perform runtime optimization namely: energy efficient, power governor and power efficient agent. However, what makes GEOPM powerful is that it allows one to easily extend the capability of the framework by implementing new agents. Therefore, more advanced control strategies might be integrated using agents which can utilize machine learning models as decision makers to optimize the runtimes in a more efficient and robust manner.

### 4.2.2 Generating Training Data with GEOPMbench

One of the first steps of training a machine learning model is to acquire the training data. The key factor to consider when collecting the training data is to have an adequate amount of data which reflects realistic conditions. As our goal in the project, as stated in Section 1, was to find the optimal frequency with machine learning using hardware counter readings, we needed to collect trace files which contained hardware counters' values read in runtime.

Every HPC applications' characteristics as a whole differ at runtime due to differences in the application code. As a result, the optimal frequency settings for one application is generally irrelevant for the others. However, small portions of the application that show similar characteristics, which are called Regions, exhibit similar behavior for different applications. Each application's characteristics are defined by how these

regions come together. Since our machine learning models should generalize well on many different applications, they should be able to capture these different region constructions. Therefore, we had to collect numerous traces that would ideally contain all the mentioned region changes. One of the methods for achieving such diversity on trace files is to run as many different HPC applications as possible, such as the Nekbone and AMG benchmarks[23, 24]. However, capturing all the region transitions is a difficult tasks and it would require tens of different applications. Hence, it is not feasible due to necessary work and time required to compile and adjust the codes to work properly on our platform.

Considering all these difficulties of the above-mentioned approach, we decided to simulate the wide variety of real HPC applications to capture all the region changes using an artificial application. With the version 0.6.0 GEOPM comes with **GEOPMbench** software which is a highly customizable artificial HPC application and it can be modified to mimic different behaviors. GEOPMbench comes with the following different regions:

- **Dgemm Region**: compute bounded region, performs dense matrix-matrix multiplication using the Intel MKL library[25].
- **Stream Region**: memory bound region, performs element-wise addition on two arrays and saves the result in a new array.
- **All2All Region**: network bound region, performs all to all reduction with standard MPI functions.
- **Sleep Region**: frequency-invariant, sleeps for a definite amount of time.
- **Spin Region**: performs busy waiting for a definite amount of time.

With GEOPMbench one can define a list of regions, their respective difficulties (big-o) and total repetition time (loop count) to create the desired application. Theoretically, region difficulty of 1.0 should last around 1 second in runtime. To illustrate the case, the example configuration on Listing 1 should have theoretic runtime around 10 seconds where each repetation should take approximately 5 seconds. However, the theoretical values are calculated according to the Intel GEOPM team's development environment. Since our development system was different than the GEOPM team's in terms of hardware and configuration, our results were not reflecting the theoretical expectations. Therefore, we had to re-calibrate the regions' difficulties to last approximately the same amount of time. Sleep region remained unchanged since the sleep command does not show variance in different machines. We scaled up the Dgemm difficulty by 3 to match with the Sleep region. Running the Stream region was more challenging since the region's difficulty is linearly correlated with the amount of memory allocated. On default settings the stream region reserves 11 GB memory per core which will result in over 700GB of memory for 62 core runs. Therefore, we scaled down the memory difficulty by 25 to fit in the available memory of one node of CoolMUC3. However, one issue that we observed after the mandatory scaling is that with the available memory stream region lasts half the time of the other regions. Finally, we wanted to include All2All region in our experiments as well, but it kept crashing on the runtime during the 62 core runs. Hence, we had to leave the network bounded region out in our experiments.

*Listing 1: Example Configuration*

```
{
  "loop-count": 2,
  "region": ["dgemm", "stream", "sleep", "
      dgemm"],
  "big-o": [1.2, 1.2, 1.2, 1.3]
}
```

*Listing 2: Scaled Configuration*

```
{
  "loop-count": 2,
  "region": ["dgemm", "stream", "sleep", "
      dgemm"],
  "big-o": [3.6, 0.06, 1.2, 3.9]
}
```

After we scaled the regions to our platform, we performed test runs to understand how the regions behave in different CPU frequencies. The energy trace of the example configurations showed that each region has a distinct energy behavior. The Figures 2 and 3 shows two normalized energy traces of the example configuration at Listing 2 with 1.1GHz and 1.5Ghz respectively. The black points that shows dgemm regions are computationally bounded where the processor spends the most energy to perform large number of operations. In addition to compute bound region, we observe red and yellow regions which indicate application

initialization and memory bound regions respectively. These regions spend less energy compared to compute bound region since they utilize the CPU less. Finally, the sleep regions displayed in pink color spend the least energy among all regions since they almost perform no CPU operations. Also, when the 1.1 GHz run compared to 1.5 GHz one can notice that energy increased almost linearly except the sleep region.

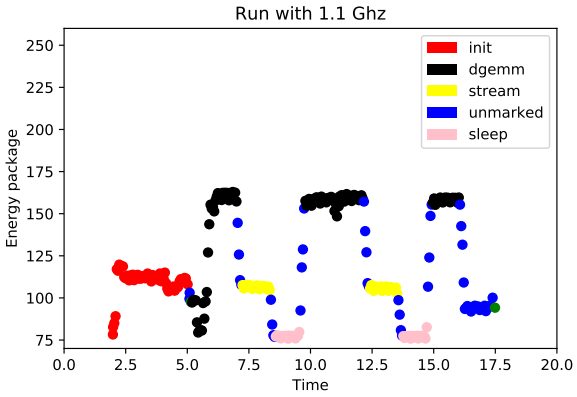


*Figure 2: Time normalized energy trace of e.g configuration run with 1.1 GHz*

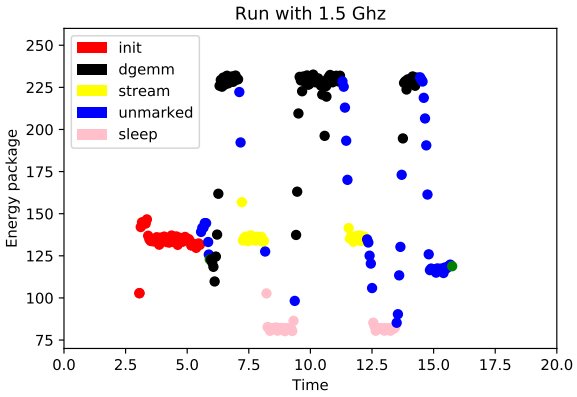


*Figure 3: Time normalized energy trace of e.g configuration run with 1.5 GHz*

In addition to distinct region behaviors, we also noticed that due to runtime inconsistencies, data sampling rate was varying throughout the runtime. As it can be seen on the Figure 4, the time difference between each sample is concentrated around the control loop time of 50ms. However, there are numerous points close to 0.04 and 0.06 and few other points in the very extreme sampling times. The inconsistencies in the sampling rate reflects to the energy traces as in Figure 5 where very high and very low energy values can be observed. Since we are collecting the data from a live system, the sampling variance is expected and unavoidable. Therefore, before feeding the data to the the machine learning models, these inconsistencies should be eliminated with data analysis techniques such as normalization or interpolation.

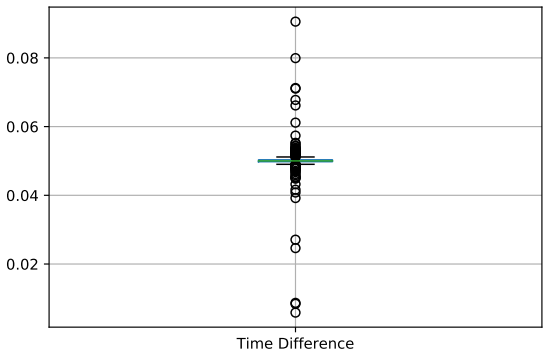


*Figure 4: Sampling variance*

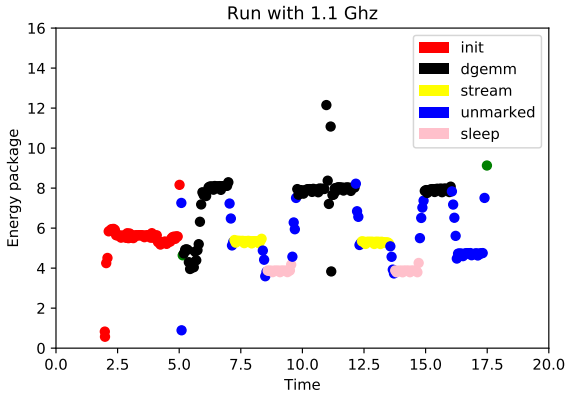


*Figure 5: Outliers due to time sampling rate*

As we stated earlier, our data should reflect real HPC runtimes where the control agents will modify the frequency to perform optimization. To capture the dynamically varying frequency behavior, we implemented a GEOPM agent which dynamically changes the frequency on control loops (50ms) . To experiment with different strategies we used to following approaches:

- Frequency doesn't change on the runtime but same application run multiple times with different fixed frequencies (exploration purposes)

- Frequency 80% likely to change on every control loop
- Frequency changing in every 3 control loops

In order to have sufficient sized training set, we generated 300 random configuration files which include Dgemm, Stream and Sleep regions. These configurations are generated by randomly choosing four regions from the regions three regions listed, where each regions difficulty is randomized between 1 to 1.5 seconds. Finally, we've selected a repetition count between 1 and 5 to vary the length of the total application. To avoid runtime issues we did not allow the configurations where the total stream region difficulty surpasses the memory limits. With the above mentioned dynamic agent, we ran the each configuration 3 times which resulted in 900 runtime trace files. Furthermore, we ran 20 more configurations in similar fashion using the second reserved node to compare the characteristics of different nodes. The data from the second node will be used as a test case to measure the success of the model.

### 4.2.3 Test Case: GADGET

Training the machine learning model on artificial benchmarks can be more comprehensive and convenient. However, actual HPC application can have different characteristics compared to the simulated ones. Therefore, good model performance on artificially generated training set should also be verified with actual HPC applications to asses the real world performance. Therefore, one important test case for our machine learning models is to testing them with real application traces after training them on the data generated from artificial benchmarks. Achieving good performance on the data generated from an actual applications is a solid indicator of the success of the model based agent implementations in the future. Hence, we selected GADGET[3] as our test application which is a software for cosmological N-body / Smoothed-particle hydrodynamics simulation developed at Max Planck Institute for Astrophysics [26]. We generated the data with our agent as in training data and used 8 MPI tasks and 8 OpenMP threads to maximize performance.

In our system GADGET application takes around 2 minutes to complete and it is a memory bounded application with occasional computation spikes. As the GADGET energy trace depicted in Figure 6.2.4 shows, the application behaves slightly differently from our training data, where the region transitions are less frequent.

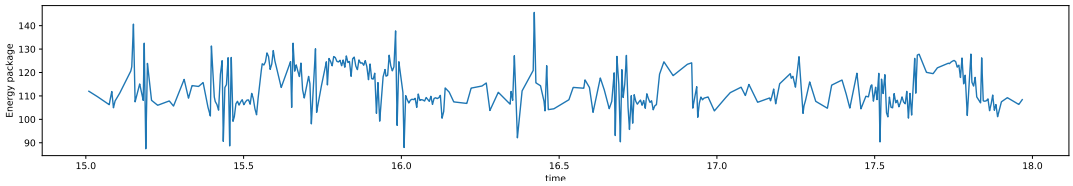


*Figure 6: Small fragment from GADGET runtime trace between 15 to 18 seconds*

## 4.3 DCDB

GEOPM has only support for non-programmable hardware counters which limits the amount of data that we can collect. Programmable counters such as cache misses, branch misses also carries important insights for optimizing the energy efficiency. Therefore, by augmenting GEOPM dataset with further hardware counter readings, machine learning algorithms can learn better. Therefore, we looked for additional tools to complement GEOPM in the data collection process. DCDB is an experimental tool developed by LRZ which used for monitoring and collecting events counter, driver information and processes information. The simple overview of how DCDB works as a monitoring system can be described as below:

1. Initiate Pusher before executing an application. Pusher collects the sensor data from a different file system in Linux kernel. Pusher will gather data as the specified time interval in DCDB querying command.

[3] Special thanks to Dr. Luigi Iapichino

2. Data collected by the Pusher will be published to the collect agent to be stored persistently in Apacahe Cassandra (NoSQL database).

3. Once the application is finished, user can collect the data based on naming convention. <hostname>.cpui.<metric> or <hostname>.<metric>

For CoolMUC3, DCDB monitors perf-events, sysfs data and procfs metrics. Perf-events collects sensor data per CPU-core, while the other metrics are collected on the node level summary.

- **Perf Events** as abbreviation for performance events, measures events coming from different resources that are software events, hardware events. In addition to event counters, perf-events also includes system profiling[27].

| | | | |
|---|---|---|---|
| **HW-events** | instructions<br>ref-cycles<br>references | cpu-cycles<br>branch-instructions<br>misses | cpu-migration<br>branch-misses<br>page-faults |
| **SW-events** | context-switches | | |

*Table 1: List of perf-events metrics*

As it is mentioned in Section 4.2.2 the applications are utilizing 62 cores. The workload measurements for perf-events is per CPU granularity, meaning that sensor data from 62 cores values were collected. Final value per metric achieved by reducing the core level measurements by summation.

- **Sysfs** is pseudo file system from Linux kernel. It provides information from associated device drivers as listed in Table 2. Here, sysfs provides implementation into temperature of the kernel and host fabric interface. The increasing temperature might suggest with the increasing frequency.

| | | | |
|---|---|---|---|
| knltemp | energy | hfi0temp | hfi1temp |

*Table 2: List of sysfs metrics*

- **Procfs** filesystem presents information about running processes. Process structure is a collection of several components [28] including memory, registers and other devices. As displayed in table 3 procfs contains many metrics different metrics. However, the metrics that might be valuable in the analysis section of the project are MemFree, MemAvailable, Buffers and Cached. These metrics were expected to help better classification of the memory intensive regions in the applications.

| | | | |
|---|---|---|---|
| col_nice | col_system | col_idle | col_iowait |
| intr | ctxt | btime | softirq |
| MemFree | MemAvailable | Buffers | Cached |
| Active | Inactive | AnonPages | KernelStack |
| PageTables | VmallocTotal | VmallocUsed | VmallocChunk |
| HardwareCorrupted | pgpgin | pgpgout | pgfree |
| pgfault | thp_fault_fallback | thp_collapse_alloc | thp_split |
| thp_collapse_alloc_failed | thp_zero_page_alloc | thp_zero_page_alloc_failed | |

*Table 3: List of procfs metrics*

### 4.3.1 Data Collection

As mentioned previously in Section 4.2.2, GEOPMbench is used as main application to perform data collection with the configurations mentioned in Section 4.1. DCDB was started before running the GEOPMbench application by activating DCDB environment for each bash shell. As we have more data on DCDB than GEOPM the data retrieval difficulty increases more rapidly than GEOPM. From Section4.2.2, we had 920 samples in total that needed fetching from the DCDB system. To retrieve the specified amount of data in reasonable timeframe, we created 3 different kind of bash scripts, where each of them handle different sensor data group (Table 1, 2 and 3). This made our work easier and faster, because we executed those scripts in parallel.
It means we divide query iteration for one node data collection into several iteration parts and run them in different process. Large portion of time is mostly spent while gathering perf-events data since the size is much larger compared to the other groups. For one sample, it took around 8 minutes to be fully fetched and we had 920 samples data in overall. All the features that must be collected within perf-events group are much less compared to the one from procfs. This makes sense since the query returns total number of 62*10*920 $\approx$ 570K entries for both nodes.

### 4.3.2 Data pre-processing

Pandas data analysis library was used for data parsing, manipulation and pre-processing. We need all metrics that we obtained in Section4.3 appended as new additional columns to GEOPM traces.

After collection part of DCDB has finished, we continued with pre-processing phase. The objective is to create the same structure as GEOPM trace file, for easier access and convenience in the analysis phase. We treated each DCDB data group of measurement differently. For each GEOPM sample, we had 4 files that need to be processed further which consist of GEOPM, perf_events, procfs and sysfs trace files. These files were merged into a single unit. However, trace files from different data sources come with different timestamps. DCDB time stamps are absolute UNIX timestamps whereas GEOPM timestamps are relative time according the start of the application. Therefore, the timestamps needed to be standardized in order to merge the fields correctly. In order to convert the relative GEOPM time to absolute UNIX timestamps, we required a reference point. Therefore, we retrieved the time before running and after completion of each application. Then using the reference timestamps we converted the GEOPM timestamps into compatible form with DCDB.
The integration part was done sequentially. First, we merged metrics from GEOPM with perf_events, then continue with sysfs and procfs. In order to align the timestamps, we compared same instructions-retired metric collected from both GEOPM and DCDB. This metrics was supposed to have same behavior and pattern, so that we can use them as our baseline in aligning time-series from different data sources.

### 4.3.3 Eliminating excessive leading points

In most of the cases, samples generated from DCDB have excessive leading points that come earlier than the one from GEOPM. A sample trace where this phenomena is clearly visible is shown in Figure 7. In the trace DCDB and GEOPM were stacked together before any adjustment with respect to their time-series and it can sees, there is no synchronization between red and blue curves. The idea of alignment of timestamps comes from modifying these curves to create the same pattern. Hence, we needed to define a starting point, which symbolize by blue dot, such that we only keep DCDB data points from that point onward and remove the rest. Concurrent curves in Figure 7 can be achieved by making the green and blue dot be placed by the same time point. To address this problem, we further define a parameter which will be used as elimination point of DCDB data (all points before the blue dot).

After observing the behavior of more than 10 trace files dataset, it is concluded that the GEOPM timestamp is lagging behind DCDB by 1.7 seconds. The rearrangement of the timeseries will result in loss of some points in the beginning of DCDB data that has timestamp value less than GEOPM initial timestamp plus the eliminating parameter of 1.7 seconds. Hence, all data points before blue dot will be removed and the rest will be kept intact. Indeed we need further process, where we shifted the DCDB time-series along with its value, in order to have almost equal time interval between data sources where 2 dots previously mentioned were placed in equal x-axis coordinate.

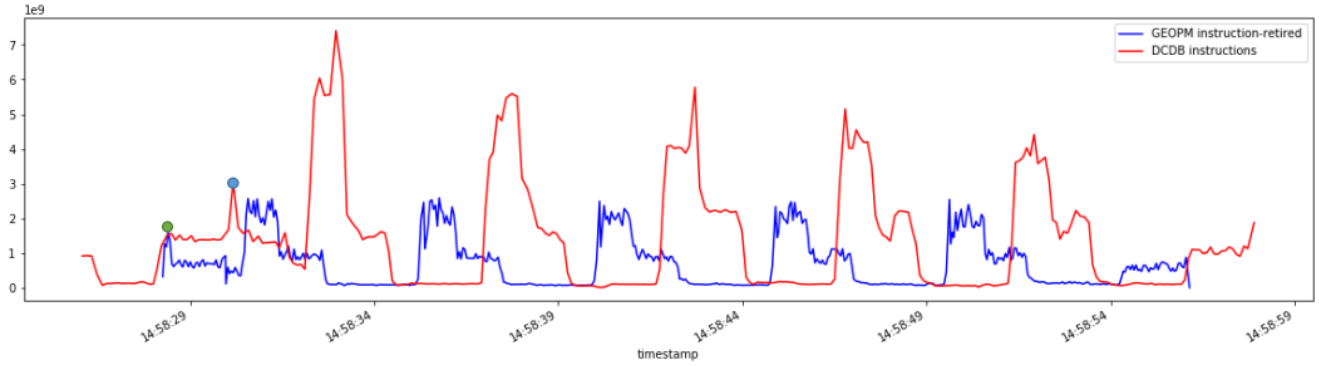


*Figure 7: Illustration of instruction sets between 2 sources before time-series alignment*

#### 4.3.4 Fitting the DCDB data into GEOPM

By this step we already synchronized DCDB and GEOPM time-series interval. However, since the DCDB samples the hardware counters less frequent than GEOPM (100ms) we need to perform further processing to merge each data point in one timestamp, therefore we picked time-series provided by GEOPM to become our only time-series and then to fit metrics from DCDB accordingly. Super-sampling technique was used at first to decide DCDB values that fits certain time point of GEOPM. This technique picks the DCDB value which has smallest time difference with specific GEOPM sample.
Another aspect which affects good data reconstruction was sampling rate granularity from these data sources. Sampling rate means referes to the the number of samples collected per second. Initially, DCDB sampling rate was 1 seconds, while GEOPM is sampling with 50ms. If we do mapping for both, several points from source which has higher sampling rate are going to be mapped to same point from other source. This imbalance resulted in reconstructed DCDB data run in flat line after integrating it with GEOPM Figure 8. Afterwards

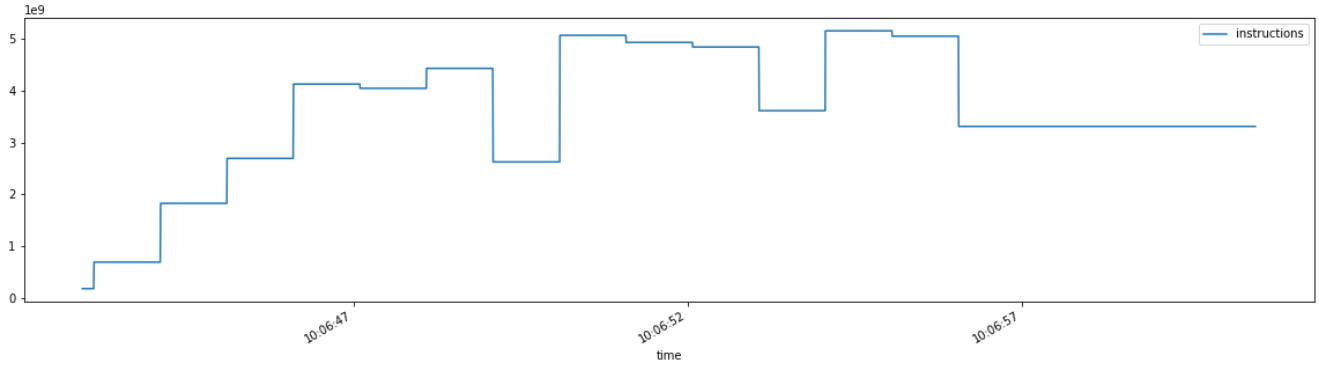


*Figure 8: Reconstruction with lower sampling rate and supersampling*

with help of our mentors we increased the sampling rate for DCDB to achieve closer granularity between GEOPM. Sampling rate went from 1seconds to 100ms. As the results are shown in Figure 9, the resulting time series became closer to the time series that we observed in GEOPM .
Second technique that we use as our final method in reconstructing data is linear interpolation. Interpolation creates new data point in a certain interval. Assume we want to create a new DCDB data points based on timestamp of GEOPM. The results we obtained by using interpolation were better than the previous one Figure 10. Interpolation technique made each data point of GEOPM will have different mapping to DCDB. Hence the reconstructed DCDB data Figure 10 does not have slightly flat line compare to Figure 9.

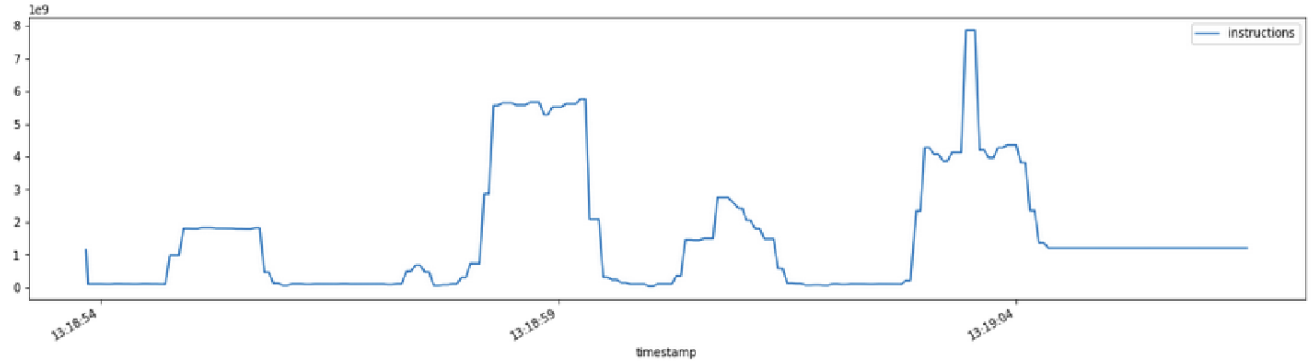


*Figure 9: Reconstruction with higher sampling rate and supersampling*

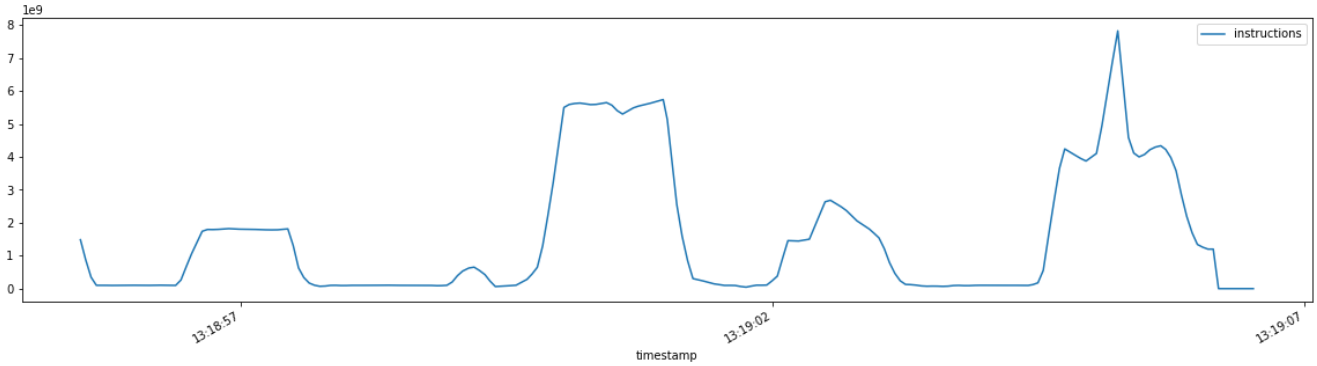


*Figure 10: Reconstruction with higher sampling rate and interpolation*

### 4.3.5 GEOPM and DCDB Final Integration Result

After thorough steps which include removing preliminary points, reconstructing DCDB into GEOPM using interpolation and synchronize two data sources, we can get final integration part as shown in Figure 11. Good quality of data sources integration is based on curves coincidence Figure 11. In a few cases, usually at preliminary and closing samples, two curves shows some time gap. As discussed in Section 4.3.3 we relied only to a single elimination parameter value and applied this parameter to the whole pre-processing stage. Ideally, it would be better if we could adapt the filtering values on per sample basis which would result in a more robust merge.

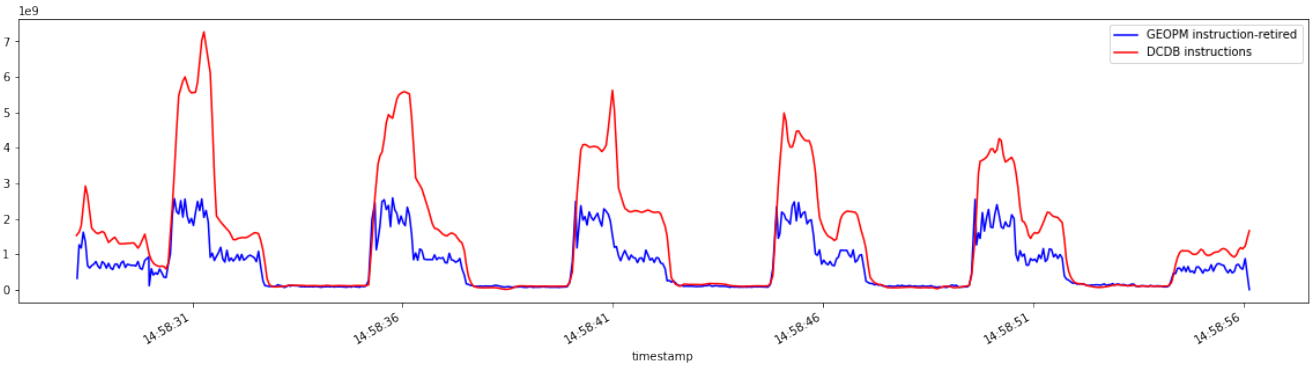


*Figure 11: Final Integration*

# 5 Data Exploration and Pre-processing

The collected data are the traces containing various hardware and application related metrics sampled in time. A trace contain node level metrics for a particular application. The data generation process of these traces are discussed in detail in Section 4.2.2. Over the course of this project, various datasets were used. Each of these datasets and insights generated from them is discussed Section 5.1. To make these datasets well-conditioned and usable for our models, various pre-processing steps were applied, which are discussed in Section 5.2.

## 5.1 Datasets and Insights

At the initial stages of the project, a dataset based on Nekbone application in GEOPM was made available by the mentors to peek into the trace files and get familiarized with the metrics available in them. For our machine learning model, the datasets generated are of two types; one where the whole application is run on constant frequency and the other where the frequency was changed during runtime so as to imitate the application behavior in terms of hardware metrics such as power, instructions retired, frequency.

### 5.1.1 Nekbone application

This dataset contains traces of an application run using two GEOPM agents, namely power governer and power balancer, which are used for runtime energy efficiency. We have a total of 180 traces generated from 10 compute nodes with 62 threads on each node. Each trace contains hardware metrics such as energy, power, thread cycles, along with GEOPM application metrics such as region-id, runtime, etc. Table 4 and 5 summarizes the list of metrics available in a trace file. The trace file contains these metrics enumerated for various timestamps corresponding to the local time of the application.

| **Hardware metrics** | **Description** |
|---|---|
| Package energy | Total energy of all sockets of the node |
| DRAM energy | Total energy of all DRAM of the node |
| Package power | Total power of all sockets of the node |
| DRAM energy | Total power of all DRAM of the node |
| Frequency | Average frequency of all cores. |
| cycles-thread | Average clock cycles executed by the cores since the beginning of the execution |
| cycles-reference | Average clock reference cycles since the beginning of the execution |

*Table 4: Hardware related metrics from GEOPM*

| **GEOPM application metrics** | **Description** |
|---|---|
| Region ID | 64-bit integer ID of the region that all ranks are running. If all ranks are not in the same region, it will be recorded as unmarked. Epoch boundaries are recorded as entry and exit to the epoch region. |
| region progress | minimum per-rank reported progress through the current region. (0 or 1) |
| region runtime | maximum per-rank last recorded runtime for the current region. |
| power limit, policy power cap, policy step count, policy max epoch runtime, policy power slack, policy power limit. | GEOPM policy related metrics |

*Table 5: Application related metrics from GEOPM*

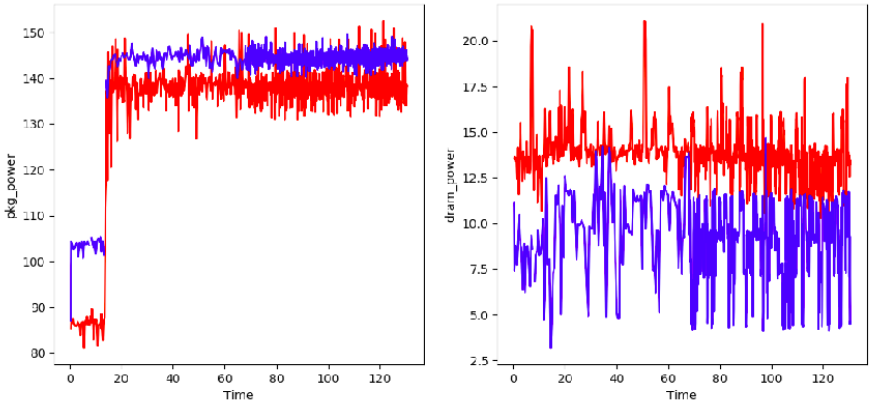


*Figure 12: different nodes running the same Nekbone application. Left-package power, Right-DRAM power*

We use all the hardware metrics for our prediction as they give us valuable information about the current and previous states in an application during runtime.Out of the GEOPM application performance measures, region-ID is a particularly useful metric telling in which region the application is running. All other GEOPM related metrics were removed from the ML model during further data generation. Region-ID is difficult to directly use in our machine learning models as it is specified by the user manually and they hold no tangible information about the type of region as the user might fail to characterize the regions correctly. Although when we generate data through GEOPMbench application, these region-ID's directly tell us the region we are in, for example, stream or dgemm. Instead, we use region-ID to evaluate our ML model or get insights into the behavior of models/data over various regions.

Figure 12 shows package and dram power on different nodes of the same application. It can be seen that the performance of the same application varies significantly over various nodes.So, our runtime model should address the variability of performance on different nodes.

### 5.1.2 Random config. data - fixed frequency runs

The dataset contains all the above hardware metrics and region-id along with an additional metric -instructions retired. This metric gives the total number of instructions executed by all sockets(CPU) on the node. Each trace is generated using a randomly using This data generation process is discussed in detail in Section 4.2.2 Each configuration is run 3 times on a single node of 62 cores, giving us a total of 900 traces.

Since the data is time series, the autocorrelation of the sequence is an important property to find. Figure 13 shows the partial auto-correlation for various input metrics for 20 lags. It is clear that for fixed frequency runs, only 2-4 lags are sufficient to describe the value in the sequence.

### 5.1.3 Random config. data - varying frequency runs

Our machine learning model should give actionable knowledge about the optimal frequency setting using the history of the collected metrics. This history then becomes the testing data for our model. Since, the history will contain the metrics with varying frequency, we should use data where frequencies are varying randomly throughout an application. Further details on data generation process can be found in Section 4.2.2. The partial autocorrelation plots look similar to the fixed frequency runs, where 2-4 lags are sufficient to describe the value of the metrics. Two kind of varying frequency data is used in the models. These are:

- frequency 80% likely to change in text sample.(2 nodes, 900 traces each)
- frequency changing after every 3 samples.(300 traces)

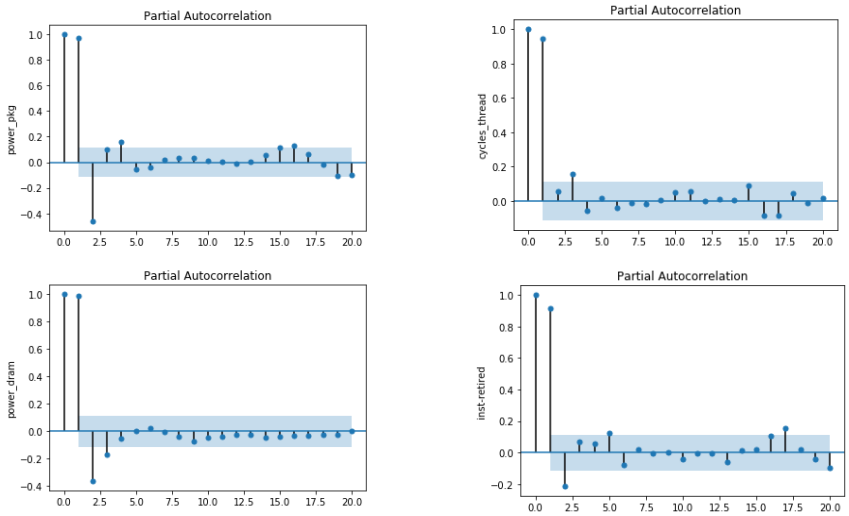


*Figure 13: Partial autocorrelation of 20 lags for constant frequency data, top left-package power, bottom left-dram power, top right-thread cycles, bottom right -instructions*

## 5.2 Data pre-processing

Cleaning and normalization is an important step to get the data which will produce a good machine learning model. The data pre-processing was done using Pandas(python), an open source tool which provides data structures and analysis operations. It's a popular choice for pre-processing and analysis of time series data. The data is stored in simple row-column format. Each column contains each metric listed in time. A row lists all the metrics at a particular timestamp. The details of each pre-processing step is described below.

### 5.2.1 Outlier removal

To get well-conditioned data for our models, the z-score estimate is used. Z-score of a data point in a column is the number of standard deviations from the mean data point. All the data points which lie outside -3<z<3 are considered as outliers and are removed. This operation was done column-wise, That is data in a row is removed from all columns if any column value is an outlier. Outliers account for 4% of total samples in our dataset.

### 5.2.2 Time normalization(1 lag models)

The data generated from GEOPM is not uniformly sampled in time. This is discussed in Section 4.2.2, in which Figure 4 depicts the sampling variance observed in our data. Some hardware metrics from GEOPM are accumulated over time. These are package energy, dram energy, thread cycles, reference cycles and instructions retired. These accumulated metrics are converted to interval based metrics using differencing. Since the time intervals between successive samples is not fixed(sampling variability), these difference metrics need to be made independent of time. This is done by normalizing these difference metrics with the time intervals. Figure 14 shows the package and dram energy after normalization and how they compare to the corresponding power.

After time normalization, the energy and power are seen to be equivalent as can be seen in Figure 14. Also, power (for both package and dram) is more smooth and well behaved than the corresponding energy. Hence, package and dram power are a better metric for ML model than the corresponding energy.

### 5.2.3 Time interpolation(multiple lag models)

As will be discussed later in Section 6.2, we need time series data which is uniformly spaced in time. This is not the case as the time interval is not a constant as seen from Figure 4. Hence, linear interpolation is used

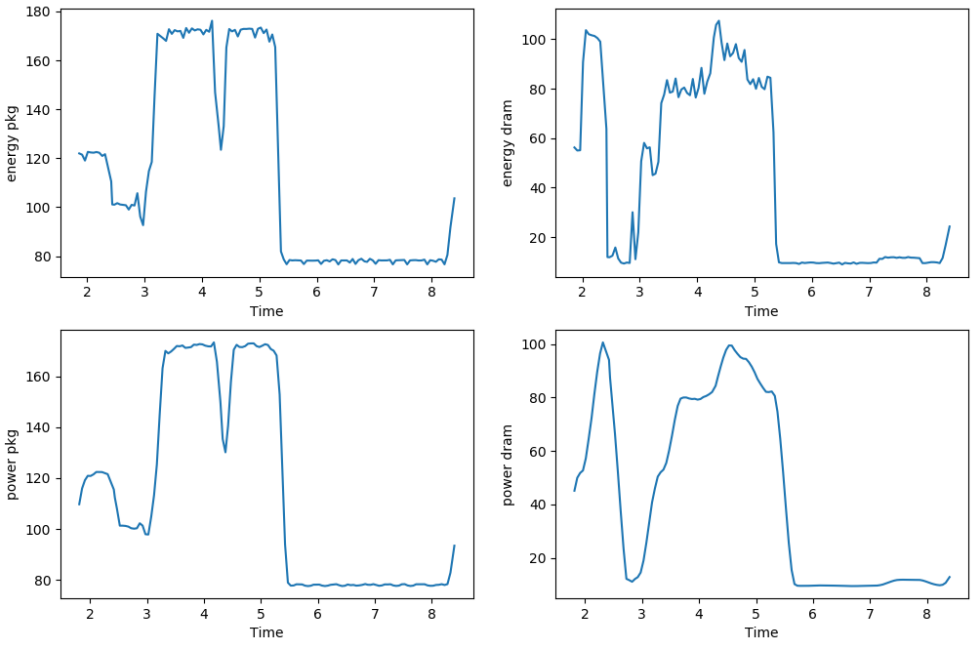


*Figure 14: Comparison between Power and Energy time series. top left- normalized package energy, bottom left- package power, top right- normalized dram energy, bottom right- dram power*

to sample at a fixed time interval. This time interval is calculated on per trace basis, which is given by the mean value of all the time intervals(which is approximately 0.05). This is not taken as a constant value apriori because we want to maintain the same number of samples per trace before and after interpolation. This is done to ensure the same mean frequency values over a moving window before and after interpolation.

## 5.3 Feature generation

As will be discussed in 6.2, one of the ways to use i.i.d. analysis for sequential(multiple lag) data is to convert the sequence into statistical features representing the properties of this sequence, so as to retain predictive power. This brings us to the choice of 'good' features to be used in our model. A good feature for our model should satisfy these two conditions:

- Cheap to compute: Since in a real application, these features have to be generated in real-time. Hence, they should be computationally inexpensive.
- Fit to problem: Choose features which are a good match for our problem and characterize the series.

Keeping these two conditions in mind, basic statistical measures like mean, standard, deviation, etc., were chosen as features. These statistical measures then treat this sequence of numbers as the realization of a random variable, based on which various expectations are computed. The features are generared in python using pandas' rolling functions. The features chosen for our machine learning model are listed below.

- Exponentially weighted mean:
  Mean of the sequence weighted exponentially high towards the end. This gives the mean behaviour of the metric, giving recent values more weightage.
- Exponentially weighted gradient
  Mean of the gradients weighted exponentially high towards the end. This can predict the 'trend' in data. For our problem, high gradients can occur during region changes. Hence, this metric might predict region changing behavior.
- Standard Deviation

- Skewness
  A measure of the asymmetry of the probability distribution of a real-valued random variable about its mean. It is given by the standardised third moment of the random variable.
- Kurtosis
  A measure of the "tailedness" of the probability distribution of a real-valued random variable. It is given by the standardised fourth moment of the random variable.
- Quantile - 0.25
- Quantile - 0.5
- Quantile - 0.75
- Sum of differences
  Absolute sum of differences between the consecutive points in the sequence.
- Sample Entropy
  A modification of approximate entropy, used for assessing the complexity/unpredictability of sequence.

# 6 Models

The goal of our model is to predict the 'optimum' frequency, which can be set during runtime, for an application running on HPC. Literature research leads us to two kinds of approaches used for such models: Supervised learning and Re-inforcement learning. Most of the literature on re-inforcement learning relied on using simulators to train the models. But the realization of such models in a real setting suffers from high computational overhead, needed for training the models at runtime. Hence, a supervised learning approach with a pre-trained model is chosen for this project. The model is trained 'apriori', and is later used to make runtime predictions of optimal frequency setting. The input for this model will be the trace history at any time when the optimal frequency decision is to be made. This history will be collected using runtime applications like GEOPM.

**Choice between classification and regression**
Classification problem: The model can detect and classify the current state of the application into characteristic regions like compute bound(dgemm), memory bound(stream), initialization, etc., then choose an optimal frequency policy tailor-made for that region. But, in an actual application, the region is 'continuous' and difficult to classify, leading to poor generalization or limited applicability. Hence, classification is not a good choice for this problem.
Regression Problem: Model traces as time series and 'forecast' the value of hardware metrics like power, instructions retired, etc as a function of history values, then choose the optimal frequency based on the dependence of forecasted values on frequency. Regression is a good choice for this problem.

Using the trace history of an application as input, our aim would be to predict(forecast) hardware metrics, which becomes our output. Hence, the output of our model are the future values of various hardware metrics. Since the frequency directly affects power, the future value of package and DRAM power are our outputs.

**What is a 'good' model for this problem?**
Since we want to predict the powers as a function of frequency, a good model should have the following characteristics:
**Objective 1**: The model generalizes well over the whole data. This can be tested using validation accuracy and residual plots.
**Objective 2**: The predicted package(cpu) power should be correlated with the frequency like in a real application. This is needed to make optimal frequency decisions.

## 6.1 Modelling approach 1 - one lag prediction

For both fixed and varying frequency dataset, the partial auto-correlation of all the hardware metrics excluding frequency shows that only last 2-4 lags are enough to explain the next value. To make a simplified model, we try to predict the next value using just one lag. By doing this, we can assume our data as i.i.d. (independent and identically distributed), as we ignore the dependence on input at t on input at t-1, t-2, and so on. The proposed approach is depicted in Figure 15.

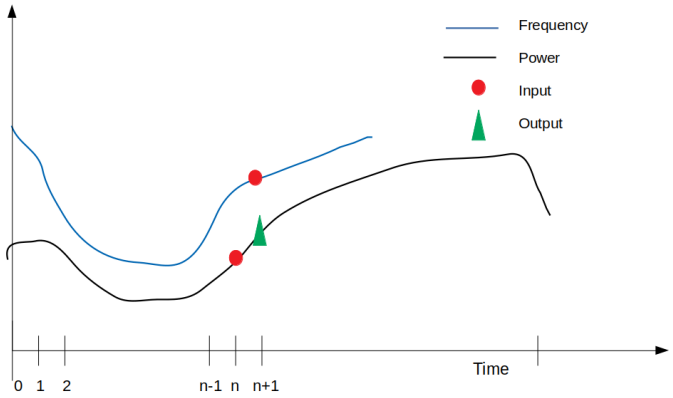


*Figure 15: One lag approach*

In this model, the dram and package power at t = n+1 is predicted using all the hardware metrics(Table 4) at t = n, and frequency at t = n+1. The frequency at next time step is taken as input to get the predicted dram and package power as a function of frequency.

### 6.1.1 Linear Regression on Fixed Frequency dataset

The dataset used is fixed frequency dataset (900 traces). The package and dram power is predicted as a linear function of input vector. The data is normalized using Standard scalar (gaussian distribution).Various linear regression models were used to compare their performance. They are Ridge(L2 regularized), Lasso (L1 regularized), which are the traditional methods used for regression, and Huber[29] and Thiel-sen[30] regression which are robust to outliers. The models are implemented using sklearn in-built library functions. The validation accuracy in terms of R2 score[31] for all the methods is summarized in Table 1. R2 score, also called the coefficient of determination, is the proportion of the variance in the outputs that is predictable from the inputs.

| Model | R2-package power | R2-dram power |
|---|---|---|
| Ridge | 0.98 | 0.99 |
| Lasso | 0.97 | 0.99 |
| Huber | 0.98 | 0.99 |
| Thiel-sen | 0.98 | 0.99 |

*Table 6: R2 validation accuracy - Linear regression(fixed freq dataset)*

The predicted vs actual plot (Figure 16) for package power shows some discontinuity. This is suspected to the turbo region, where the power-frequency behavior is expected to be non-linear. The residual plot (Figure 17) for package power shows a wide non-linear dispersion of error. This means the model doesn't generalize well for our data.

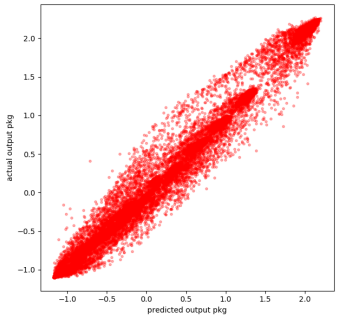


*Figure 16: Package power - Predicted Vs actual, Linear Regression on Fixed freq dataset*

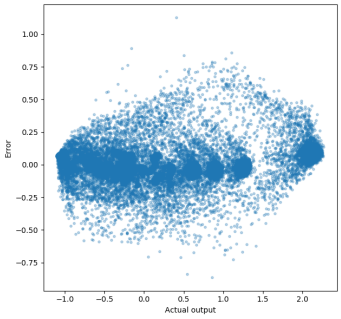


*Figure 17: Package power - Residual plot, Linear Regression on Fixed freq dataset*

The corresponding weights of the input features in linear regression implies the dependence of predicted value on input features. **The weight corresponding to the frequency is around 1.6 per 0.1GHz for package power and -0.18 per 0.1Ghz for dram power**. This behavior is expected for dram power, which is independent of frequency for the HPC application. But the dependence of package power on frequency is rather low to gain any actionable knowledge on optimal frequency setting.

### 6.1.2 Random forest regression on fixed frequency dataset

Among all available regressors, we chose multi-output random forest regressor that is already implemented in Scikit-learn, an open source Python library that implements a range of machine learning, preprocessing, cross-validation and visualization algorithms. The algorithm and its pros and cons are explained in the next paragraphs.

Random forests are an ensemble learning method where a group of weak learners can come together to form a strong learner, Figure 18 They are used for nonlinear multiple regression that operate by constructing a multitude of decision trees at training time where each leaf contains a distribution for the continuous output variable that is a real valued number. We fit a regression model to the target variable, the data is split at random sample (about two third of the total set) split points. We calculate individual trees error between the predicted values and the actual values. The variable resulting in minimum error is selected for the node. Then this process is recursively continued till the entire data is covered.

For regression the predicted value at a node is the average response variable for all observations in the node. The greater the inter-tree correlation, the greater the random forest error rate, so one pressure on the model is to have the trees as uncorrelated as possible.

Random Forests are a popular and powerful method for various machine learning tasks. Some advantages of this algorithm are mentioned here. It can deal with unbalanced data and has an effective method for estimating missing data and maintains accuracy when a large proportion of the data is missing. It runs efficiently on large databases and can handle thousands of input variables without variable deletion. It gives estimates of what variables are important and detect variable interactions. Prototypes are computed that give information about the relation between the variables and the classification. It requires little data preparation such as normalization.

However there are some weaknesses that could be pointed out for this method. Random forests have been observed to overfit for some datasets with noisy classification/regression tasks. It has a weak performance on a small training set. Unlike decision trees, the classifications made by random forests are difficult for humans to interpret and the feature selection process is not explicit and feature fusion is not obvious.

For training the random forest we used 300000 sampled in total and we randomly split them to train and test data with ration of 0.7 and 0.3 respectively. Figure 19 shows the prediction result on test data. Average of relative error is 0.023 and R2 score for the predicted samples is 0.96. After computing the each feature importance in training the random forest we realized that the frequency of the current state does not contribute to the other features from previous lag to predict the power of the current lag.

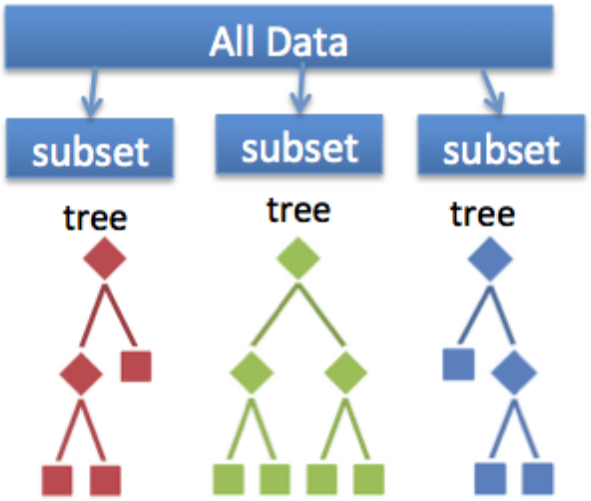


*Figure 18: Random Forest*

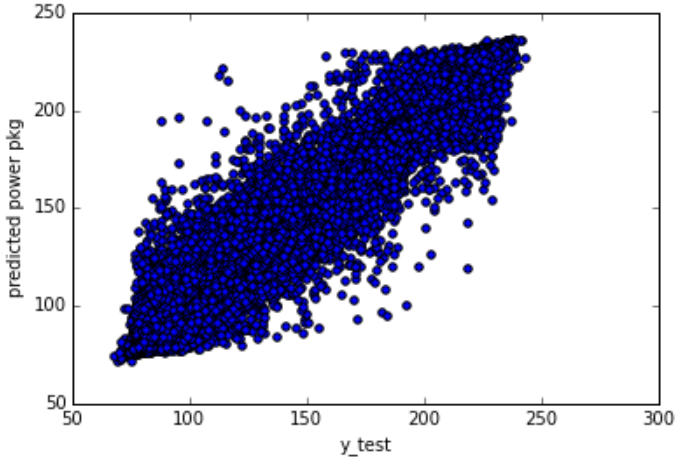


*Figure 19: Package power - predicted VS actual, Random forest on fixed freq dataset*

**Proposed solution**

- It is possible that the model is not be able to capture good dependence of package power on frequency due to constant frequency runs in the dataset. A dataset with varying frequency runs might be able to capture better behavior. Also, a varying frequency data is closer to the real data which will be encountered when running the application with frequency optimization policy.
- The non-linear residual might be improved using a non-linear regression model.

### 6.1.3 Linear Regression on varying freq. dataset

To get better frequency weights for output power, a dataset with varying frequency was used(900 traces). The dataset is generated such that frequency is 80% likely to change in the next time sample. For modelling, the same process is repeated as in above sub-subsection. The validation accuracy(R2 score) for package and dram power is summarized in Table 2. **The weights corresponding to input frequency to predict package power is found to be 2.4 per 0.1GHz.** This in an improvement over fixed frequency dataset but still not enough to get actionable knowledge on optimal frequency decision.

| Model | R2-package power | R2-dram power |
|---|---|---|
| Ridge | 0.98 | 0.99 |
| Lasso | 0.95 | 0.98 |
| Huber | 0.97 | 0.99 |
| Thiel-sen | 0.98 | 0.99 |

*Table 7: R2 validation accuracy - Varying frequency*

### 6.1.4 Nu Support vector Regression on varying freq. dataset

To resolve the poor residual plot of linear regression model, a non-linear model, namely Nu Support vector regression is used. The input features are kernalized using RBF(radial basis function). The RBF is very versatile to make better generalization as it can make arbitrarily small decision boundaries. The model is implemented using sklearn, with a function called sklearn.svm.NuSVR[32].

Hyperparameter tuning: The model was tuned with respect to Nu(fraction of support vectors) and C(penalty parameter of error). The best model was obtained fro Nu = 0.75 and C = 10. The model was trained on 300 trace files (100,000 samples). The validation accuracy(R2) for package power is 0.993, whereas for DRAM power is 0.998. Figures 20 and 21 show the predicted Vs actual plot of package and dram power. The plot for package power shows a spread where the values are under-predicted.

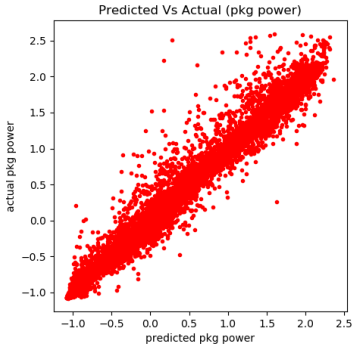


*Figure 20: Package power-Predicted Vs actual, Nu Support vector Regression on Varying freq dataset*

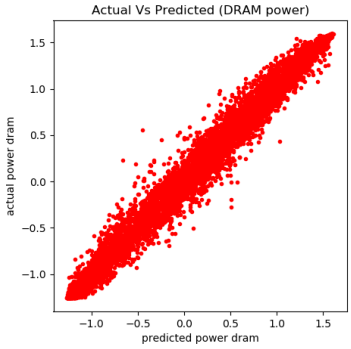


*Figure 21: DRAM power-Predicted Vs actual, Nu Support vector Regression on Varying freq dataset*

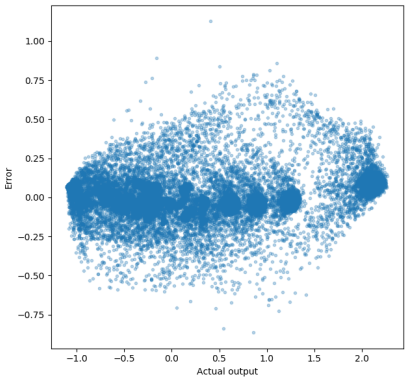


*Figure 22: Residual plot(package)-Linear regression*

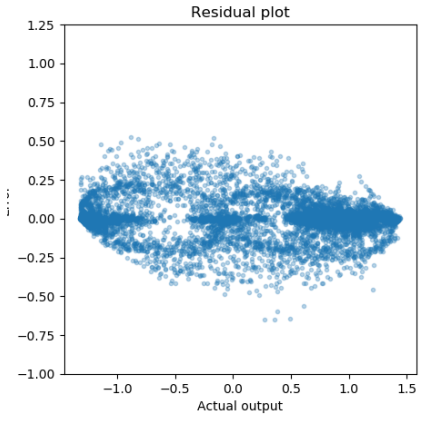


*Figure 23: Residual plot(package)-Nu SVR regression*

The comparison of residual plots between the linear regression and Nu support vector regression can be done

with Figure 22 and 23. The non-linear behavior of residual error is resolved while using Nu support vector regression. However, Nu SVR doesn't solve the problem of frequency-power correlation in our predictions. **The dependence of predicted package power on frequency is low in Nu Support vector Regression**, similar to linear regression on varying frequency dataset.

### 6.1.5 Random Forest Regressor

Beside the linear regression we tried random forest regressor to predict power package and power dram based on data with varying frequencies.

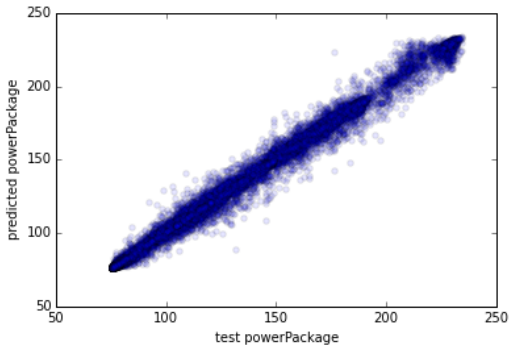


*Figure 24: Package power-Predicted Vs actual, Random forest on varying freq dataset*

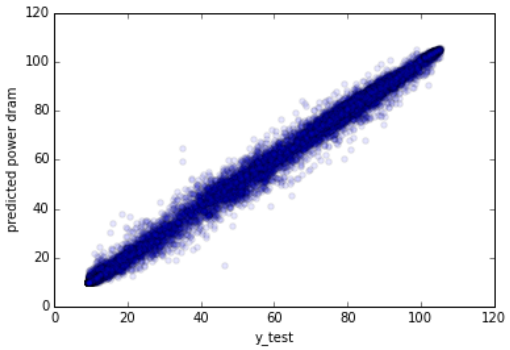


*Figure 25: DRAM power-Predicted Vs actual, Random forest on varying freq dataset*

The model was trained on 52000 samples and tested on 22000 samples. Total relative error for test data is 0.024, which is 0.015 for power package alone and 0.033 for power dram. Total R2 score is 0.993, 0.992 for power package and 0.995 for power dram. Still frequency of the current lag does not contribute to the other features from previous lag for predicting power of the current lag.

Since random forest principle is based on splitting the data through the decision trees, we don't need to normalize the data.

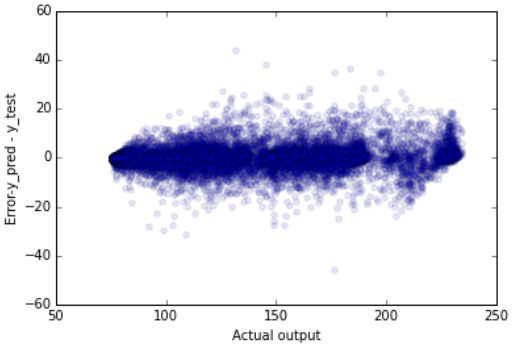


*Figure 26: Residual plot(power package), Random forest regression*

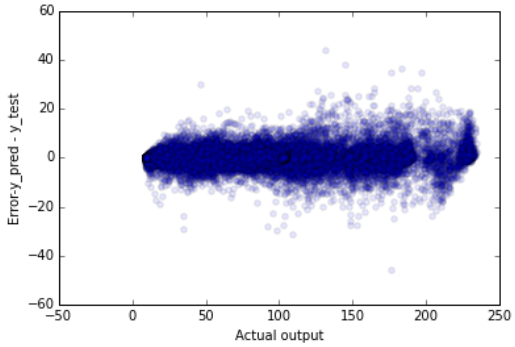


*Figure 27: Residual plot(power dram), Random forest regression*

Fig. 26 and 27 indicate the prediction error versus the actual value for power package and power dram respectively.

### 6.1.6 Region based linear regression

To resolve the issue of bad dependence of frequency on predicted package power, a region based regression approach is tried. The motivation is that the dependence of frequency on package power varies from region to region.

The model is segregated into regions using region-id metric from GEOPM. Ridge regression is used to predict powers for data from a particular region. The correlation between frequency and power, along with the validation accuracy is summarized in Table 8. We see that this correlation is 'stronger' for dgemm region, weak for sleep region, and somehwere in the middle for stream region. This behavior is as expected based on the characteristics of the regions. Dgemm region should vary strongly with frequency as compared to sleep region. We can conclude that regression on each individual region gives 'more realistic' correlation of power with frequency.

| | **pkg power/0.5GHz freq** | **dram power/0.5GHz freq** |
|---|---|---|
| Sleep Region | 2.7 | -0.04 |
| Dgemm Region | 16.2 | 1.25 |
| Stream Region | 9.6 | 1.8 |
| Validation acc. | 80% | 95% |

*Table 8: R2 validation accuracy - region based linear regression*

**Proposed solution**

- Incorporation of multiple lags in our model can characterize the local 'region' behavior.
- For multiple lags the input is a sequence(series) intead of one past value. The data is non- i.i.d. in such a case. RNN/LSTM can be used.
- We can use i.i.d. supervised learning to solve this problem by generating features which characterize the sequence. The features are then i.i.d. and we can use simple supervised learning methods.

## 6.2 Modelling approach 2 - multiple lag prediction

As proposed above, incorporating more than one lag can help in improving our correlation with frequency. The motivation for this approach can be understood from the region based linear regression where the dependence of frequency on output power becomes more 'realistic'. This might mean that the model performs well when it is trained with data belonging to a 'local region'. Hence, we can try to capture the behavior of this local region by using multiple lags in our model. The models which treat the input data as a sequence in time are called temporal models. Such models range from hidden markov chain models, kalman filtering, to more complex Recurrent Neural Networks(RNN) and long-short term memory (LSTM) neural networks. However, such models suffer from difficult training, need for well-conditioned data, sensitivity to noise and biases, and generalization issues.

Instead of making a 'sequence' as input in our model, we can generate features which can characterize this sequence in the sense of preservation of information contained in the series. The main benefit is a simpler supervised learning model can be used. This comes at an expense of generating the features. Since our machine learning model should predict the optimum frequency at runtime, this computational effort of generating the features should be as low as possible. This trade-off and the choice of features is discussed in detail in Section 5.3. The sequence of data points which are used to generate features should be uniformly spaced in time. This is needed to assume each data point in the sequence as a random variable independent of the time interval. Following this, we use linear interpolation on our raw time series to get evenly spaced time series.

To decide on the length of this sequence, two things were considered:

- Longer sequence is better to produce more 'concrete' values of statistical features.
- Feature generation and hence, decision making will be faster for shorter sequence.

Considering these two factors, sequence length of 10 lags is chosen.

### 6.2.1 Predicting Instructions retired

The optimal frequency decision till now was based on the correlation between frequency and power, and the frequency which minimizes the power is assumed to be the optimal. The frequency which gives us the lowest power will always be the lowest frequency. This is however, not a correct policy for optimal frequency. This is due to the power-time tradeoff. Minimizing the energy in an application means minimizing the time integral of total power. Running an application at low power might take significantly longer and turn up consuming more energy than running it at high power as it might take less time. The prediction of run-time of an anonymous application is a difficult task and very application specific, as two applications similar in terms of type of regions can take very different amount of times, depending on the application itself.So, a different notion of 'progress' is needed, which can be used to make the optimal frequency decision. For this purpose, we can use instructions retired, which directly tells us the amount of cpu instructions executed in the application. Higher instruction count means we might be progressing fast and vice-versa.

The Decision policy:
Using the history of hardware metrics in the current application, choose the frequency for next timestep/window which gives high instructions retired and use as minimum power power as possible. For this purpose, we define the efficiency as a function of frequency as:

$$E(f) = \frac{Instructions(f)}{Power(f)} \tag{1}$$

The optimal frequency is then given by:

$$f_{opt} = argmax_f E(f) \tag{2}$$

### 6.2.2 The Frequency-Power lag

All our models till now are unable to get good dependence of package power with frequency. After looking closer to the time series of package power and frequency, it was found that there is a lag in the correlation between them. After setting the frequency in GEOPM, it takes some time for the actual core frequency to be set, which causes this lag. This delay can be seen in Figure 28, where we see a delay of 50-100 ms. Since our sampling rate is 50ms, it means the power is lagging behind by 1-2 samples.

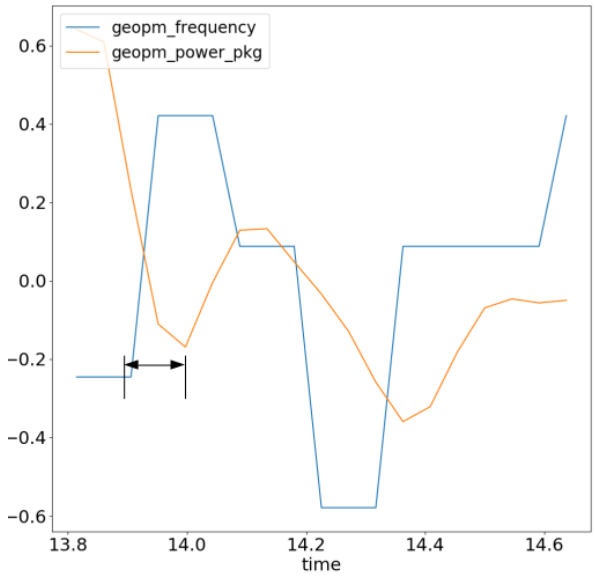


*Figure 28: Frequency power lag*

There is a need for a model which will account for this lag in some way. This is done by generating data with a varying frequency in which the frequency changes after every 3 timestamps. Since, input is a 9 lag sequence, the frequency will vary thrice within the sequence, this should give enough variability in

frequency to have correlated data in the feature space. The idea is that in feature space the lag won't affect the correlations. The outputs of the model are the mean of instructions and power over the 3 timestamps where the frequency is the same. This might compensate for the frequency-power lag in our output vector.

### 6.2.3 Random forest on fixed frequency

Due to the reasons explained in 6.2 we extracted the features from 10 lags and trained the model to predict power package and instruction retired of the next lag. We try to predict the model performance(power package and instruction retired) using generated data with fixed frequency. Figure 29 and Figure 30 indicates the predicted values versus the actual values for power package and instruction retired respectively. The average relative error for both test data is 0.119. Despite from the fact that the model is predicting the output values relatively good but the coefficient of frequency in feature importance gotten from random forest is zero. Our goal is to find 'realistic' correlation of the frequency and the power consumption.

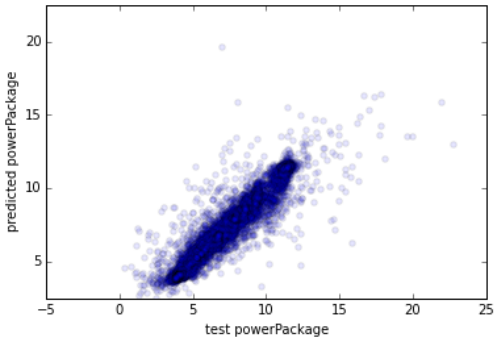


*Figure 29: Package power-Predicted Vs actual, Random forest on statistical data with fixed frequency*

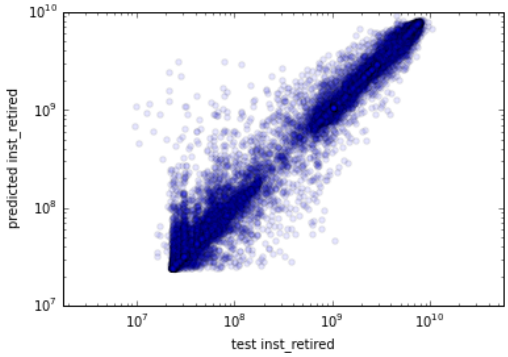


*Figure 30: Instruction-Predicted Vs actual, Random forest on statistical data with fixed frequency*

### 6.2.4 Random forest on varying frequency

As explained earlier having varying frequency in generated dataset is more realistic and we tried to use such data for training random forest regressor. We trained the model with the data with frequency 80% changing in every control loop (refer to Section 4.2.2). By analyzing the results, we realized that the rate of frequency changing is too high which is different from the real scenario and not stable to have reliable modeling. Hence we decreased the rate of frequency change and generated data for frequency changing in every 3 control loop. The input is features generated from 9 lag sequence, and output is the mean instructions and power in the next 3 lags. The results are shown in Figure 31 and Figure 32. Average of the relative error on all predicted outputs is 0.135.

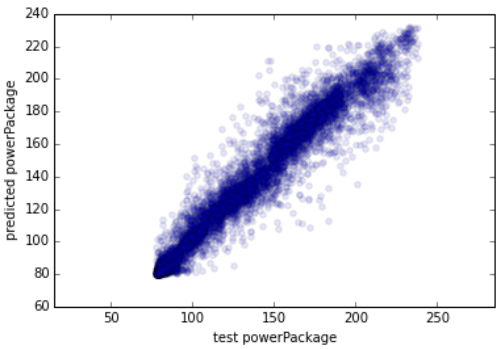


*Figure 31: Package power-Predicted Vs actual, Random forest on statistical features with varying freq*

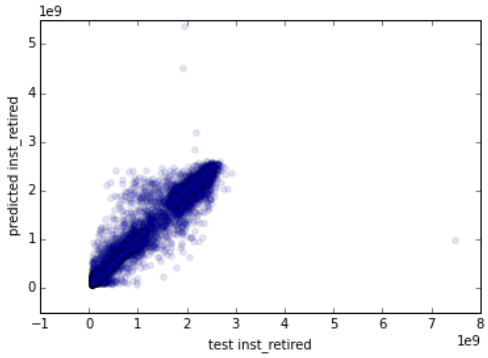


*Figure 32: Instruction-Predicted Vs actual, Random forest on statistical features with varying freq*

Figure 33 indicates the importance of each feature in training the model. As can be seen average of instruction retired for the last 9 lags have the highest impact on the trained model, however the frequency of the current sample for which we want to predict the performance (power package and instruction retired) is among top 10 important features.

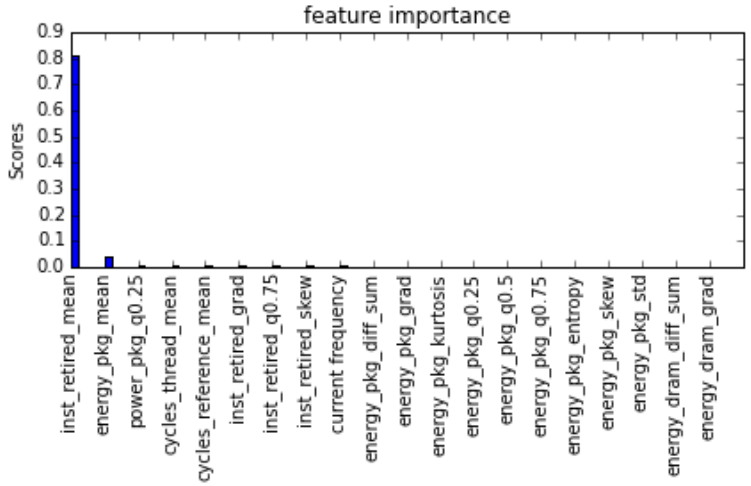


*Figure 33: Feature importance distribution*

After training the model, we evaluated the performance on test datasets. The first test dataset is the features generated from traces of another node, to see the prediction capability across nodes. Fig. 34 and Fig. 35 show the predicted power and instruction on a dataset from another node. The results have R2 score of 0.973 and relative error is 0.132 which means the trained model has high accuracy and might be generalized across different nodes. However, the two nodes are very similar, leading to high accuracy. Data from more nodes is needed to give more conclusive result.

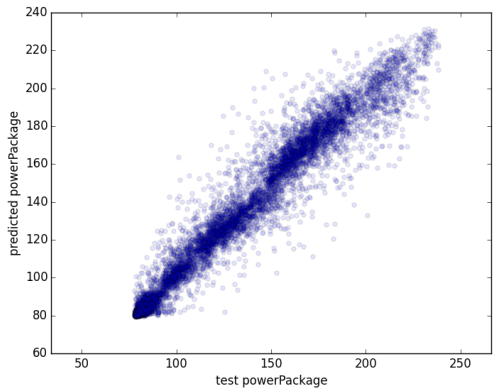


*Figure 34: Package power-Predicted Vs actual on another node*

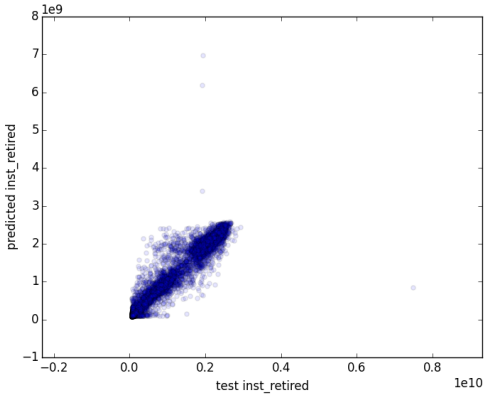


*Figure 35: instruction-Predicted Vs actual on another node*

We also evaluate the trained model on GADGET data. Figure shows that application is characterized by frequent changes in regions and memory bounded-ness, and is much different than our training data. The predicted vs actual values are shown in Fig. 36 and fig. 37. Mean value of the relative error is 0.261. This implies our model generalizes fairly for an actual application.

### 6.2.5 Merging dcdb data to the GEOPM features

So far all the models were trained based on the data from GEOPM, however it was explained in Section 4.3 that adding hardware data from dcdb could be useful to get a more realistic and robust model. Hence we merged dcdb data to GEOPM data and trained random forest on these data together. The corresponding

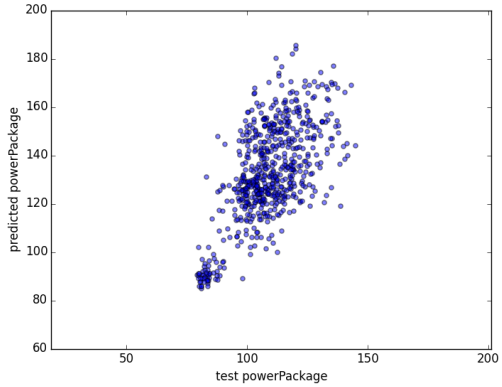


*Figure 36: Package power-Predicted Vs actual on GADGET data*

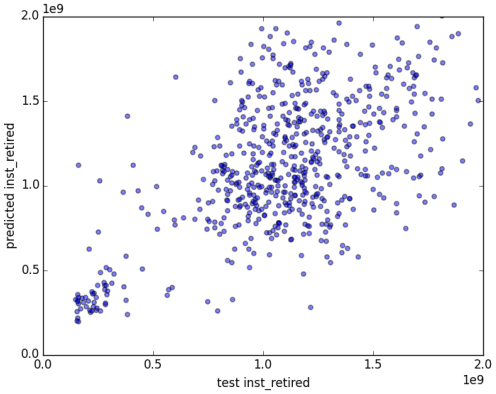


*Figure 37: instruction-Predicted Vs actual on GADGET data*

results for predicting the power package and instruction retired are shown in Figure 38 and Figure 39. About 7000 samples were used for training the model and about 3000 are tested in these figures. The average relative error for all test data is 0.788. As can be seen in Figure 39, there are some very large values in

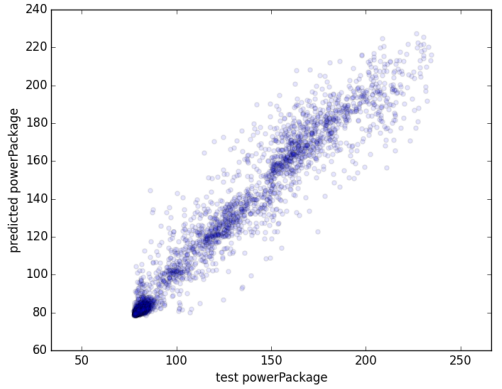


*Figure 38: Package power-Predicted Vs actual, merged data from GEOPM and DCDB*

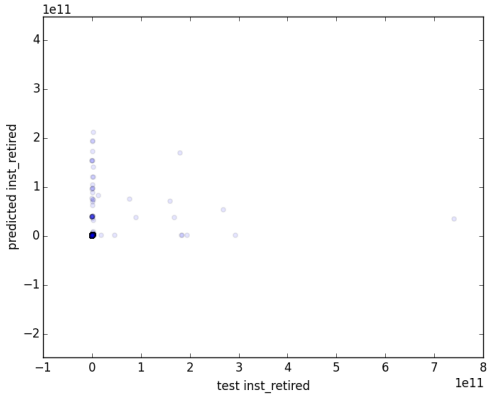


*Figure 39: instruction-Predicted Vs actual, merged data from GEOPM and DCDB*

instruction retired that might be misleading for the visualization since most of the data are concentrated in lower values. After excluding those outliers that appear randomly and sparesly in the data we got much better results as shown in Figure 40 and Figure 41. The relative error on all test data is 0.135.

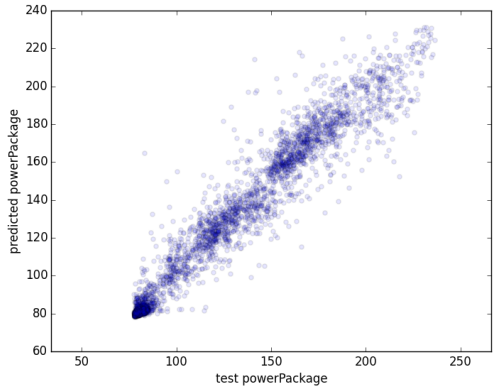


*Figure 40: Package power-Predicted Vs actual, outliers excluded*

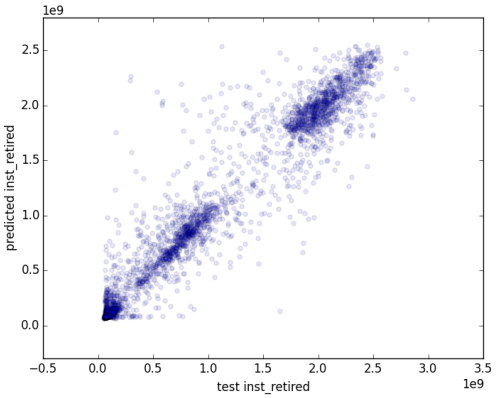


*Figure 41: instruction-Predicted Vs actual, outliers excluded*

# 7 Discussion & Future Work

In this project we tried to optimize the energy consumption in HPC systems with machine learning methods utilizing dynamic frequency scaling. We generated and pre-processed the data from GEOPM and DCDB, and based on them we trained different regression models to predict optimum frequency settings.

Table 9 summarizes the performance of various 1-lag models on data generated with fixed and varying frequencies. All the models are trained on a limited set of metrics (Table 4) and predict only power as output. It can be concluded that with fixed frequency we get better predictions even though an application will not have fixed frequencies in real scenario. We also conclude that random forest performs better than both linear regressor and Nu Support Vector regressor, so for further models only random forest is used. Even though we can predict the powers at next timestamp accurately, the 1-lag model suffers from low correlation between package power and frequency.

| **Dataset** | **Average relative error** |
|---|---|
| Fixed frequency - linear regressor | 0.174 |
| Fixed frequency - random forest | 0.023 |
| Varying frequency - linear regressor | 0.181 |
| Varying frequency - SVM | 0.074 |
| Varying frequency - random forest | 0.024 |

*Table 9: Comparison of model performance on different configurations of the features*

Further models were developed using statistical features (Section 5.3) generated from past 9 lags as input to predict the mean instructions and power over the next 3 lags. Table 10 compares the mean relative error on predicted outputs (power package and instruction retired) for different configurations of statistical features explained in section 5.3 trained with random forest regressor. The relative error is reasonably low for all datasets trained with this multiple lag approach. The feature importance of frequency (Figure 33) is also higher as compared to 1 lag approach.

| **Dataset** | **Average relative error** |
|---|---|
| Fixed frequency (GEO) | 0.119 |
| Varying frequency (GEO) | 0.135 |
| Varying frequency (GEO + dcdb) | 0.788 |
| Varying frequency with excluded large instruction (GEO + dcdb) | 0.135 |

*Table 10: Comparison of model performance on different configurations of the features*

From the analysis results of both modelling approaches, we conclude that random forest can predict power and instructions with high accuracy based on statistical features from the data made available by GEOPM and DCDB. Our trained model shows good performance on the test data from the same node as well as from a different node. We suspect that successful generalization of our machine learning models to the second node might be the result of little variance between the nodes we tested. To have a better understanding in this issue, more nodes should be used as a test case. Our prediction in the cosmological N-body simulation code - GADGET, shows that our model generalizes well for a real application. There is still room for improvement such as trying better configurations of input features which could be achieved by appling PCA on the combined data from GEOPM and DCDB. The model parameters can still be tuned and optimized at a more comprehensive level. Furthermore, our work have concentrated on applications that are run on a single node but considering multiple nodes can be a crucial and interesting research question. In future iterations of this project, implementing our model in a GEOPM agent would tell how much energy saving is achieved compared to the other Agents and the current LRZ solution.

# Glossary

**High Performance Computing Systems** compute systems that has high capability where the resources are distributed over multiple machines. 2

**High Performance Computing** practice of utilizing parallel programming to run advanced applications on the multicore systems efficiently. 2

**Threads** a sequence of instructions that can be executed independently from each other. 2

**Scheduler** module in HPC systems that handles application queuing and resource distribution. 2

**Runtime system** container for HPC application that resides between operation system and application which optimizes the application runtime with control algorithms. 2

**DVFS** a power managment strategy where the Voltage / Frequency settings are modified the optimize the total energy cost. 4

**Hardware counters** special purpose registers built into modern microprocessors that keeps track of hardware related activities. 6

**Region** a small fragment of the application that exhibits same behaviour throughout. 6

**Perf Events** a observation tool that contains measuresments from events coming from different resources including software and hardware. 10

**Sysfs** a pseudo file system that contains information about kernel subsystems, hardware devices, and associated device drivers. 10

**Procfs** a special filesystem that presents information about running processes. 10

# References


[1] Valve Corporation. Steam hardware software survey: January 2019, 2019-02-06. https://store.steampowered.com/hwsurvey/Steam-Hardware-Software-Survey-Welcome-to-Steam.

[2] Top 500 list, 2019-02-08. https://www.top500.org/lists/2018/11/.

[3] Top 500 list, tianhe-2a, 2019-02-06. https://www.top500.org/system/177999.

[4] Muhammad Shafique, Siddharth Garg, Jörg Henkel, and Diana Marculescu. The eda challenges in the dark silicon era: Temperature, reliability, and variability perspectives. In *Proceedings of the 51st Annual Design Automation Conference*, DAC '14, pages 185:1–185:6, New York, NY, USA, 2014. ACM.

[5] Jörg Henkel, Heba Khdr, Santiago Pagani, and Muhammad Shafique. New trends in dark silicon. In *Proceedings of the 52Nd Annual Design Automation Conference*, DAC '15, pages 119:1–119:6, New York, NY, USA, 2015. ACM.

[6] Z. Wang, Z. Tian, J. Xu, R. K. V. Maeda, H. Li, P. Yang, Z. Wang, L. H. K. Duong, Z. Wang, and X. Chen. Modular reinforcement learning for self-adaptive energy efficiency optimization in multicore system. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 684–689, Jan 2017.

[7] Axel Auweter, Arndt Bode, Matthias Brehm, Luigi Brochard, Nicolay Hammer, Herbert Huber, Raj Panda, Francois Thomas, and Torsten Wilde. A case study of energy aware scheduling on supermuc. In *Proceedings of the 29th International Conference on Supercomputing - Volume 8488*, ISC 2014, pages 394–409, New York, NY, USA, 2014. Springer-Verlag New York, Inc.

[8] W. Lee, Y. Wang, and M. Pedram. Optimizing a reconfigurable power distribution network in a multicore platform. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(7):1110–1123, July 2015.

[9] X. Lin, Y. Wang, and M. Pedram. A reinforcement learning-based power management framework for green computing data centers. In *2016 IEEE International Conference on Cloud Engineering (IC2E)*, pages 135–138, April 2016.

[10] M. Triki, Y. Wang, A.C. Ammari, and M. Pedram. Hierarchical power management of a system with autonomously power-managed components using reinforcement learning. *Integr. VLSI J.*, 48(C):10–20, January 2015.

[11] Yanzhi Wang, Qing Xie, Ahmed Ammari, and Massoud Pedram. Deriving a near-optimal power management policy using model-free reinforcement learning and bayesian classification. In *Proceedings - Design Automation Conference*, pages 41–46, 2011.

[12] Hao Shen, Jun Lu, and Qinru Qiu. Learning based dvfs for simultaneous temperature, performance and energy management. In *Proceedings - International Symposium on Quality Electronic Design, ISQED*, pages 747–754, 2012.

[13] Ying Tan, Wei Liu, and Qinru Qiu. Adaptive power management using reinforcement learning. In *IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers, ICCAD*, pages 461–467, 2009.

[14] Mwaffaq Otoom, Pedro Trancoso, Hisham Almasaeid, and Mohammad Alzubaidi. Scalable and dynamic global power management for multicore chips. In *Proceedings of the 6th Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures*, PARMA-DITAM '15, pages 25–30, New York, NY, USA, 2015. ACM.

[15] Hwisung Jung and Massoud Pedram. Supervised learning based power management for multicore processors. *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, 29(9):1395–1408, September 2010.

[16] Sheng Yang, R. A. Shafik, G. V. Merrett, E. Stott, J. M. Levine, J. Davis, and B. M. Al-Hashimi. Adaptive energy minimization of embedded heterogeneous systems using regression-based learning. In *2015 25th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, pages 103–110, Sep. 2015.

[17] M. G. Moghaddam, W. Guan, and C. Ababei. Investigation of lstm based prediction for dynamic energy management in chip multiprocessors. In *2017 Eighth International Green and Sustainable Computing Conference (IGSC)*, pages 1–8, Oct 2017.

[18] A. S. Kumar and S. Mazumdar. Forecasting hpc workload using arma models and ssa. In *2016 International Conference on Information Technology (ICIT)*, pages 294–297, Dec 2016.

[19] Slack official website. https://slack.com/.

[20] Coolmuc3 official website. https://www.lrz.de/services/compute/linux-cluster/coolmuc3/.

[21] Jonathan Eastep, Steve Sylvester, Christopher Cantalupo, Brad Geltz, Federico Ardanaz, Asma Al-Rawi, Kelly Livingston, Fuat Keceli, Matthias Maiterth, and Siddhartha Jana. Global extensible open power manager: A vehicle for hpc community collaboration on co-designed energy management solutions. In Julian M. Kunkel, Rio Yokota, Pavan Balaji, and David Keyes, editors, *High Performance Computing*, pages 394–412, Cham, 2017. Springer International Publishing.

[22] Geopm project website. https://geopm.github.io/.

[23] Nokbone application code. https://github.com/Nek5000/Nekbone.

[24] Amg benchmark code. https://github.com/LLNL/AMG.

[25] Mkl library website. https://software.intel.com/en-us/mkl.

[26] Gadget software website. https://wwwmpa.mpa-garching.mpg.de/gadget/.

[27] Linux kernel profiling with perf. https://perf.wiki.kernel.org/index.php/Tutorial.

[28] Keeping track of processes. https://www.cs.auckland.ac.nz/ alan/courses/os/-book/4.How.14.keepingtrack.pdf.

[29] Peter J. Huber. Robust estimation of a location parameter. *Ann. Math. Statist.*, 35(1):73–101, 03 1964.

[30] H. Theil. H. a rank-invariant method of linear and polynomial regression analysis. *Indagationes Math.*, 35:85 – 91, 1950.

[31] https://en.wikipedia.org/wiki/Coefficient_of_determination.

[32] https://scikit-learn.org/stable/modules/generated/sklearn.svm.NuSVR.htmlsklearn.svm.NuSVR.