MACHINE LEARNING BASED DVFS FOR ENERGY EFFICIENT EXECUTION OF MULTITHREADED WORKLOADS

by

Richard Hay, B.S.

A thesis submitted to the Graduate Council of
Texas State University in partial fulfillment
of the requirements for the degree of
Master of Science
with a Major in Computer Science
December 2014

Committee Members:

Apan Qasem, Chair

Dan Tamir

Hassan Salamy

FAIR USE AND AUTHOR'S PERMISSION STATEMENT

Fair Use

Duplication Permission

ACKNOWLEDGEMENTS

This thesis would not have been possible without my supervisor, Apan Qasem, whose advice, guidance and support from idea to a full fledged product. I would also like to thank my committee members Dan Tamir and Hassan Salamy. A special thanks to friends who gave me specific advise on how to get through this. I'd also like to make a very special thanks to Team Awesome a.k.a Team Out To Lunch.

TABLE OF CONTENTS

Page

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

ABSTRACT

Concerns over high power consumption of large computations and data centers have been growing in recent years. Many software and hardware strategies for reducing power have been proposed as remedies. Dynamic voltage and frequency scaling (DVFS) is one technique that can be effective if given expert knowledge. However, DVFS effectiveness is sensitive to workload characteristics and architectural parameters. Lack of knowledge can hurt DVFS strategies and render it ineffective. This thesis presents a supervised machine learning (ML) strategy for automatically making smart DVFS decisions to improve energy efficiency of multi threaded and multiprogram workloads. The technique uses hardware performance counters to construct feature vectors that capture program behavior and thread interaction in a meaningful way. The resulting models have high accuracy in picking optimal frequencies. Experimental results on contemporary benchmark suite show that application of a ML technique is able to reduce energy consumption by as much as 24% on memory-intensive workloads.

CHAPTER I

INTRODUCTION

Energy efficiency in chip design is important now due to the rising thermal footprint of the CPU. As clock speed increased so did power consumption due to leakage from heat. Processors logistically became much harder to cool. This allowed for the necessary shift from single core systems to heterogeneous multi-core systems. In todays world, supercomputers and data centers incur large cooling costs. There is a tangible impact to saving energy whether it is monetary or environmental and people have become more energy conscious as of recent. There are tools available currently that give the opportunity to save energy. Exploiting tools such as dynamic voltage and frequency scaling (DVFS) can help reduce power consumption.

Because of the importance of energy efficiency in high-performance computing, many techniques have been proposed for managing power and reducing energy consumption. Core gating is a technique that has uses for certain situations. In this technique a core is disabled at times when it has nothing to work on. This situation is called starvation. By disabling a specific core the processor is effectively using less power than before. Thread mapping also happens to be a promising technique for power efficiency and reducing the execution time of a program. Thread mapping involves making choices about where the threads of an application should be run on the processor. The distinct advantage this provides is the ability to reduce execution time and power just on mapping threads of an application to cores on a processor. Since this project deals with workloads which are sets of multithreaded

applications it is suitable to apply thread mapping techniques. It will allow for exploration of the effects affinity configurations have with power and execution time. Affinity is the core to thread assignment of an application. With the ability to map threads, it is useful to use thread migration since it is the most dynamic option once threads are mapped. A thread migration strategy takes threads and moves them to different cores with the strategy of balancing load. If execution is important then thread migration can be beneficial for moving applications to idle cores thereby eliminating starvation. There are a few penalties to this transfer but those penalties can be mitigated with the right decisions.

DVFS is a technique that dynamically changes the voltage and frequency of a processor. Although DVFS is useful, it is difficult to come up with an effective strategy because there is a lot of information needed to make a good decision. DVFS provides opportunity to exploit the frequency of the processor with some overhead associated with the switch from one frequency to another. Modern processors divulge the time it takes to change steps in frequencies and that change is measured in microseconds. This is a physical factor that hinders changes in frequency. It is called transition delay. There are various power saving states that are controlled by firmware. The governor is a mechanism that is a part of DVFS that dictates the range of available frequencies the processor can change to. It also establishes how aggressive the scaling between frequencies can be. This delay from signal to action means that the information collected and decided upon needs to be resolved in that time frame. There is also the software issue of receiving the state

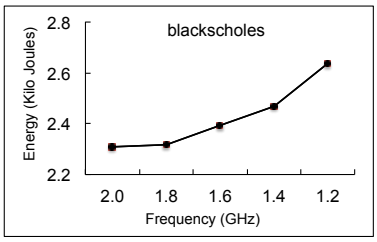
change signal within the software and not just the hardware change. But in terms of benefit, if there is a prediction mechanism then the change can happen right before it is needed negating the previously mentioned issues. In order to make intelligent decisions about DVFS there needs to be some feature set that can be collected to create a prediction model.

This thesis aims to develop a learning based prediction model for DVFS that leverages hardware performance counters to create the feature set. This model will be applied to multiprogram workloads with the intention of gaining some power efficiency. In newer generations of processors manufacturers have provided more counters as well as better descriptions of what the counters record. By polling certain counters it is possible to generate estimates on power. Some of those counters that can be used include cache misses, branch predictions, and clock cycles. It is possible to predict how much power the processor will use in some arbitrary time slice so long at the application has been characterized. Performance counters provide the ability to characterize applications. Power as well as other performance counters provide key information in assessing decisions made by our power aware DVFS system.

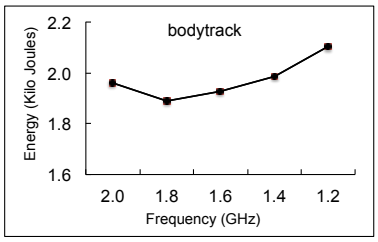
This thesis contributes a few tools and systems for collecting information as well as processing that information. `MLDataGen` is a system developed for the collection of performance counter information. The information `MLDataGen` collected was used to generate the models for power estimation. The collected values were vetted based on importance and the limitations that come from multiplexing live

performance counter values. The values that were selected were used to generate a regression model. We applied this model to a system without a power performance counter. What was learned from this was eventually applied to `perfCollection` which is the second generation of the `MLDataGen` tool. This was used to collect information that would apply to our machine learning models for `smartDVFS`. There are three models being used and they are Baysian, Decision Trees, and Support Vector Machines.

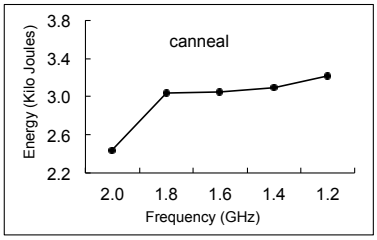
The developed models show that, if used intelligently, DVFS can result in significant energy savings on contemporary parallel workloads. Furthermore, the models provide key insight into program behavior in the presence of frequency scaling. This knowledge can be used to develop more sophisticated techniques for power-aware optimizations. Figure 1.1 shows that there are certain frequencies where power consumption values and execution time trade offs are beneficial. A program where the benefit is seen clearly is steamcluster.



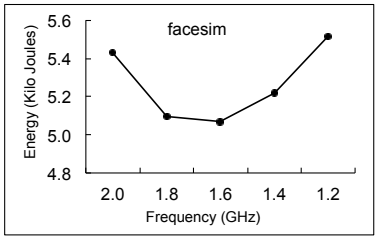
(a) Blackscholes



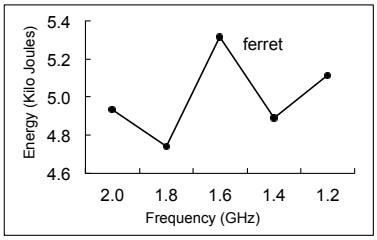
(b) Bodytrack



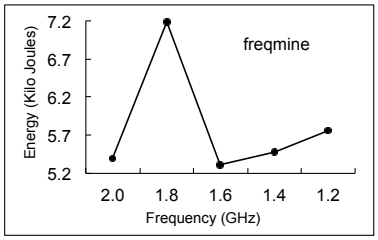
(c) Canneal



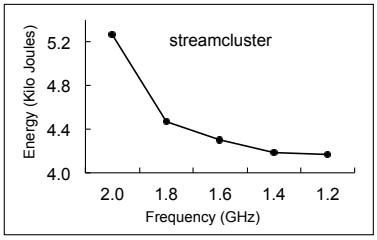
(d) Facesim



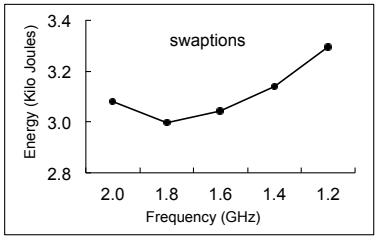
(e) Ferret



(f) Freqmine



(g) Streamcluster



(h) Swaptions

Figure 1.1: Variations in energy consumption in PARSEC benchmarks as a function of core frequency

CHAPTER II

RELATED WORK

2.1 Performance Counters in Power-aware Optimizations

Singh et al. work uses performance counter information to estimate power consumption. They developed a regression model based on readings from a set of hardware performance counters. It also covers the difficulties of reading performance counters and issues with multiplexing. It introduces the concept of power being a piecewise function in that low power consumption fits linearly and high power consumption fits exponentially [Singh et al., 2009].

Power estimation has been explored for quite some time and there are many concerns with how the numbers are generated and how accurate they are. But, there have been some strides to get numbers that correspond with actual values. Chip manufactures have exposed the actual power consumptions of their chips and on top of that each individual cores consumption. Older studies used programs that simulated CPU architecture such as McPAT and newer studies took wattage measurements directly from the wall socket using something similar to a Kill-A-Watt [Li et al., 2009]. Moving forward it is more likely usage of performance counters will trump older methods with its accuracy.

2.2 Dynamic Frequency and Voltage Scaling

There is research being done on core gating by IBM. Vega et al. explored the potential of core gating to show real power savings on an IBM Power 7 chip. The work explains the benefits and importance of thread mapping on a processor. By using the PARSEC benchmark suite, Vega was able to show that programs with starving cores can and should be turned off as a result saving some power that otherwise would have been wasted idling. This type of research is important in the field of web servers where there is a scenario in which a server processor is not receiving page requests. In that scenario and many others it would be prudent to shut off parts of the processor until demand picks up again [Bienia et al., 2008].

Power management is handled differently on different operating systems and hardware. The context in which software and hardware dictate power management commands is blurry. Different processor architectures handle power states with a combination of software and or hardware. Shen et al. sheds light about software based solutions for power management. There is also coverage on DVFS but a significant part of research is focused on evaluating temperatures and software based solutions for power management. Shen provides information describing performance and temperature correlation. It is known that as the demand for performance increases the trade off is power and temperature. The work provides discussion on optimal performance counters to use and determines that instructions per second (IPS) can be used to estimate power consumption with reasonable accuracy [Shen

et al., 2012]. This idea was supplanted in this thesis by the usage of unhalted clock cycles which in essence are the cycles in which the processor is actively executing instructions. This information has value in that it shows the total amount of cycles the processor was actually busy. That counter in particular provides as much detail if not more than IPS [Jung and Pedram, 2010].

CHAPTER III

FRAMEWORK

3.1 Overview

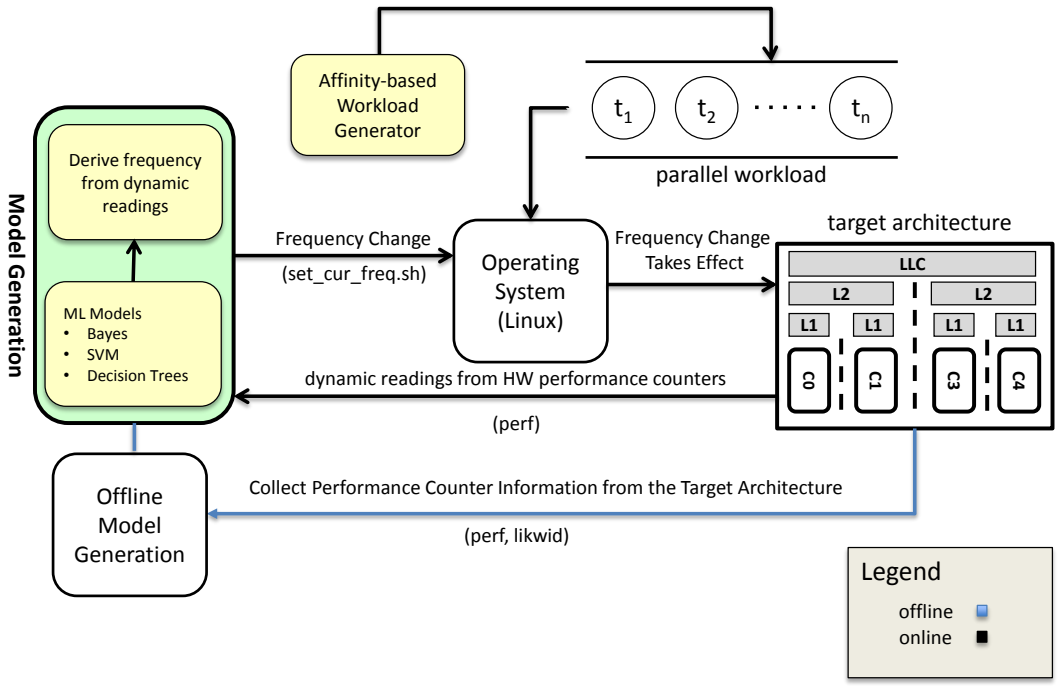


Figure 3.1: Framework overview

The framework for our system has many moving parts some of which are automated and some of which are manual and offline processes. But, each step is just as important as any other and all data collected has its importance in the framework.

Figure 3.1 is a broad overview of what the framework does and its various components are explained in detail.

3.2 Workload Generation

`wkldgen` is a tool created to reduce time to launch multiple applications as well as provide custom configuration mechanics. A workload is a collection of one or more programs. The tool itself allows flexibility in the following areas, types of applications, number of applications, number of threads, input dataset, and affinity configurations.

`wkldgen` is used to generate flexible application launching for use in performance counter collection. Rather than launch each application manually with a simple script it would save time if the launch of applications were generalized and given a few options that would be useful. Affinity configurations provide insight into how a programs execution time and power can be affected by arranging where on the processor threads are run. There was also an interest in the effects of adding additional threads to the power and execution time. Since the processor used for the majority of experiments has 12 logical cores and 6 physical cores the interaction between thread count and power can be explored. Logically, increasing software thread count saturates the cores and increases power consumption. This would be an intuitive understanding except when the program has significant memory accesses. Programs with heavy computation exhibit this intuitive behavior while memory bounded applications do not. This distinction gives some room to find out

optimizations that would be useful for memory bounded applications. In other words, being able to find frequencies that reduce power consumption but still maintain a reasonable execution time. This trade off is key to building a heuristic to address the problem of balancing power savings and execution time.

Table 3.2 shows five affinity configurations each workload was run under. Each affinity mapping is unique with the exception of affinity 0. Affinity 0 is the operating system's choice on mapping threads. Affinity 1 is placing all the threads on half the processor. Affinity 2 takes an application's threads and either places them on the left or right side of the processor. As an example, there is a workload with four programs and they are each running four threads. The first two applications are placed on the first half of available cores on the processor while the last two applications are placed on the last half. So, you have 8 threads sharing 6 logical cores. Affinity 3 distributes threads in such a way that each program has overlapping threads on a core. This creates competing interests from two programs on one core. Using the same example of 4 programs and 4 threads the first program would take up cores 0-3 and the second program would map to 3-7 essentially giving it 5 logical cores with one core in competition. Affinity 4 applies threads in an interleaving fashion. Given the example of 4 programs and 4 threads with 12 available logical cores the first program's threads will be on cores 0,3,7, and 11.

Performance counter selection was based on the work by Singh et. al. in which they ranked the most correlated counters to power. Table 3.2 is a list of adapted performance counters from their work. The counters selected have slight variations

Table 3.1: Affinity configurations

Affinity Name	Configuration Example
Affinity 0 (Operating System)	$A_{1\text{-}2}C_{0\text{-}5}$
Affinity 1 (Cohort)	$A_1C_{0\text{-}2}$ $A_2C_{0\text{-}2}$
Affinity 2 (Half 'n Half)	$A_1C_{0\text{-}2}$ $A_2C_{3\text{-}5}$
Affinity 3 (Resource Competing)	$A_1C_{0\text{-}2}$ $A_2C_{2\text{-}5}$
Affinity 4 (Interleaved)	$A_1C_{0,2,4}$ $A_2C_{1,3,5}$

or approximates in some cases to the work of Singh due to differences in architecture between AMD and Intel hardware. The performance counter information collected this way was used in deriving the power estimation model `Watt's Up` seen later.

In the figures 3.2–3.5 there is a distinction between two workloads that were characterized by calling them compute and cache. Characterize in this context is referring to the recording of a program or workload throughout its entire execution. These workloads include programs from the PARSEC benchmark that stress the computation portion of the processor while cache workloads have large amounts of memory accesses. Hence, cache bounded programs tend to exhibit high cache miss rates and will prove to be useful in maximizing power efficiency. The `Aff` labeling in the figures refers to the configurations provided in Table 3.2. Affinity configurations are an important aspect of power and performance. The operating system needs to monitor and decide what is the most appropriate core configuration to run multi-threaded applications. By applying a few changes in thread mapping it is possible to beat the operating system's optimal mapping. One thing the operating

Table 3.2: Performance counters vetted from Likwid and Perf

Performance Counter	Description
Power [W]	The average power consumption in watts for a time slice.
Time [Seconds]	The time from start of execution to end.
UNHALTED_CLK_CYCLES	Number of clock cycles that executed an instruction.
FP_COMP_OPS_EXE_X87	The amount of floating point instructions executed.
BR_INST_RETIRED_ALL_BRANCHES	Number of branching paths instructions executed.
BR_INST_RETIRED_CONDITIONAL	Number of branches executed conditionally.
BR_MISP_RETIRED_ALL_BRANCHES	The number of mispredicted branch predictions.
INSTR_RETIRED_ANY	Number of instructions executed of any type.
UOPS_RETIRED_ALL	Micro-Ops code executed.
RESOURCE_STALLS_ANY	Issues involving loading resources for instructions.
INT_MISC_STALL_CYCLES	Stalls from something other than Load/Store operations.
L3_LAT_CACHE_MISS	The number of L3 cache misses.
ICACHE_MISSES	The number of instruction cache misses.
L2_RQSTS_MISS	L2 cache requests that resulted in a miss.

system does not seem to do effectively is map threads. Evidence of this can be seen in Figure 3.2(b).

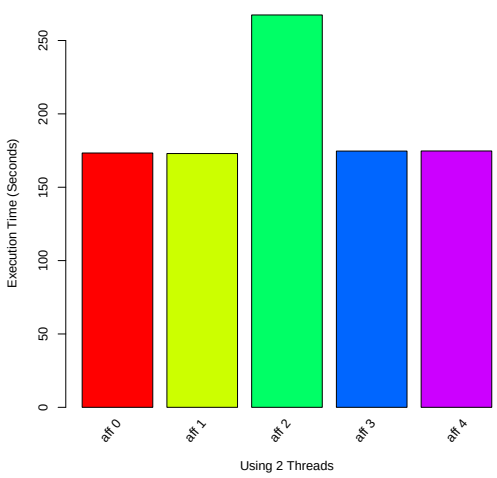
Figure 3.3 shows the trend in power as thread count increases under different affinity configurations for a compute-bound workload. There are significant differences in power consumption between thread counts and affinity configurations. The most notable aspect of these graphs is the power consumption in comparison to execution time. In Figure 3.3(a) the power difference between Aff0, the operating system, and Aff1 is significant. They both have roughly the same execution time but Aff1 has considerably less power usage. These advantages fall apart when the

thread count is 8 or higher. The memory bound workload exhibits similar power patterns to the compute bound workload refer to Figure 3.3 and Figure 3.5. The main differences is the execution time penalty is more significant and the power savings are less drastic. This information leads to believe that optimizations should come from compute bounded workloads as the values appear favorable towards power efficiency.

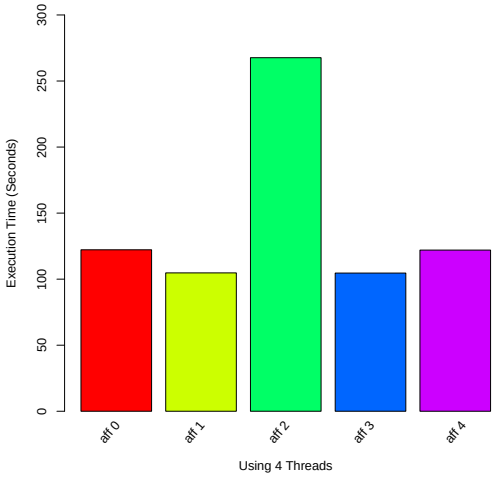
3.3 Fine Grain Probing of Performance Counters

Performance counters are a new feature that the public has been given access to. The counters are registers on the CPU that provide useful information about the CPU in time slices. With the ability to access these counters you gain the power to observe and profile the state of the system. There are a significant amount of tools that allow polling of performance counters.

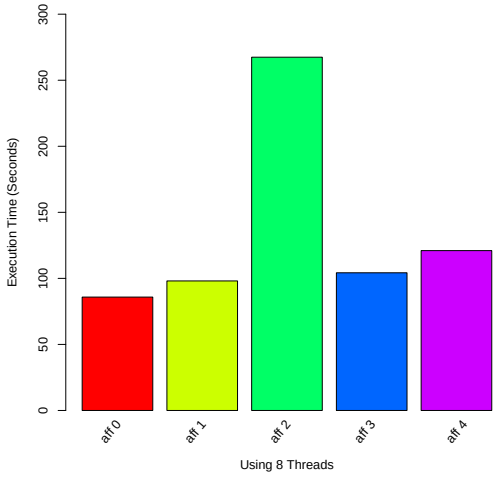
There are some performance counter polling tools that are used for this thesis which include Likwid and Perf. Likwid is a performance counter monitoring solution that is used to collect data [Treibig et al., 2010]. Likwid provides access to power performance counters on Intel's Xeon processor while Perf has better compatibility with older processors. Perf also is useful in conjunction with liwkid because it has less overhead when collecting performance counter information. This makes Perf attractive for real-time monitoring. Both programs expose the issue of multiplexing. Multiplexing in this context means the limited available counters that the machine can track and record information for. The processor can only



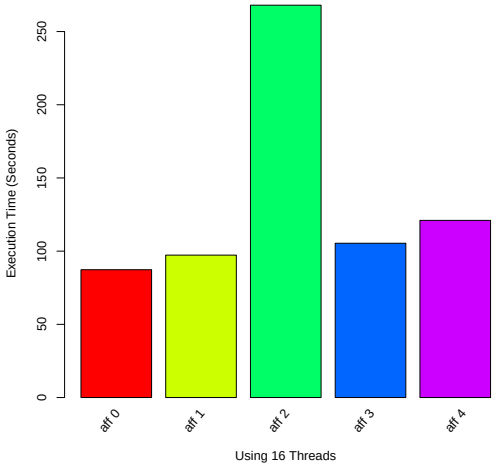
(a) 2-threaded workload



(b) 4-threaded workload



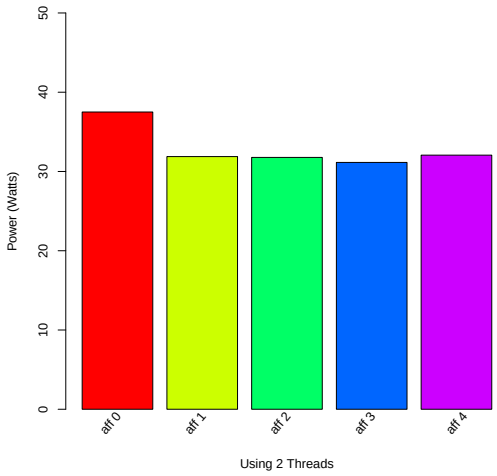
(c) 8-threaded workload



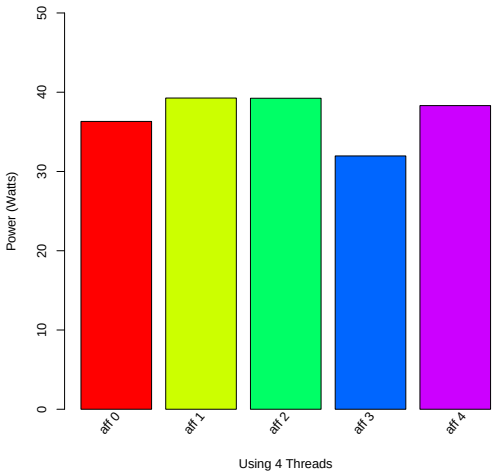
(d) 16-threaded workload

Figure 3.2: Execution time for compute-bound workload for different affinity configurations

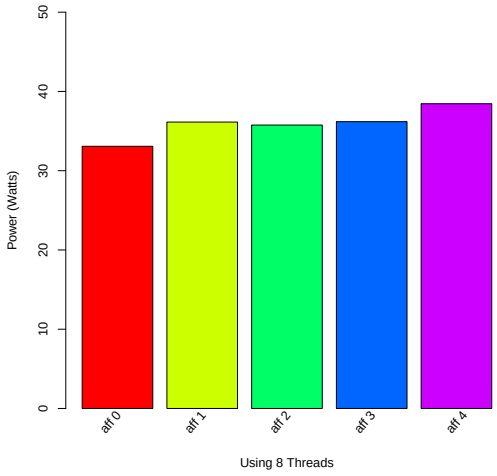
accommodate a small number of counters due to space restrictions on the die. This ends up limiting the amount of counters that can be recorded. Likwid exposes a fixed number of counters on an Intel Xeon processor. The Intel Xeon that is in use



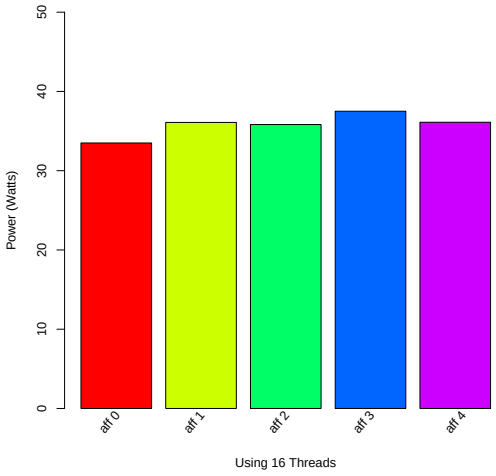
(a) 2-threaded workload



(b) 4-threaded workload



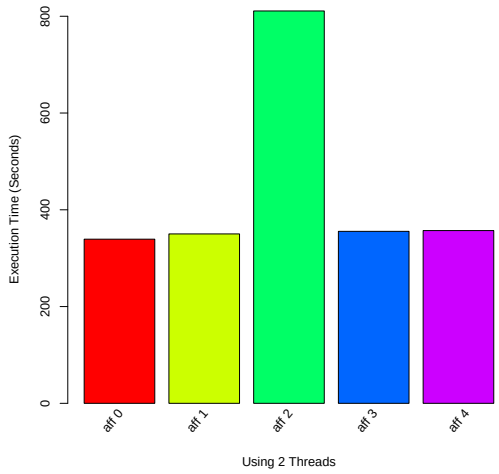
(c) 8-threaded workload



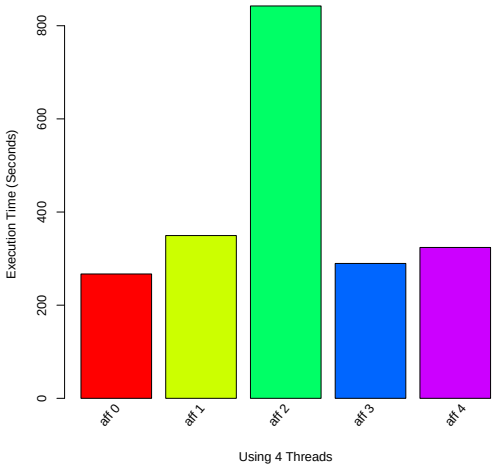
(d) 16-threaded workload

Figure 3.3: Power consumption for compute-bound workload for different affinity configurations

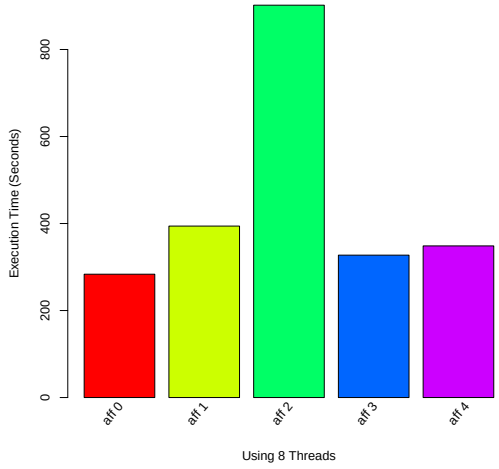
for this thesis only has space for a few counters to be recorded which leads to a scenario where information limitation forces specific choices that need to be made. For instance, MBOX holds instructions related to performance metrics while MEM



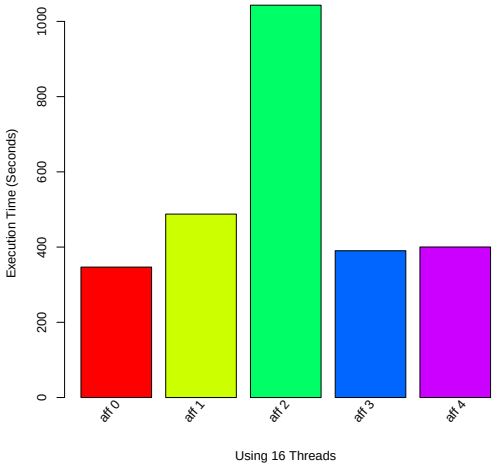
(a) 2-threaded workload



(b) 4-threaded workload



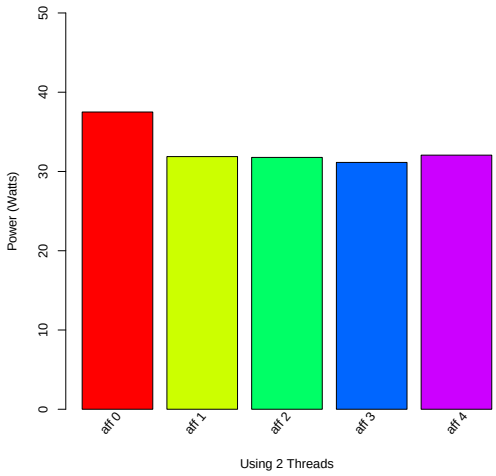
(c) 8-threaded workload



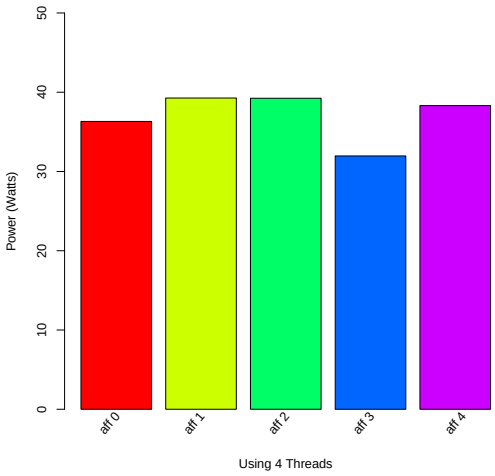
(d) 16-threaded workload

Figure 3.4: Execution time for memory-bound workload for different affinity configurations

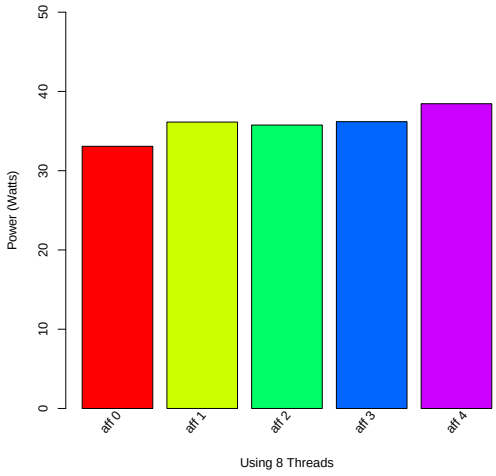
holds cache related performance counters. As part of the system requires data generation from selected features it is necessary to provide a tool that records performance counter information under these limited circumstances.



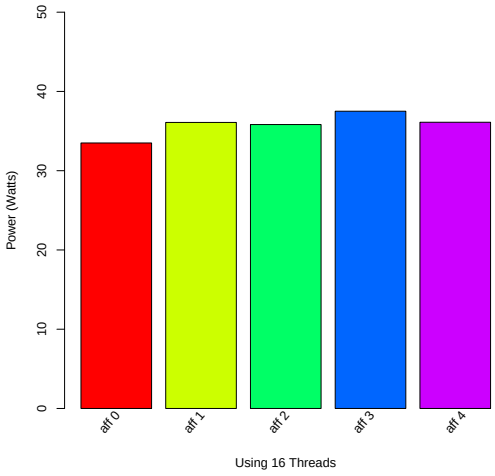
(a) 2-threaded workload



(b) 4-threaded workload



(c) 8-threaded workload



(d) 16-threaded workload

Figure 3.5: Power consumption for memory-bound workload for different affinity configurations

MLDataGen, the data generation tool, is given a set of user defined performance counters to look at and outputs text files with the recorded data for a user defined time slice. This data would then be formatted and used to generate a

model of a specific application or a workload.

3.4 Power Estimation

Watt's Power is a program that implements power estimation for a program. Once it profiles the programs or workload it will calculate a wattage based on a regression equation. This was built in part to validate a power model that would work on older systems that did not include performance counters for power consumption. While it proved valuable in trying to understand current power estimation and forecasting models it was not helpful in providing a backwards compatible way to apply our `smartDVFS` heuristic. The formula used to create the power estimation model is Power [W] = CPU_CLK_UNHALTED_CORE + FP_COMP_OPS_EXE_X87 + L3_LAT_CACHE_MISS + BR_MISP_RETIRED_ALL_BRANCHES + UOPS_RETIRED_ALL. Refer to table 3.2 for descriptions of each variable in the equation. The training of the linear regression model used data that was normalized based on unhalted core clock cycles. When fed into Watt's Up , the unhalted core clock value is unchanged but the rest of the values in the formula have been scaled appropriately. Table 3.4 shows the accuracy of predicted power values. This information validates what was said in Singh et. al. work. Their research concluded that power requires two models rather than one model. Low power value predictions behave differently from high power value predictions. While the estimation can be inaccurate it at least shows the trends in power consumption. This model could be improved if split up between low and high power models.

Table 3.3: Coefficients for Watt's Up linear regression

	Coefficient Value
(Intercept)	16.126921936879
CPU_CLK_UNHALTED_CORE	0.000000000658
FP_COMP_OPS_EXE_X87	-0.049011893798
L3_LAT_CACHE_MISS	77.347080712590
BR_MISP_RETIRED_ALL_BRANCHES	0.630887016032
UOPS_RETIRED_ALL	-0.015837407026

Table 3.4: Real versus estimated power for Watt's Up

Real Power Value	Estimated Power Value	% Error
49.00	51.80	5.71%
43.00	39.83	7.37%
37.50	31.60	15.73%
29.00	25.70	11.38%
12.00	3.40	71.67%

CHAPTER IV

SUPERVISED LEARNING FOR DVFS

4.1 DVFS from User Space in Linux

The first step to applying DVFS policies in Linux is to find documentation on how to manipulate it. DVFS configuration in Linux is entirely controlled by a set of text files located in a device folder that holds information about the CPU. This information can be modified directly if given the permission to write to those files, which may have security ramifications, as well as having the governor set to `userspace`. The governor is a written policy on how to handle energy demand from the processor. The table 4.2 lists the available governor types on Linux platforms. The one governor that applies to this thesis is `userspace` since it allows a user to modify the processor's frequency directly. This method was chosen because the tool that was initially used `cpufrequtils` required root permissions and it proved to be cumbersome. `cpufreq-info` is a tool that provides information about the processor and specifically derives its data from the files in Table 4.1. It ended up being much more simple to modify the files directly and possibly more efficient. Table 4.1 shows the various files that can be modified to control things such as frequency and governors. Also worth noting, each core has its own individual files prescribed to them. That means there is complete control over what each core is doing at anytime. The penalty for changing frequency is the transition delay between states. The Intel Xeon processor this thesis uses as its platform is able to run between the

frequencies of 1.2 Ghz to 2.0 Ghz.

Table 4.1: Modifiable files that control DVFS policies in Linux

File Name	Description
scaling_governors	Describes the type of governor currently in use.
scaling_frequency	Describes the current frequency in Hz.
scaling_setspeed	Allows for setting the speed of the processor.
cpuinfo_max_freq	Shows maximum allowed frequency.
cpuinfo_min_freq	Shows minimum allowed frequency.

Table 4.2: Different types of governors supported by Linux

File Name	Description
ondemand	Gives power to the processor as needed.
userspace	Allows User Defined Values for files in Table 4.1.
conservative	Only increases frequency when needed. (Laptops)
powersave	Maintains a power budget and minimum frequencies. (Laptops)
performance	Maintains max power and frequency.

4.2 Set-up for ML-based DVFS

In order to derive a good machine learning model it is necessary to pick the right features. Without the right features a model will suffer. The features selected for the ML model correspond with some of the most effective performance counters that correlate with power. There are four features and they are cache misses, core

utilization, dTLB misses, and FP Intensity. Cache misses refers to the general amount of cache misses on the entire processor. There is nothing discriminatory or specific about the cache miss performance counter such as only collecting L3 cache misses. Core utilization is a percentage of core usage. It is derived by taking the max frequency of the processor and dividing it by the unhalted clock cycles. dTLB is the data translation look-aside buffer. It is a valuable metric for seeing how many memory accesses are happening between addressable caches. Basically, it is great for seeing if the flow of information is causing cache pressure. FP Intensity is a combination of two different performance counters. It represents the amount of floating point operations executed. FP Intensity was later scrapped in order to accommodate the two values that derived that feature. The two performance counters are uncore and fpinst. The uncore value represents actions on the processor that do not have a specified category. Fpinst is the floating point instruction performance counter. The next step is to decide how to classify a success or failure in the context of frequency switching.

After collecting data from the features selected, a heuristic is necessary to classify what is good and what is a bad case. The ideal good case is a configuration in which power consumption is lowered and execution time is also lowered when the frequency is switch to a lower setting. This is not realistic nor physically possible. The realistic good case that is being searched for is one where the penalty in execution time is mitigated against the power savings of making the switch. A bad case is where that balance is not met. In other words, the time penalty is too much

for the power savings to be worth the switch in frequencies. Essentially, the prediction being made is what frequency to run the program or workload at. Since the data collection is in real time if another program is added mid execution the prediction models will account for it.

4.3 Training Data Generation

Features that were selected from set-up are then collected for use in training the models. The data collection process involves some steps. First, wkldgenneeds to be configured to run the programs to be profiled. Once it is established what programs are to be profiled it is necessary to figure out the codewords that reference the performance counters on chip. Both Perf and Likwid have different names for the same performance counter which makes it confusing so, stick with one. The models in this thesis were generated using data collected through Perf but, it is also doable using Likwid. The preference for Perf comes from the large overhead from using Likwid. Also, from experience using Likwid some performance counters exhibit inaccuracies which need to be cleaned up manually. The data that is collected is then processed through R to get an average value from the entire execution of a program. This is done for each frequency step chosen. A table in excel is created with the values generated and exported from R.[R Core Team, 2014] This excel table has three additional columns of information added. There is a column that is the multiplication of power and execution time. The next column is the E_i/E_j ratio and the last column is the classification of the frequency step. There are twenty of

these tables which represents the frequency steps between 1.2 Ghz and 2.0 Ghz. The E_i/E_j ratio is a heuristic in which the multiplied power and execution time value (PxE) is divided by two frequency rows. An example is dividing the 2.0 Ghz PxE value with the 1.8 Ghz PxE value. This generates a ratio that if greater than 1 means the frequency switch is beneficial while a value less than 1 means the switch is not. Beneficial in this context is the power savings worth it in comparison to the execution time penalty. The E_i/E_j ratio will expose what programs benefit from under clocking.

Table 4.3: Frequency table 2.0 Ghz to 1.8 Ghz for workload Canneal/Raytrace

Freq	CacheM	CU%	dTLB	UnCor	FPIns	Time	Power	PxE	E_i/E_j	class
2.0	3979833	66	2774811	6143341	28710112	211	28	5931	1.018	good
1.8	3649193	59	2203711	5679636	26411418	229	25	5825	1.018	good
1.6	3289830	52	1978288	5297132	23706746	255	22	5850	1.018	good
1.4	2864456	46	1685495	4646520	20872948	289	20	5974	1.018	good
1.2	2511332	39	1889318	4197877	18210645	331	18	6162	1.018	good

The frequency tables are then cross-validated in Weka using k-fold.[Witten and Frank, 2005] Cross-validation is important since it gives a picture about how well that data will conform to the models generated. The first models that were generated were using single programs from the PARSEC benchmark suite. This included eleven different programs all with there own characteristics. All of the programs in the suite are parallel applications and at minimum have PThread parallelization. After this was successfully done it was stepped up to Multiprogram

workloads from the same suite. This included combinations of two programs from the suite run at the same time. The percentage accuracy from cross validating using k-fold with a value of 10 was 95%.

CHAPTER V

EXPERIMENTAL RESULTS

5.1 Experimental Setup

There are two computers that were used for this project. Both of them were running Ubuntu 12.04 LTS and as of recently were upgraded to 14.04 LTS. One of them runs an Intel Xeon processor. It is one of the few current processors that supports access to the power performance counter. Their currently is a shortlist of processors that have this specific feature. Another feature of this processor is the ability to disable parts of the processor through the BIOS. Such features as Intel SpeedStep as well as HyperThreading can be disabled. While consumer processors have been adopting more options there is significantly more flexibility with enterprise products.

5.1.1 Platform

All the data collected came from a 12-core Intel Xeon Processor with a 15 MB cache and 4 GB of RAM. The system itself is a Dell Workstation and the Xeon line of processors are meant to be server optimized and enterprise grade. That means they are designed to handle large volumes of tasks hence why it has so many cores that operate at 2.0 Ghz. The other system which was used for power estimation was an Intel Core 2 Quad running at 2.40 GHz. It has a cache size of 4 MB and 4 GB of RAM. It is a consumer grade processor that does not contain features associated with the Xeon processor. The Core 2 Quad system was only used for exploring

power estimation on a system without power performance counters.

5.1.1.1 Benchmark Tools

There are two benchmarking tools that were used for the purposes of this thesis. They are PARSEC and stress. Stress is a Linux program that gives the user the ability to run specific synthetic benchmarks on the system. It has worker threads that run square root calculations, memory accesses from cache, as well as hard drive usage. Stress ended up being a successful platform for testing extreme cases in frequency. It was used as a verification tool for dynamic frequency switching. Stress confirmed the changes in frequency and power consumption from the manipulation of frequency in the Linux environment.

PARSEC is a contemporary multithreaded benchmarking suite that includes various parallel programs provided by different contributors. PARSEC allows the user to run a program which they can apply a series of arguments to such as thread count, data size, and more program specific settings. The suite itself has programs that are compute intensive and memory intensive.

5.1.2 Workloads

Selecting the right workloads to use is essential to having significant results. It means that the programs that are selected need to have some diversity. That diversity comes from multiple factors which include cache usage, computations, regular code usage, irregular code usage, and other features. PARSEC ends up

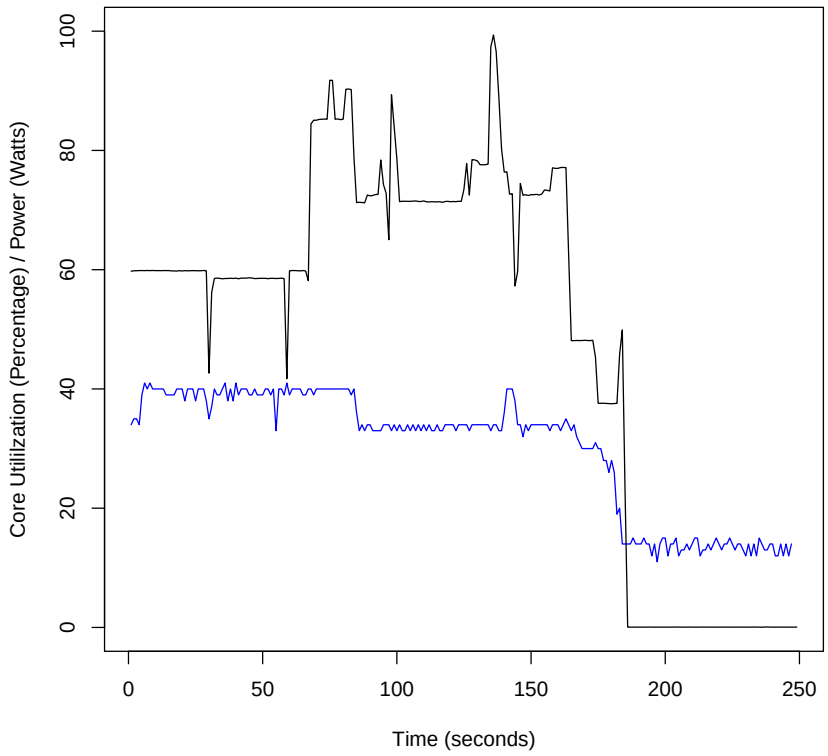


Figure 5.1: Core utilization's relation to power consumption

being the right fit since it provides a fairly comprehensive list of programs that touch various aspects of programming.

5.2 Accuracy of ML Models

Accuracy is an important part of making critical decisions. Weka provides information about the accuracy of a model and a direct way of checking is by looking at the Correctly Classified percentage. Correctly Classified is a verification of the data collected against the model generated. The higher the percentage is the

more accurate the predicted result will be. Since decisions are binary the room for error is minimized. Table 5.1 shows the accuracy of the three machine learning models that were used. The Mp and Sp moniker denotes whether the model is using multiprogram data or single program data. The Sp models use eleven programs in the PARSEC benchmark out of twelve and those eleven are shown in Table 5.2. The Mp data is a subset of possible combinations of two program configurations coming from PARSEC. Those program configurations that were used to train the models are listed in Table **??**. The accuracy of Sp is a few percentage points higher than Mp. This result has to do with the fact that the amount of data fed to Sp models was the entire program space. The Mp models had more to contend with since the amount of configurations expanded and the complexity in program interactions increased. There are still some strange results where minimum accuracy is a concern. SVM Sp and Bayes Sp share minimum accuracies just as DTree Mp and Bayes Mp. Single program workloads and multiprogram workloads benefit the most from a Decision Tree model when evaluating for mean accuracy.

Table 5.1: Accuracy of machine learning models for multiple and single programs

	Bayes Mp	Bayes Sp	Dtree Mp	Dtree Sp	SVM Mp	SVM Sp
Mean Accuracy %	94	97	95	98	93	96
Min Accuracy %	85	89	85	94	80	89
Max Accuracy %	100	100	100	100	100	100

5.3 Energy Efficiency

The next step after the machine learning models are generated is to apply them to an experiment. The values in table 5.2 show interesting results, specifically, the last two programs in the set have power savings that are on par with the time penalty. Those programs are memory intensive applications that require streaming in data. Canneal needs to read an entire input file before it is executed and streamcluster dynamically classifies information that is streamed. Canneal and streamcluster are both rated to run at the lowest frequency possible according to the models generated. The power savings percentage is derived from the average power consumption over the entire run of the program as reported by Likwid.

Multiprogram workloads have significant differences from single program workloads. There are some unexpected results if compared with the single program results. The execution time penalty and power savings gain for most multiprogram workloads were close. The exeception was the block of values in the x264 section of the table. In those cases, the power savings is doubled compared to the time penalty percentage.

5.3.1 Single Program Workloads

Single program workloads do not seem to exhibit exploitable characteristics that benefit energy efficiency. Also, it is not apparent what combinations of programs could benefit from sDVFS heuristic. The reason for this has to do with how the

Table 5.2: Power savings and execution time penalty for single programs using SVM model

Program Name	Model	Power Savings	Execution Time Penalty
*** bodytrack ***	svm	-10.26%	-4.43%
*** blackscholes ***	svm	-4.30%	-27.15%
*** ferret ***	svm	2.34%	-18.79%
*** facesim ***	svm	0.87%	-27.63%
*** fluidanimate ***	svm	5.13%	-31.08%
*** freqmine ***	svm	2.15%	-35.10%
*** raytrace ***	svm	-4.61%	-26.75%
*** swaptions ***	svm	1.33%	-26.35%
*** x264 ***	svm	2.27%	-33.10%
*** canneal ***	svm	19.95%	-22.07%
*** streamcluster ***	svm	15.28%	-14.62%

model was trained and classified. The classification technique will take a ratio that benefits power and since the execution time penalty was not significant enough it chose low frequencies. The average power savings across all the programs and all the models is 3.18%. The average execution time penalty is -23.87%. While these results are not promising the addition of more programs yields better results25

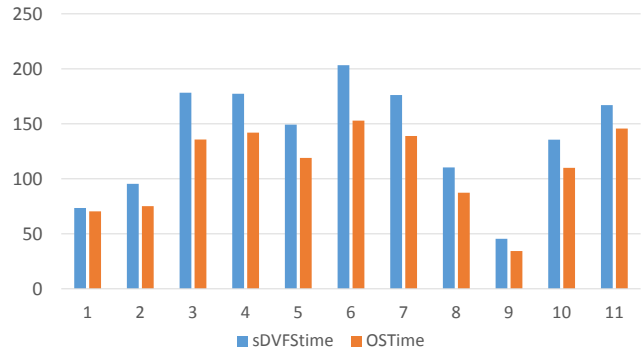
5.3.2 Multi Program Workloads

Multi program workloads had the biggest benefit from the smartDVFSheuristic. Trends that are visible in Table 5.2 do not have much bearing on the multiprogram

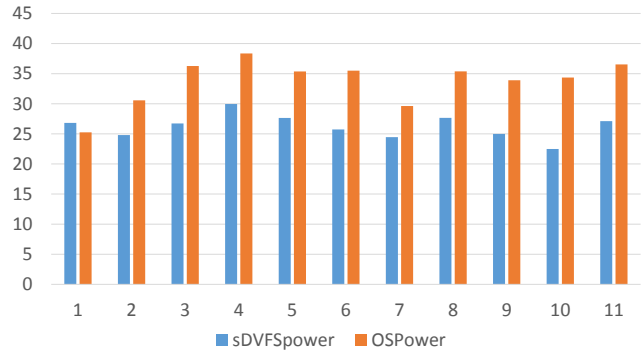
Table 5.3: Power savings and execution time penalty for multi programs using DTree model

Program Name	Model	Power Savings	Execution Time Penalty
*** x264 bodytrack ***	DTree	12.62%	-11.38%
*** x264 blackscholes ***	DTree	20.56%	-11.83%
*** x264 ferret ***	DTree	8.76%	-11.76%
*** x264 facesim ***	DTree	8.59%	-9.25%
*** x264 fluidanimate ***	DTree	23.40%	-11.34%
*** x264 freqmine ***	DTree	23.46%	-11.05%
*** x264 raytrace ***	DTree	27.13%	-10.82%
*** x264 swaptions ***	DTree	21.39%	-11.30%
*** x264 x264 ***	DTree	26.00%	-10.40%
*** x264 canneal ***	DTree	20.47%	-7.50%
*** x264 streamcluster ***	DTree	9.50%	-5.87%
*** canneal raytrace ***	DTree	0.71%	-9.28%
*** canneal swaptions ***	DTree	-3.42%	-12.27%
*** canneal x264 ***	DTree	4.72%	-5.21%
*** canneal canneal ***	DTree	5.86%	-7.13%
*** canneal streamcluster ***	DTree	-0.17%	-7.38%
*** streamcluster raytrace ***	DTree	-4.04%	-8.29%
*** streamcluster swaptions ***	DTree	2.83%	-7.65%
*** streamcluster x264 ***	DTree	8.68%	-7.34%
*** streamcluster canneal ***	DTree	0.42%	-5.98%
*** streamcluster streamcluster ***	DTree	4.68%	-6.28%

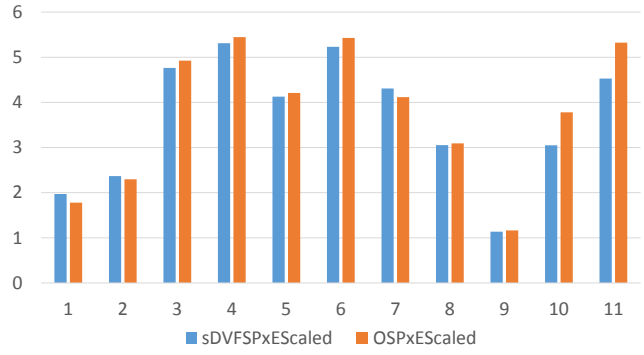
results. But, what is noticeable is the performance of x264 in combination with a few other programs. In fact the power savings on average across all the programs and all models beats the execution time penalty. The average power savings is 10.52% and the average execution time penalty is -9.02%.



(a) Bayes Execution Time

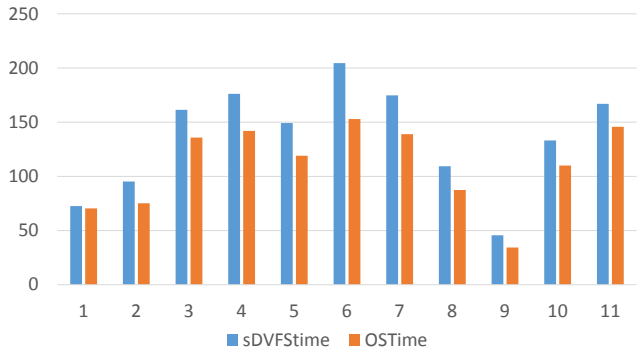


(b) Bayes Power

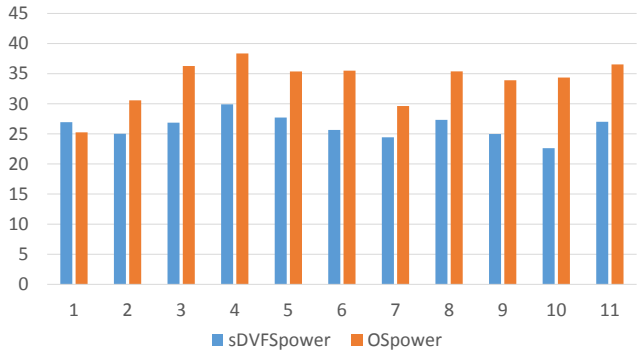


(c) Bayes PxE Scaled

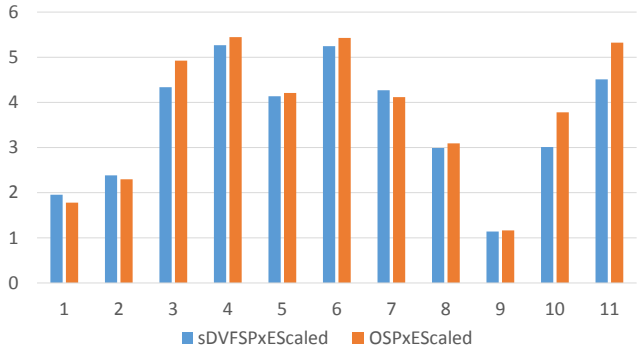
Figure 5.2: Single program results for Bayes model



(a) Decision Tree Execution Time

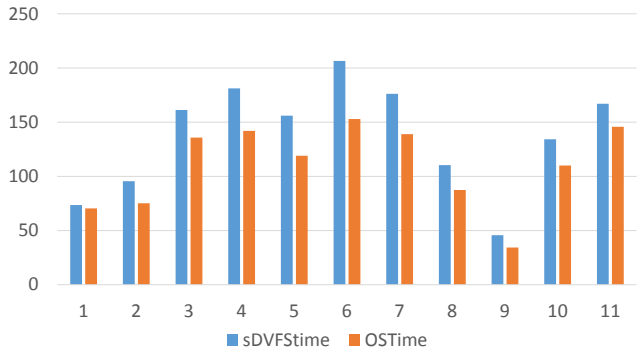


(b) Decision Tree Power

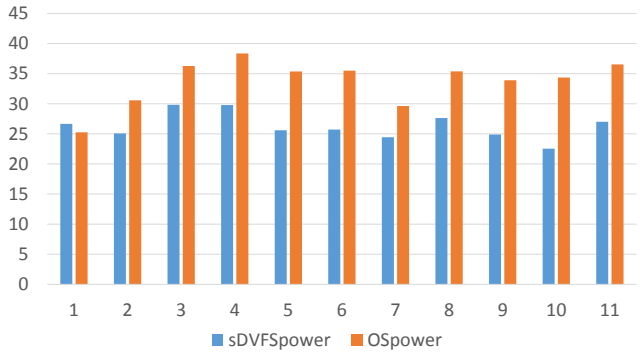


(c) Decision Tree PxE Scaled

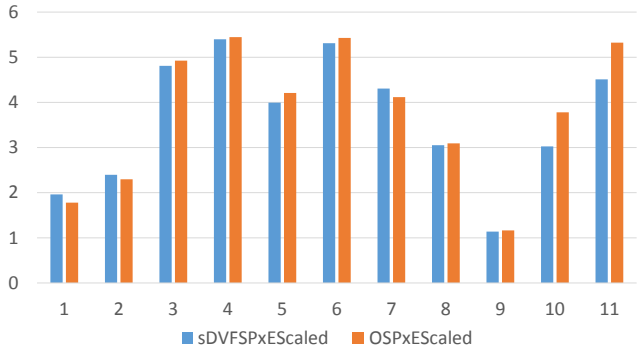
Figure 5.3: Single program results for DTree model



(a) SVM Execution Time

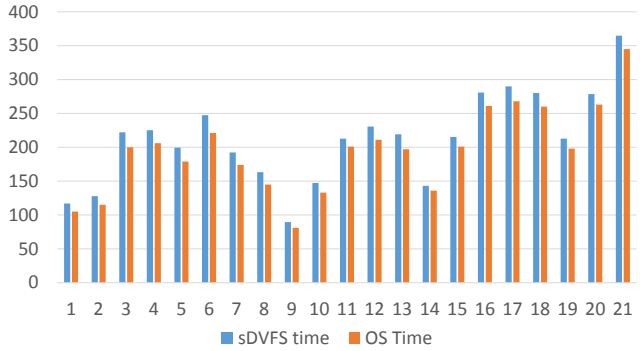


(b) SVM Power

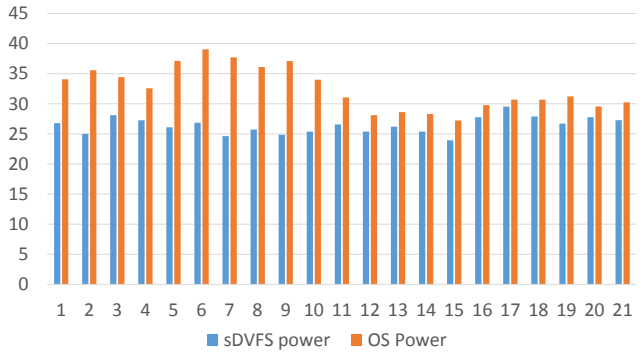


(c) SVM PxE Scaled

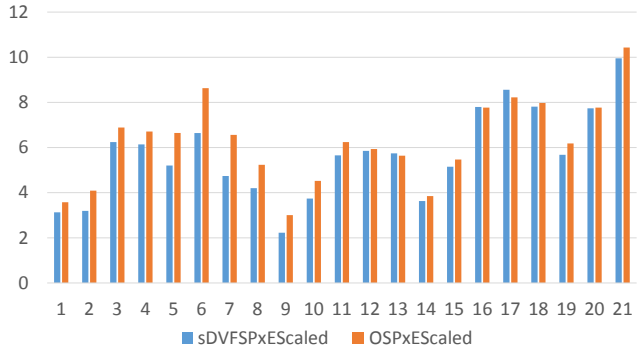
Figure 5.4: Single program results for SVM model



(a) Bayes Execution Time

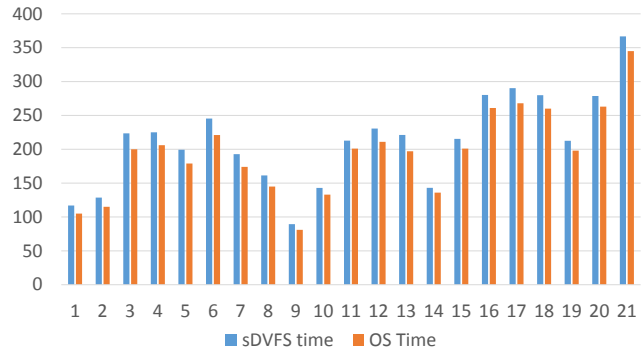


(b) Bayes Power

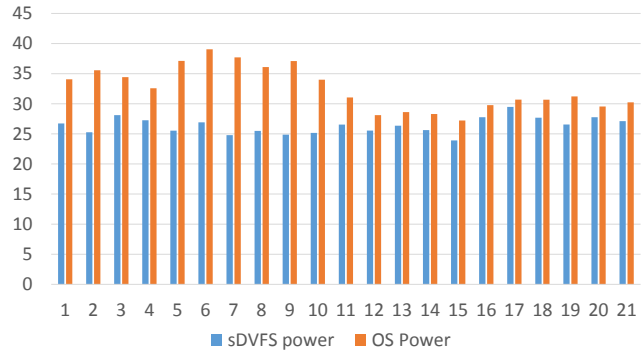


(c) Bayes PxE Scaled

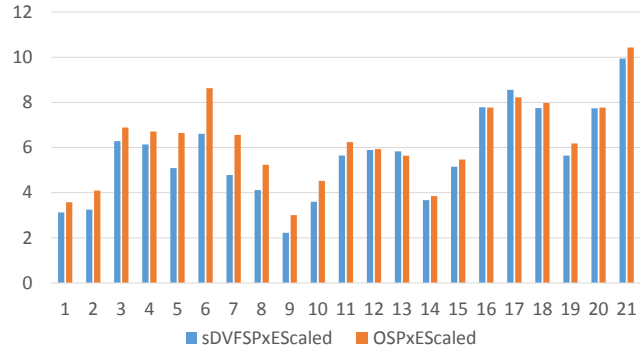
Figure 5.5: Multi program results for Bayes model



(a) Decision Tree Execution Time

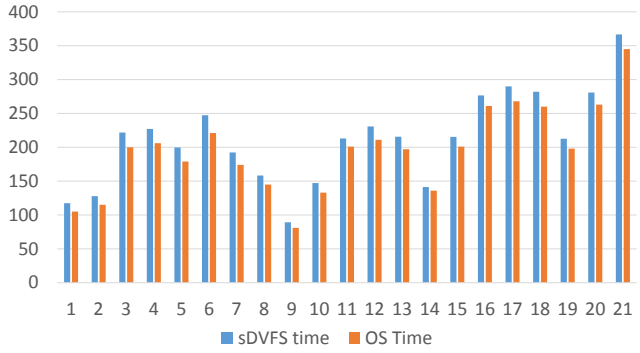


(b) Decision Tree Power

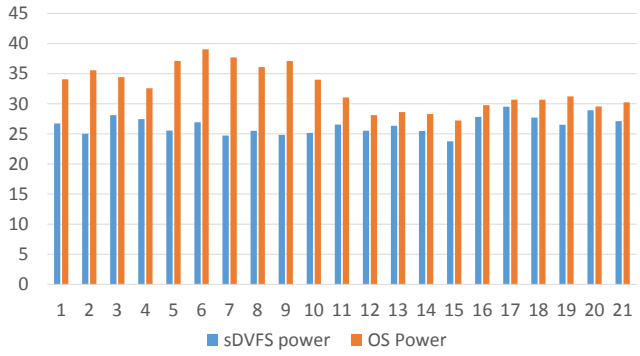


(c) Decision Tree PxE Scaled

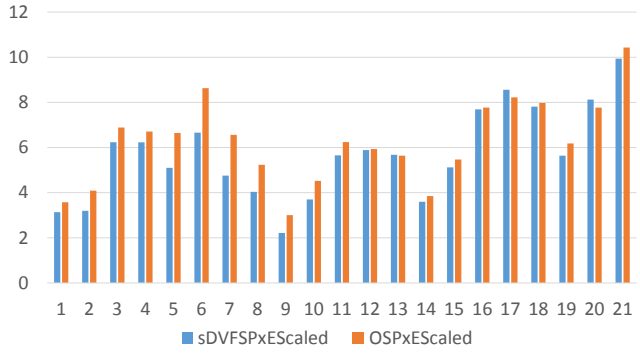
Figure 5.6: Multi program results for DTree model



(a) SVM Execution Time



(b) SVM Power



(c) SVM PxE Scaled

Figure 5.7: Multi program results for SVM model

CHAPTER VI

CONCLUSION

The results are surprising in that it is possible to obtain power savings that significantly outweigh the execution time penalty. By feeding the right program characteristics into ML models `smartDVFS` was able to produce frequency selections that are beneficial to power while minimizing damage to execution time. There is a possibility that exploring the whole search space for PARSEC two program workloads might generate further improvement in power consumption. The framework is a foundation in which to explore possible power saving strategies throught smart heuristic evaluations.

BIBLIOGRAPHY

Bienia, C., Kumar, S., Singh, J. P., and Li, K. (2008). The parsec benchmark suite: characterization and architectural implications. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, PACT '08, pages 72–81, New York, NY, USA. ACM.

Jung, H. and Pedram, M. (2010). Supervised learning based power management for multicore processors. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 29(9):1395–1408.

Li, S., Ahn, J. H., Strong, R. D., Brockman, J. B., Tullsen, D. M., and Jouppi, N. P. (2009). Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, pages 469–480, New York, NY, USA. ACM.

R Core Team (2014). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Shen, H., Lu, J., and Qiu, Q. (2012). Learning based dvfs for simultaneous temperature, performance and energy management. In *Quality Electronic Design (ISQED), 2012 13th International Symposium on*, pages 747–754.

Singh, K., Bhadauria, M., and McKee, S. A. (2009). Real time power estimation and thread scheduling via performance counters. *SIGARCH Comput. Archit. News*, 37(2):46–55.

Treibig, J., Hager, G., and Wellein, G. (2010). Likwid: A lightweight performance-oriented tool suite for x86 multicore environments. In *Proceedings of PSTI2010, the First International Workshop on Parallel Software Tools and Tool Infrastructures*, San Diego CA.

Witten, I. H. and Frank, E. (2005). *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, San Francisco, 2nd edition.