SAND2022-12636
LDRD PROJECT NUMBER: 227331
LDRD PROJECT TITLE: Runtime Systems for Energy Efficiency in Advanced Computing Systems

PROJECT TEAM MEMBERS:
Curtis Madsen (Sandia National Laboratories), Tian J. Ma (Sandia National Laboratories),
Dipayan Mukherjee (University of Illinois at Urbana-Champaign), Gul Agha (University of Illinois at Urbana-Champaign)

ABSTRACT

As heterogeneous systems become increasingly popular for both mobile and high-performance computing, conventional efficiency techniques such as dynamic voltage and frequency scaling (DVFS) fail to account for the tightly coupled and varied nature of systems on a chip (SoCs). In this work, we explore the impact of system unaware DVFS techniques on a mobile SoC under three benchmark suites: Chai, Rodinia, and Antutu. We then analyze performance trends across the suites to identify a set of consistent operating points that optimally balance power and performance across the system. The consistent operating points are then constructed into a dependency graph which can be leveraged to produce a more effective, SoC-wide governor.

LABORATORY DIRECTED RESEARCH & DEVELOPMENT
WHERE INNOVATION BEGINS

INTRODUCTION AND EXECUTIVE SUMMARY OF RESULTS

Power consumption is an important issue in the design of computing systems. In recent years, there have been two developments which have created a need for energy aware computing: High Performance Computing (HPC): new HPC cluster systems are designed to have a power cap [1], [2] which allows the system to scale with the increasing demand, and Mobile Computing: proliferation of battery driven mobile devices has given rise to the need for meeting ever increasing demand for performance with a longer battery life, which requires focus on energy-aware computing. Dynamic voltage and frequency scaling (DVFS) and Dynamic Duty Cycle Modulation (DDCM) are highly effective techniques for reducing system power dissipation [1], [3]. The key idea behind these techniques involves providing the system with just-enough speed for task completion, thereby reducing energy consumption quadratically. In addition to its energy saving characteristics, voltage and frequency modulation help in regulating the temperature of the cores which helps reduce system failure [4].

DVFS has been an active area of research, and the work can be broadly categorized into three categories [5]. The first category of work focuses on designing models to evaluate the power consumption of various programs. This area of work helps designers understand the impact of DVFS on various algorithms and helps software developers in designing efficient algorithms which are cognizant of both performance and power. The second category of work focuses on system-level design that tries to utilize system-level statistics of given resources from their governors and decides the operating points for the resources. Finally, the third area involves designing energy aware task scheduling, where based on the requirements of multiple processes and multi-core systems, jobs are scheduled on different machines to ensure maximum performance at minimum energy cost [6], [7].

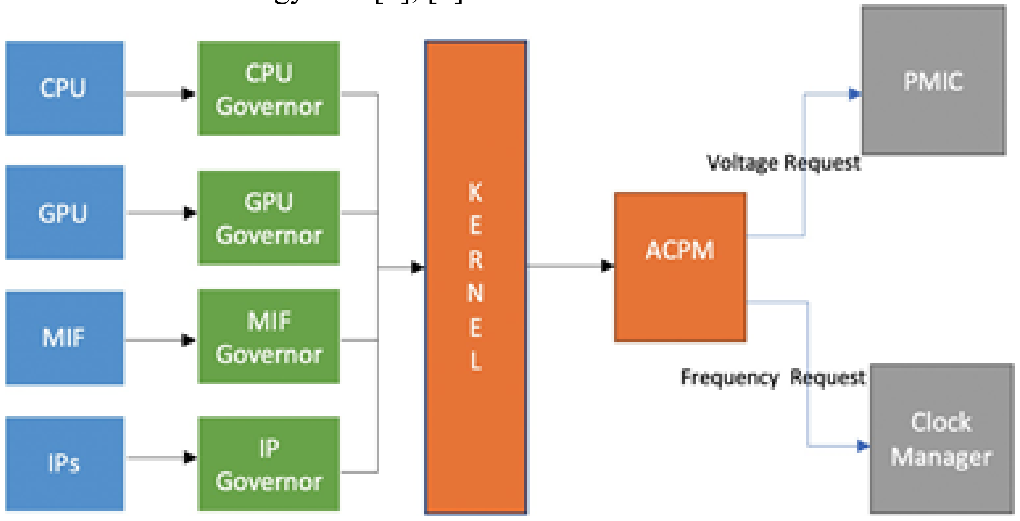


Figure 1 Architecture Schematic of DVFS in Current Systems

In current DVFS systems, all requests for state transitions are generated through the respective governors and sent to the kernel, where they are serviced by the power management unit. A schematic of this architecture is shown in Figure 1. The operating points of the resources (CPU, GPU, Memory) are decided by the governor of that resource based on its state [8], unaware of the requirements of other resources and the complete system level power demands. The lack of cross-component awareness leads to sub-optimal policies and additionally affects the energy efficiency and the quality of service (QoS) of the system [9].

To address this problem, we develop a holistic multi-component DVFS mechanism utilizing the cross-component relationship among the resources. Our approach accounts for dynamically changing input data when making power predictions and does so at the fine-grained granularity of each operator. Achieving these objectives is challenging because there are no tools to measure the power cost of each operator directly, and we do not have visibility into operator code to understand its resource usage. Instead, we collect easily accessible device-level energy draw information on the mobile device (i.e., system on a chip (SoC)) and concurrently monitor changing composition of executing application operators to attribute measured power draw in a time interval to the observed operators within that interval. A prediction model, which directly maps operator execution times to their corresponding power use, is then trained using this data and deployed on the mobile device to provide run-time predictions for power optimization decisions. Using these results, we are able to build a dependency graph among components and create power or performance equivalence classes that are used to intelligently modify the frequencies of each component to achieve a desired power and performance for the whole system.

In this work, we demonstrate that in a multi-component system of independent governors leads to sub-optimal energy and performance for the system. We study the relationship of varying operating points over different resources, which allows us to characterize the interactions among components embedded in a modern SoC architecture. Our work can be used to set individual component operating points as well as decide the transition path between points in order to achieve power savings in the system with minimal impact to performance. Through our research and development, we were able to achieve and exceed our goal of 10% power savings when comparing our approach to traditional DVFS methods.

METHODOLOGY

We have two primary design goals: 1) identify the cross-component dependency of power performance among resources, and 2) create a system-wide DVFS policy for efficient power-performance behavior. The first step toward identifying the power models across each resource is to understand the relationship among them and how the power and performance of individual resources are intertwined. Secondly, we propose a method to identify the efficient and inefficient zones across various components, and furthermore, provide an architecture for system-wide DVFS policy.

POWER MODEL

In modern resources, the power is composed of two components: static and dynamic power. Static/leakage power is a result of an unwanted sub-threshold current (I_L) in the transistor channels, which is a direct result of transistor characteristics, and is found in almost all transistor-based circuitry. Dynamic power is the more dominant component of the total power and is impacted by the voltage and frequency setting of the resource as expressed below [6]:

$$P = P_{dynamic} + P_{static}$$
$$P_{dynamic} = C_L V^2 f$$
$$P_{static} = I_L V$$

where P is the total power consumed, P_{static} is the leakage power, and $P_{dynamic}$ is the dynamic power. C_L and I_L are constant values, V is the operating voltage, and f is the operating frequency. While most modern systems follow the behavior expressed in the equations above, some resources have slightly different behavior, such as memory systems which use the following model:

$$P_{mem} = \frac{1}{2} C_L V^2 f \times access\_count + I_L V$$

where $access\_count$ is the concurrent memory access made to the resource.

Voltage and the frequency are tightly coupled and have almost linear correlation (Figure 2), to ensure safe operation of the devices. In modern systems each frequency is one-to-one mapped to a specific voltage along the Safe Boundary, which allows us to modify the operating voltage by changing the frequency. This relationship reduces the state space of our problem making it easier to focus on the impact of operating frequency of one resource on another.

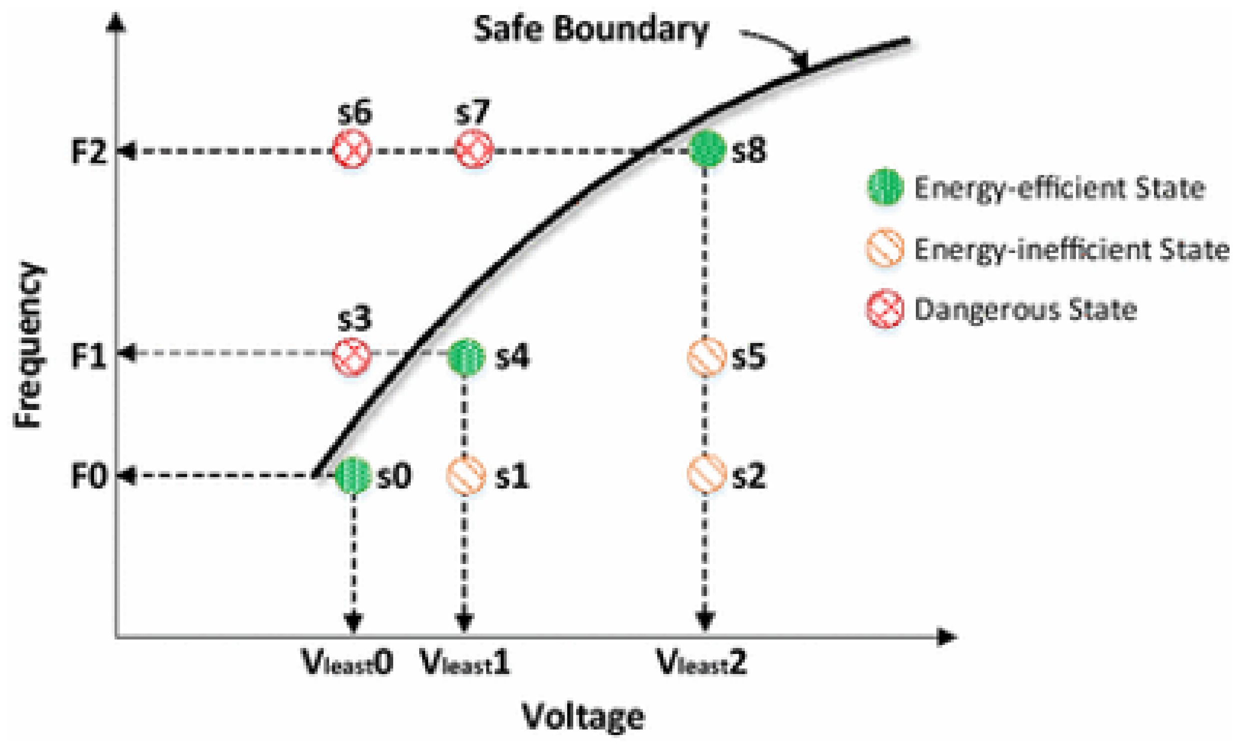


Figure 2: Diagram Depicting Safe Operating Point

DEPENDENCY GRAPH

Performance of resources is generally linearly correlated with the operating frequency of the resource. An operating point in a multi-resource system is represented as a tuple of operating frequencies, $\{f_{CPU}, f_{GPU}, f_{Mem}\}$.

Current DVFS policies for multi-resource systems do not consider the interaction among resources in a tightly coupled system like SoCs, and the heterogeneous workload of modern applications where each resource impacts the performance of others. Due to this interaction, certain configurations can lead to less wasted power while still achieving similar performance. We represent the dependency of operating points in a graphical structure (Figure 3), where different colors represent different resources, and each node is an operating frequency. Equivalence classes represent a collection of operating point which provides similar power-performance characteristics. A different class represents system requirements such as high performance or power efficiency. These dependency graphs will help developers decide the optimal configuration based on QoS requirements and other operating constraints.

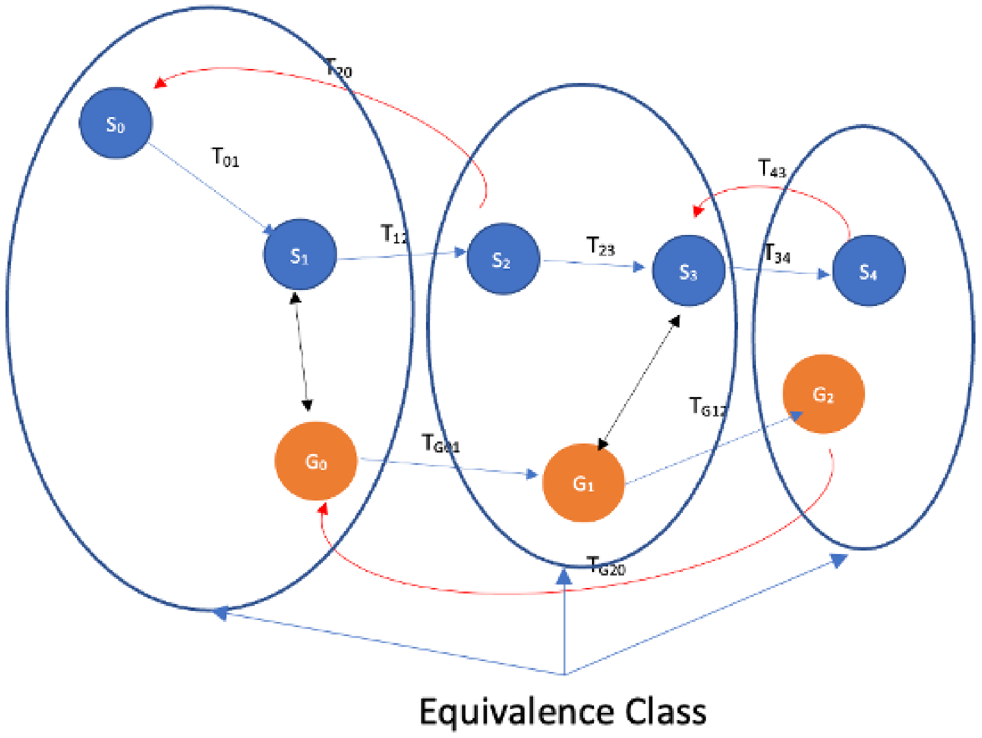


Figure 3: Schematic of an Example Dependency Graph

To design the dependency graph, we run the benchmarks for all possible configurations of voltage and frequency of each component and collect the benchmark and utilization numbers for each of the components. This allows us to characterize the interactions among different components.

DVFS POLICY

As expressed in the dependency graphs, we utilize the relationship among resources to design a system-wide holistic power management system. The various equivalence classes provide us with a set of operating points to choose from, dependent on the system requirement and operating constraints (such as low power mode or performance mode).

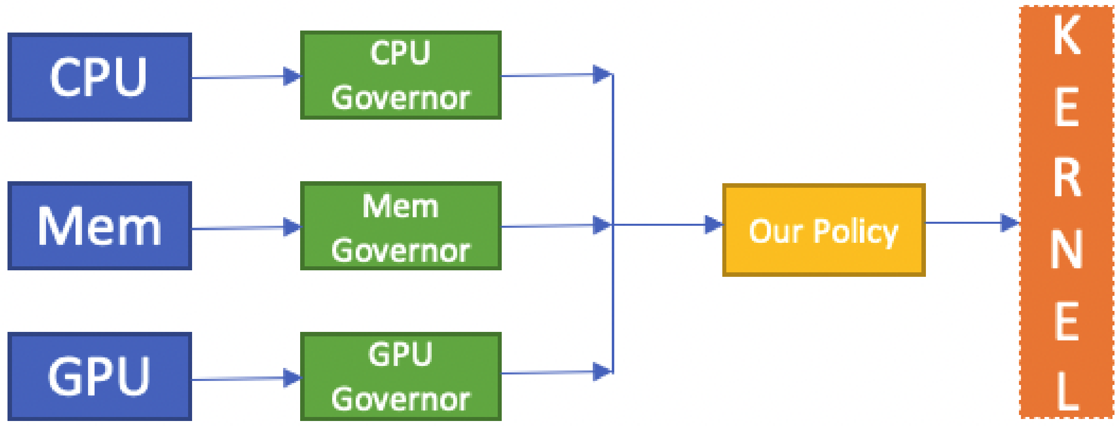


Figure 4: Schematic of Our Proposed Power Management System

We implement this policy on top of existing resource-specific governors as shown in Figure 4. Our DVFS policy adds a layer between existing resource-specific governors. We extract the resource utilization values by using either system provided methods (e.g., proc files) or the APIs provided by the resource specific governors. This model provides a global view of the system and allows us to set the system configuration per the system-wide optimal setting.

RELATED WORK

As DVFS for heterogeneous systems is a fruitful research area, there are several works related to our proposed project. GPU DVFS in isolation has been previously considered as GPUs consume more power and have larger voltage ranges than CPUs [10]. To explore GPU DVFS, Nath and Tullsen [10] present CRISP (Critical Stalled Path), an analytical tool used to understand how frequency impacts GPU application performance. At the time, state-of-the-art CPU performance frequency analysis tools split an application into two portions: pipelined computation and memory fetches. These CPU-oriented tools make the assumption that performance of the former scales with the frequency, while the latter does not, and uses this information to understand how to scale the frequency given an application's characteristics. CRISP is built on the understanding that GPUs do not exhibit this behavior, as the computation in a GPU is highly overlapped with outstanding memory requests. Therefore instead, CRISP splits a GPU application into the load critical path and the compute/store path. Where the former is the longest string of dependent loads which is overlapped with computation, while the latter is defined as the accumulation of non-overlapped compute and store operations. These two portions are important to distinguish as they are impacted by GPU frequency differently. This understanding of the GPU architecture allows CRISP to predict performance within 4% of empirical measurement as frequency is scaled.

Other works have looked into using DVFS as a means to reduce DRAM energy. This is attractive as it has been found that DRAM contributes up to 40% of a server's total energy consumption: a consequence of architects historically focusing their efforts on the CPU as well as the increased demand of memory bandwidth. Deng et al. [11] present MemScale, a scheme which proposes both DVFS of the memory controller and dynamic frequency scaling (DFS) of the physical DRAM chips. MemScale is developed using the concept of slack, defined as the difference of a desired runtime provided by the user and the actual runtime when the frequency is scaled. MemScale uses many performance counters to understand how the performance of an application is impacted by the memory controller DVFS and memory DFS, with the goal of minimizing the overall slack. MemScale is epoch-based, and every 5ms, the performance counters are profiled and application performance is predicted. After profiling, the frequency and voltage are scaled to reduce the slack of the application. While MemScale only alters the frequency of the DRAM device, there are proposed techniques to also alter the voltage. Chang et al. [12] propose Voltron, which aims to reduce the energy consumption of DRAM by reducing the supply voltage below the DRAM standards while still correctly storing the data. Another area of interest in memory DVFS is the granularity at which operations can be done. Today's systems only support power management at the rank-level, which experiences frequent wake-ups from low-power states due to the coarse granularity. GreenDIMM [13] is a proposed system which allows the operating system to control the power state of DRAM at a sub-array granularity.

Work that is most similar to our project is a follow-on project to MemScale, called CoScale [14]. CoScale looks at the combined problem of CPU and memory DVFS, and shows that these problems can not be looked at individually. For example, naively combining a CPU DVFS and memory DVFS scheme ignorantly can create scenarios where the performance counters used to control the DVFS schemes do not represent the actual application characteristics. CoScale addresses this by using a single scheme, similar to that of MemScale, but controls the voltage and frequency of both the CPU and memory. One issue with looking at both CPU and memory DVFS is the search space is much larger than when looking at a single component. To address this, the authors use a gradient search heuristic to search the space of CPU and memory voltage and frequency.

While the area of DVFS for heterogenous systems has clearly been explored, no work has looked at how CPU, Memory, and GPU frequency and voltage impact end-to-end application performance. The interaction of voltage and frequency between all three components is very important to understand to reduce the energy consumption of modern SoCs.

RESULTS AND DISCUSSION

A. TEST PLATFORM

To explore the efficacy of current governors, we use the ODROID-XU4 Single Board Computer. This board houses a Samsung Exynos 5422 SoC, and its specification is summarized in Table I. The ODROID-XU4 importantly provides DVFS access and control over both CPUs, the GPU, and the main memory. Our goal is to capture the impact that scaling each component's frequency has on several representative workloads. This process will help us identify where performance degradation occurs precisely, construct a graph that captures those dependencies, and potentially design a unified governor which avoids these shortcomings. Our system can run both Ubuntu 18.04 with Linux kernel 4.14 as well as Android 4.4.4 (v7.1) and odroidxu4-4.9.y. Both system software configurations were used allowing us to run multiple benchmark suites.

Table 1: EXYNOS 5422 SPECIFICATIONS

CPU	Octa ARM Cortex-A15 Quad 2GHz and Cortex-A7 Quad 1.3GHz
GPU	Mali-T628 MP6 (OpenGL ES 3.0/2.0/1.1 and OpenCL 1.1 Full profile)
Main Memory	2GB LPDDR3 RAM at 933MHz (14.9GB/s memory bandwidth)

B. BENCHMARKS

In order to generate an understanding of how frequency scaling impacts SoC performance, we require a comprehensive evaluation of operating point efficacy under a variety of memory, CPU, and GPU-bound workloads. To obtain this evaluation, we scaled the frequency of each component while utilizing the AnTuTu benchmark suite. The AnTuTu benchmark suite is a popular mobile benchmark suite, which provides raw component scores for CPU, GPU, and Memory. AnTuTu also provides a User Experience (UX) score, which is an aggregate score describing how the different component performances would impact user experience.
On top of AnTuTu, we wanted benchmarks that utilize the entire SoC for a single application. To achieve this, we used the Rodinia benchmark suite [15], [16], a benchmark suite for heterogeneous platforms. The Rodinia benchmark suite contains a large number of applications with support for OpenCL and OpenMP, enabling applications to execute on the GPU, the CPU and GPU, or multiple CPUs.

One gap in Rodinia is that applications execute either on the CPU or the GPU without cross-device threading or collaboration. To examine scenarios where both the CPU and the GPU are completing useful work simultaneously, we used the Chai benchmark suite [17]. Benchmarks in Chai collaborate through data partitioning, fine-grain task partitioning, or coarse-grain task partitioning.

Together, these three benchmarking approaches will provide data for system performance under varied workloads, including CPU-only, GPU-only, and combined CPU-GPU. These workloads consist of both memory and compute bound benchmarks.

C. ANTUTU

The AnTuTu benchmark suite provides a comprehensive analysis of our system. It provides a detailed breakdown of the different operating resources' performance metrics such as CPU, GPU, Memory, and UX. This breakdown helps us observe the performance discrepancies in each resource with its corresponding operating points against the overall performance score. AnTuTu has different benchmark suites, which utilize a single resource bound task or a multi-resource bound task which allows us to gauge the impact of operating points on cooperative and isolated workloads.

We performed extensive testing with AnTuTu in Android OS at different CPU, GPU, and Memory operating points. We varied the frequency of the big and little CPU from 1.2 - 2.0 GHz and 1.0 - 1.4 GHz, respectively. For the memory, we varied the frequency from 165 - 933 MHz, and for the GPU, the frequency was varied from 177 - 543 MHz. We observed high correlation between performance of big and little core frequency; hence, we modified their frequency together to reduce the number of possible configurations and to study the interaction across different components more effectively.

C.1 FREQUENCY SWEEP

We observed a few interesting trends on the impact on performance due to memory and CPU frequency. As shown in Figure 5, there are local maxima for performance while sweeping memory frequencies dependent on the CPU frequency. The local maxima points shift towards higher memory frequency as the CPU frequency increases which implies that higher CPU frequency does not yield improved performance for lower memory frequency. We also observe that memory frequency has a high impact on GPU performance, and lower memory frequency can lead to saturation and power wastage for higher GPU frequency (Figure 6).

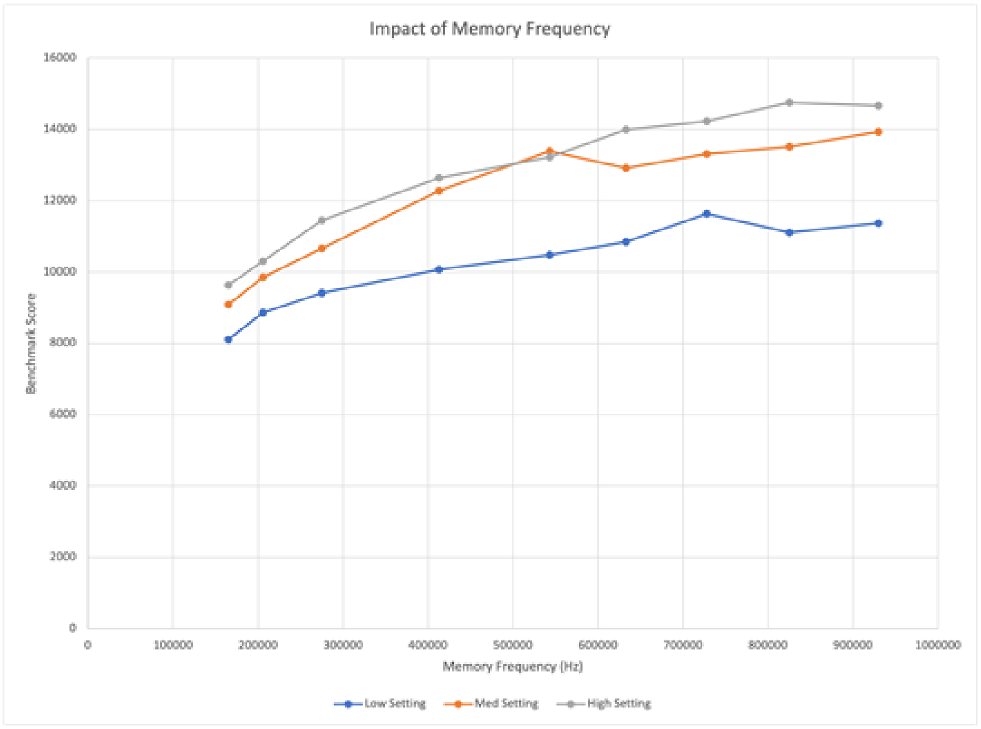


Figure 5: Frequency Sweep for Memory

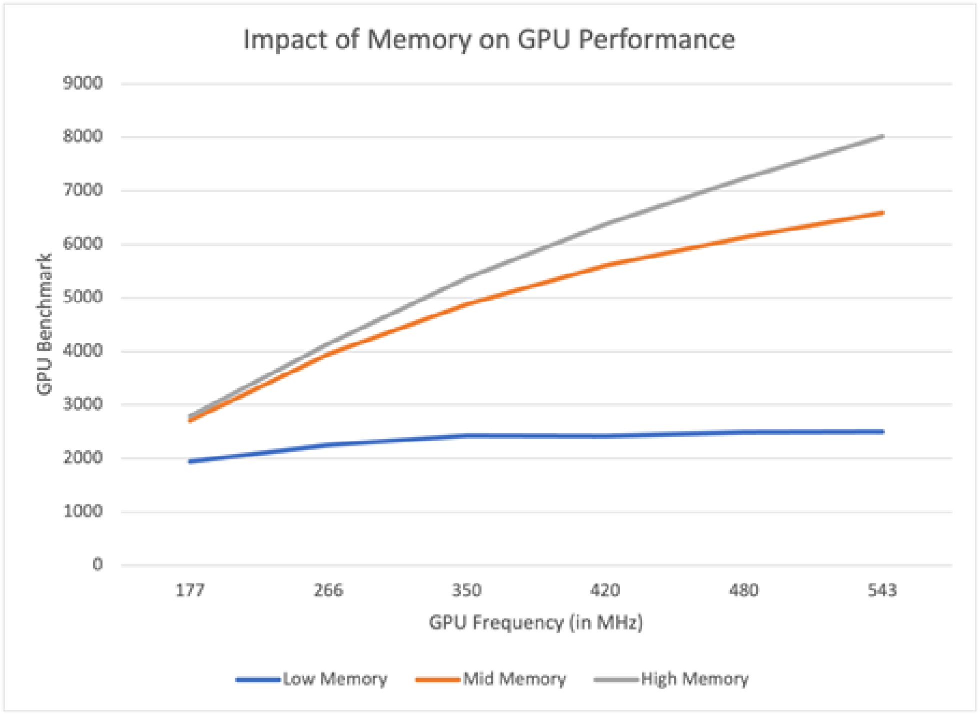


Figure 6: Impact of Memory Frequency on GPU

In Figure 7, we analyze the memory performance scores with varying memory frequencies at the two different CPU and three different GPU frequency levels. For low memory frequencies, we observe identical scores across all CPU and GPU levels while at mid and high memory frequencies, memory performance either stagnates or drops for different GPU frequency levels across all CPU levels.

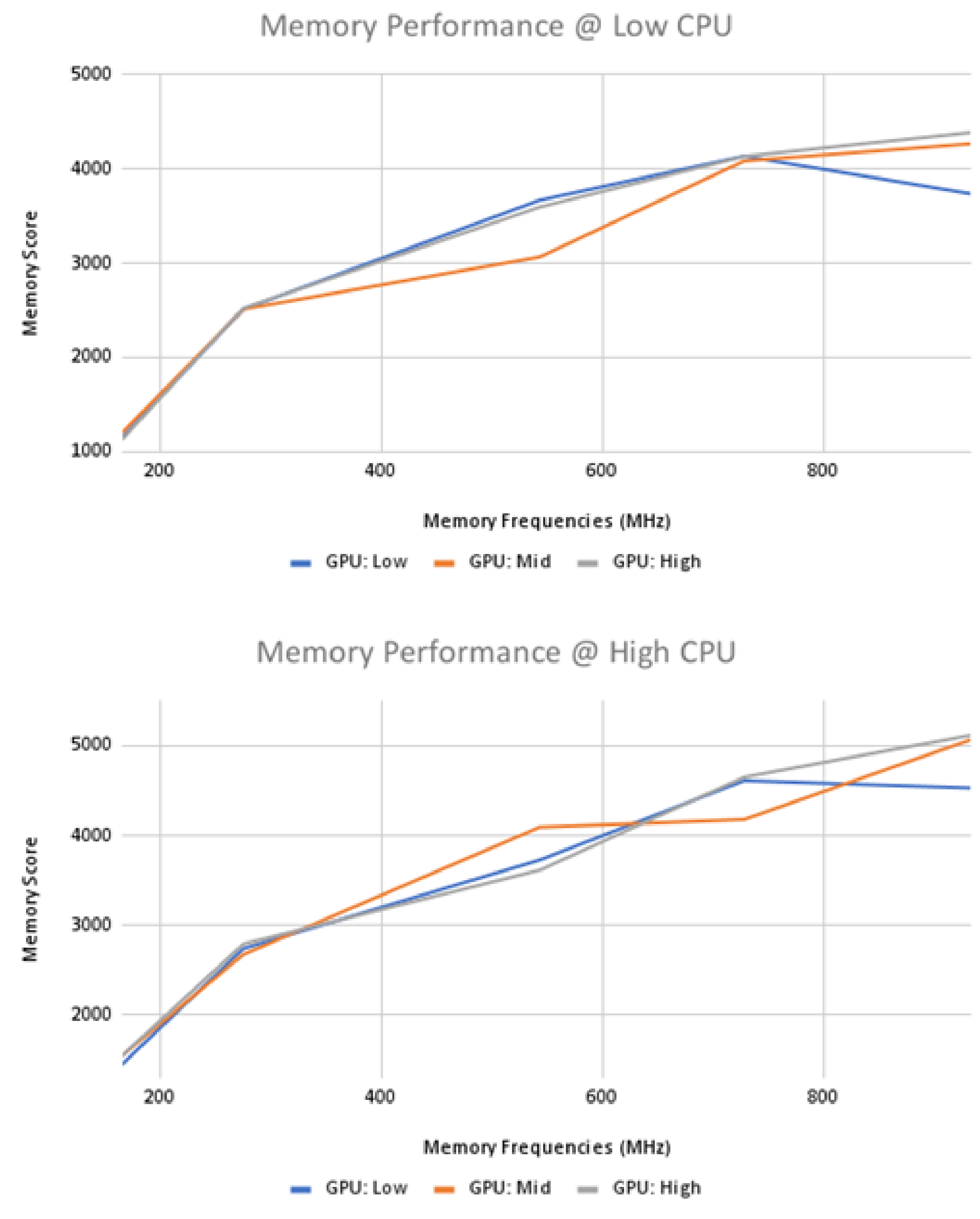


Figure 7: Memory Performance for Different CPU and GPU Settings

Figure 8 shows clusters of CPU performance scores for different CPU frequencies. We noticed clusters in the graph where the CPU performs efficiently or inefficiently: red boxes denote inefficient regions while green boxes denote efficient regions. From this analysis, we noticed that lower CPU frequencies can yield better CPU performance than a higher CPU setting in some cases. This behavior can be attributed to different GPU and memory frequencies for each data point.

These results help us in developing a dependency graph, like the one shown in Figure 3, which encapsulates the effect of operating points on the performance of the system. In the next section, we study the power consumption at different operating points and its impact on overall performance which allows us to analyze the power-performance trade-offs that are dependent on the system and applications' requirements.

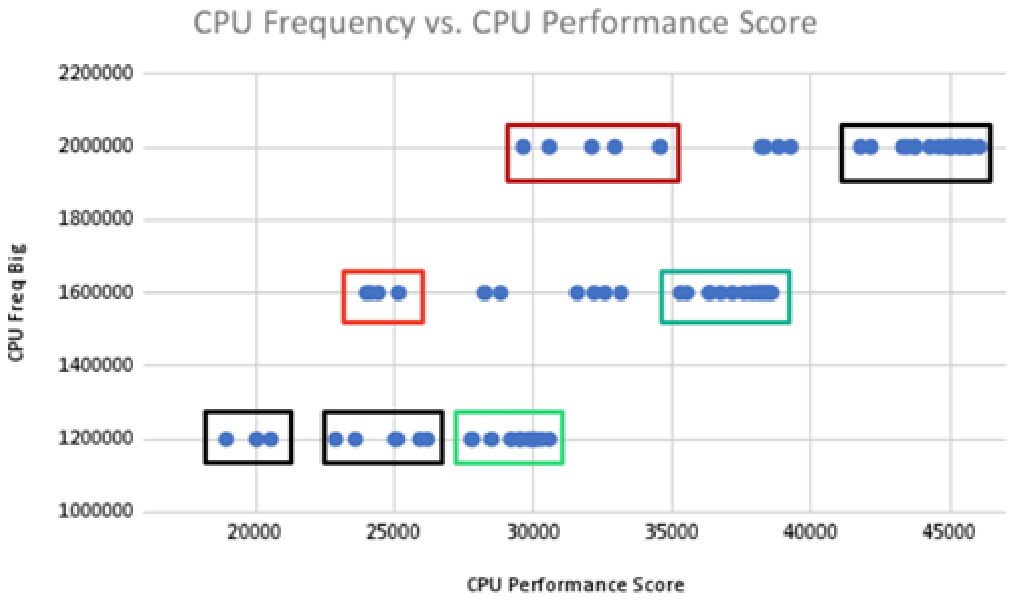


Figure 8: CPU Performance Scores at Different CPU Configurations

C.2 POWER PERFORMANCE ANALYSIS

One of the primary goals of DVFS is to reduce the operating power. Therefore, we studied power performance characteristics of the Odroid-XU4 in an attempt to understand if there are inefficiencies in operating power and performance for different configurations. To achieve this, we ran all benchmarks at all possible configurations and noted the peak power consumption by each benchmark suite.

We observe that power and performance do not have a strictly positive relationship. These results imply a complex dependency among power and performance leading to efficient and inefficient operating zones based on the system wide configurations. This phenomenon is demonstrated in Figures. 9, 10, and 11 where efficient zones are represented by blue and inefficient zones are represented by red. These results are consistent with findings observed in the previous section where GPU and memory performance show saturation at various operating points. In general, we observe that if any of the resources are operating at one extreme while the others are at the opposite extreme, there exist regions of wasted energy without significant performance gains.

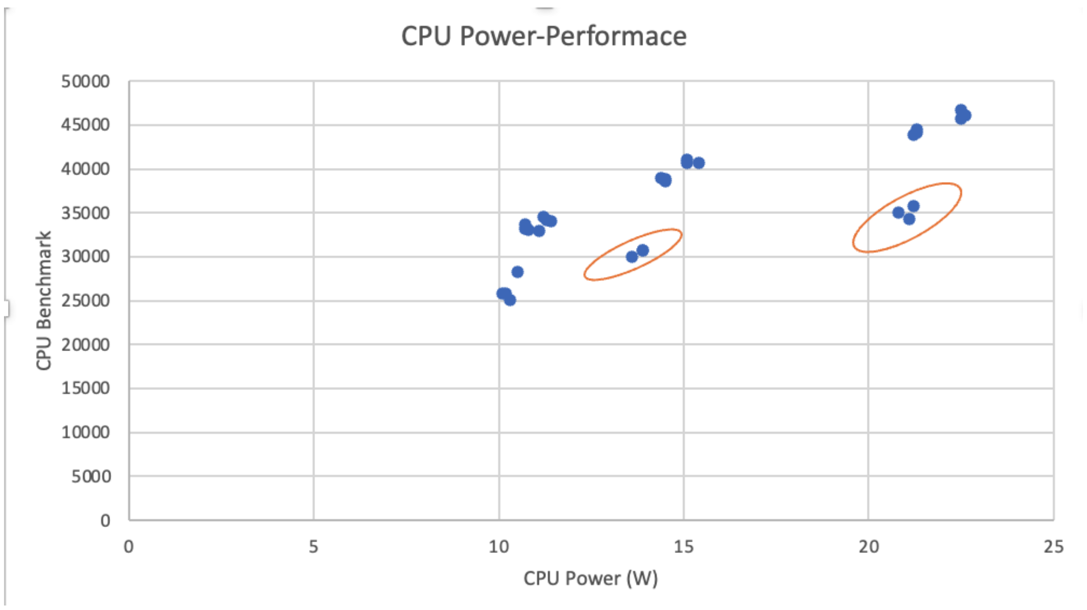


Figure 9 Power and Performance of CPU

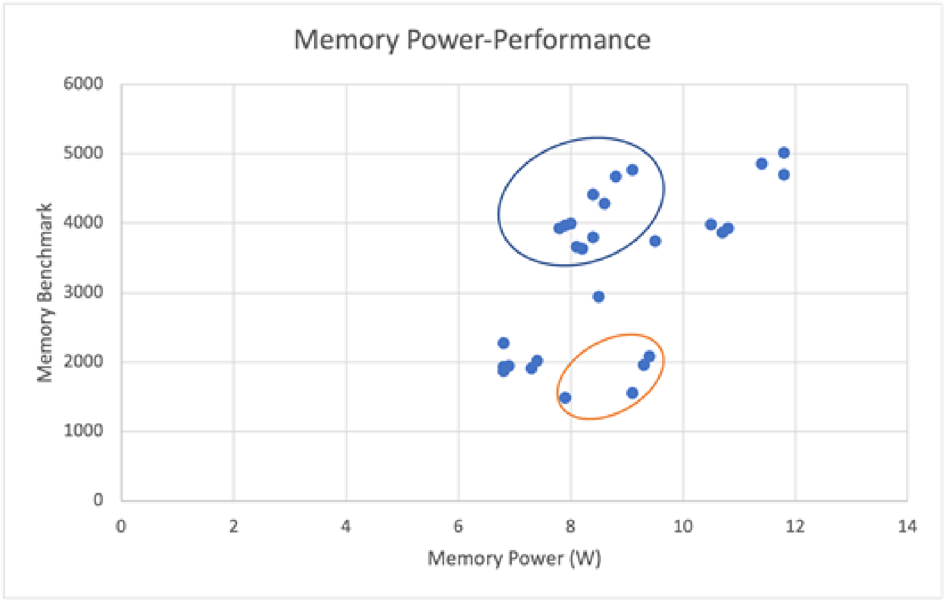


Figure 10: Power and Performance of Memory

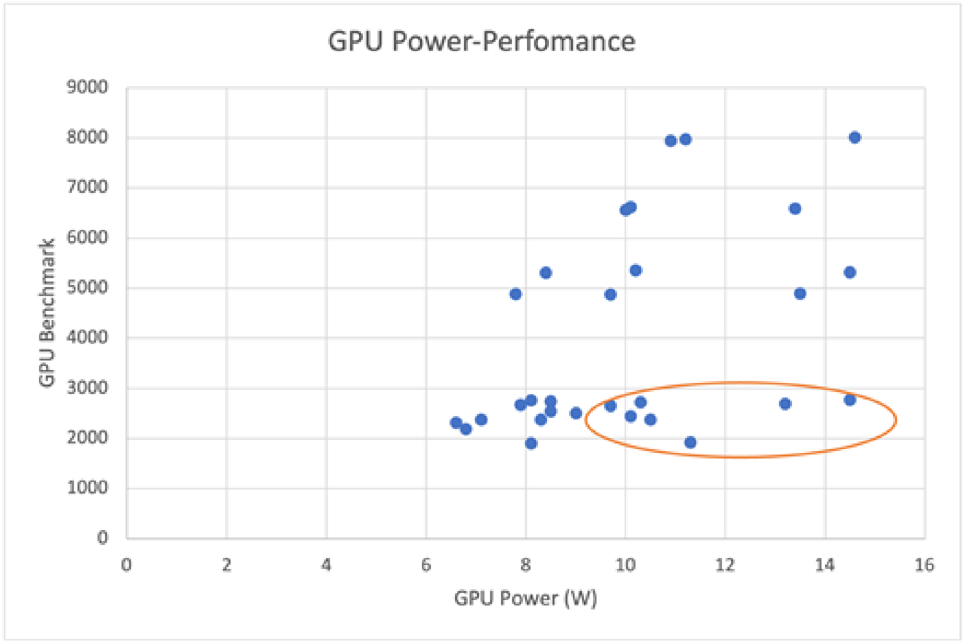


Figure 31: Power and Performance of GPU

To further understand this relationship, we calculated the Spearman Correlation of the benchmark performance for each resource with the operating point of all other resources. This analysis, in Table II, shows that memory frequency impacts performance of each component significantly, while CPU and GPU frequency impact only their own performance. These relationships are due to the architecture design; the CPU and GPU systems are both dependent on a unified memory which can cause stalls resulting in wasted cycles and loss of performance. Based on these numbers, we observe a general trend: to improve performance we first should increase memory frequency and then increase other frequencies. From the power perspective, the GPU provides the most gain for performance per power consumption across two extreme operating points: (1.65x) followed by memory (1.51x) and then CPU (0.8x).

Table 2: ANTUTU SPEARMAN CORRELATION

	MemPerf	CPUPerf	GPUPerf
MemFreq	0.921	0.632	0.776
CPUFreq	0.152	0.737	0.070
GPUFreq	-0.130	0.041	0.538

D. RODINA

The Rodinia benchmark suite [15] is a compute-intensive benchmark suite for heterogenous computing platforms. We chose Rodinia as it has a very strict division of labor between the CPU and the GPU, where the GPU performs all of the computation while the CPU performs file IO, memory allocation, etc. This was a desired characteristic to contrast to the Chai benchmark suite, which utilizes both the CPU and the GPU to collaborate in computation. Not all applications in Rodinia were able to run on the ODROID-XU4 platform, and thus, we used the following subset of applications: Pathfinder, KMeans, NW, Hotspot3D, Backprop, and Streamcluster.

D.1 FREQUENCY SWEEP

We swept the big CPU, little CPU, GPU, and memory frequencies on the ODROID-XU4 to understand the effect frequency scaling has on the selected Rodinia applications. The big CPU frequency was varied from 1.2 - 2.0 GHz, the little CPU frequency was varied from 1.0 - 1.4 GHz, the GPU frequency was varied from 177 - 600 MHz, and the memory frequency was varied from 633 - 933 MHz. For these sweeps, all applications were either CPU bound, Mixed CPU-GPU bound, or Mixed CPU-Mem bound. Three examples can be seen in Figures 12, 13, and 14, which show the sweeping results for the NW, Pathfinder, and StreamCluster applications, respectively. As one can see, the NW application is solely CPU bound as it observes a 1.31x performance improvement when scaling the CPU frequency, while only a 1.18x and 1.14x performance improvement when scaling the GPU and memory frequencies, respectively. Figure 13 shows StreamCluster, which we classify as Mixed CPU-Mem bound as we see both a 1.16x and 1.19x performance improvement when increasing the memory and CPU frequencies, respectively, while scaling GPU frequency only achieves a 1.05x performance improvement. Lastly, Figure 14 shows Pathfinder, which we classify as Mixed CPU-GPU bound as it sees a performance improvement of 1.84x when increasing the GPU frequency and a 1.29x performance improvement when increasing CPU frequency, while only achieving a 1.09x performance improvement when scaling memory frequency.

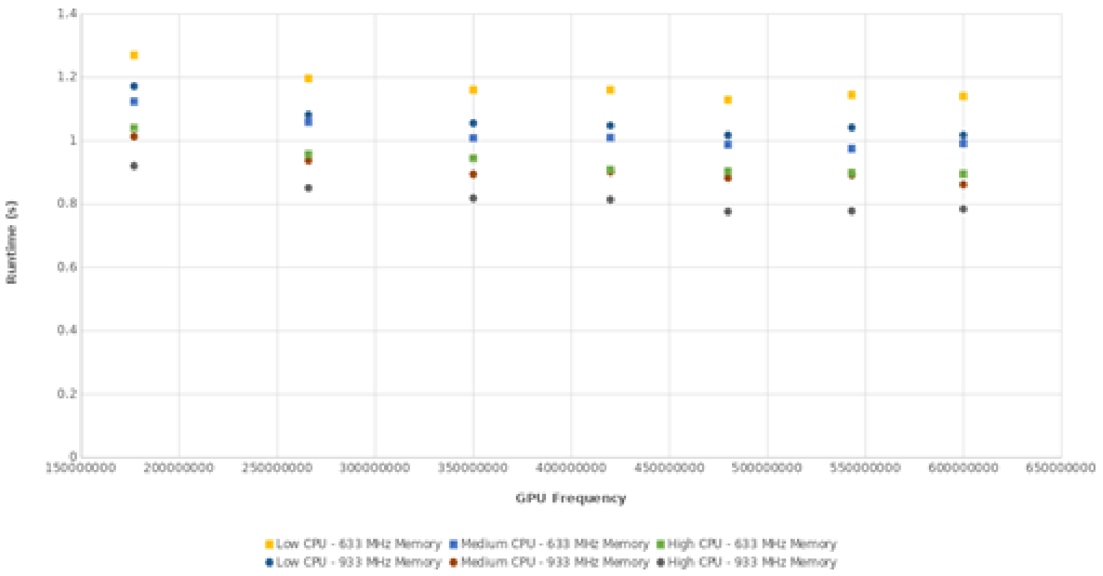


Figure 42: Frequency Sweep for NW

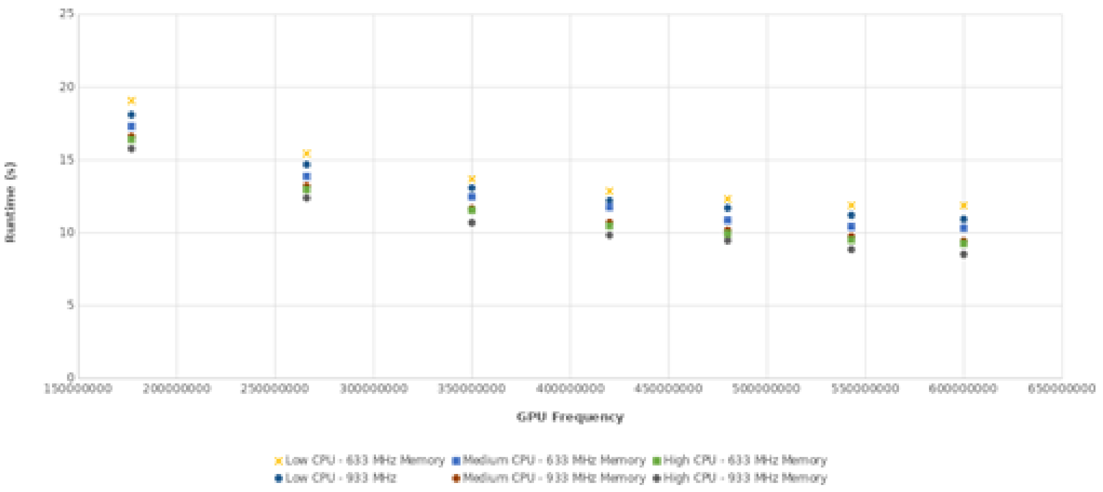


Figure 53: Frequency Sweep for StreamCluster

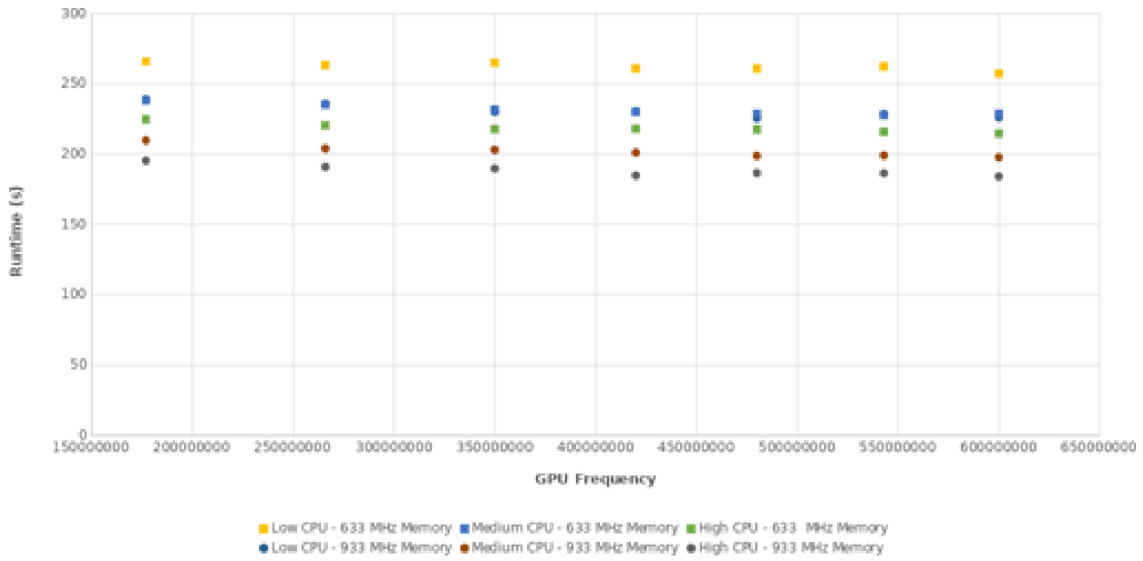


Figure 64: Frequency Sweep for Pathfinder

We have three interesting observations from this data. The first is that increasing CPU frequency always produces a non-trivial performance improvement. We hypothesize that this is because all applications are performing similar CPU operations such as memory allocation, file IO, etc. which clearly benefits from increasing CPU frequency. The second observation is that while we don't see a single application as being solely GPU bound or memory bound, we observe significant performance improvements when increasing either GPU or memory frequency for mixed bound applications. The third observation is that we never see the performance impact from memory frequency scaling to be the most significant, and it is always overshadowed by scaling either the CPU or the GPU frequency. We hypothesize that because the Rodinia benchmark suite offloads all computation to the GPU, these benchmarks are relatively insensitive to increases in DRAM access latency due to the GPU's ability to hide memory latencies behind massive parallelism.

D.2 CORRELATION ANALYSIS

To greater understand the performance impact of sweeping each component's frequency, we calculated the Spearman Correlation between the application latency and the frequency of each component. A negative Spearman Correlation between a frequency and the performance means that increasing the frequency of that components results in a decrease in the latency of the application. Additionally, the magnitude of the Spearman Correlation corresponds to the strength of the relationship between the frequency and latency. Table III shows the Spearman Correlation for each component for every Rodinia benchmark analyzed. Our first observation is that all component frequencies have a negative Spearman Correlation with the application latency for every application. This might seem obvious, but in the other benchmark suites there are scenarios where increasing a component's frequency actually degrades performance. We hypothesize this is due to the strict division of work in the Rodinia benchmarks, which is not seen in either Chai or AnTuTu. Our second observation is that the Spearman correlation perfectly aligns with our classification of CPU bound and mixed bound applications via the magnitude of the corresponding Spearman Correlation. The third observation we can make from this data is that the CPU frequency has an overall greater affect than the GPU and memory frequencies on application performance. This is very surprising as one would assume because these are all GPU workloads that tuning the GPU frequency would have the greatest effect. However, for all of these applications, the CPU must perform file IO to retrieve the input data, perform memory allocation, and perform transfers both to and from the GPU accessible address space, as well as interpret the results. Clearly, these operations benefit from increasing CPU frequency, and are consistent across all applications.

Table 3: RODINIA SPEARMAN CORRELATION

	GPU	CPU	Mem
Total	-0.36	-0.58	-0.2
KMeans	-0.70	-0.65	-0.055
NW	-0.33	-0.83	-0.37
Pathfinder	-0.83	-0.45	-0.15
StreamCluster	-0.18	-0.80	-0.53
HotSpot3D	-0.23	-0.91	-0.17
BackProp	-0.26	-0.85	-0.41

D.3 CLUSTERING ANALYSIS

One of our primary goals is to use this exploration to inform a DVFS scheme as to how frequency scaling impacts heterogenous SoCs. An issue that arises when designing a DVFS scheme for SoCs is the large design space as the number of SoC components increases. For example, our frequency sweep was over 63 unique permutations across four components which additionally introduce complexity on how each frequency impacts application performance. We hypothesize that while there are a large number of frequency permutations, we can find a few key frequency groups which have a similar impact on performance, regardless of application. On top of that, we hypothesize that we can cluster frequency groups together that lead to similar performance characteristics.

We utilize k-means clustering to cluster the performance of each application into four clusters. We then examine the frequency groups that contribute to each cluster, and search for frequency groups that are in the same cluster across all applications. Interestingly, we were able to find at least one frequency group that was in the same cluster for all applications. This shows that, despite applications being impacted by the different component frequencies, we can use these frequency groups to create an application-agnostic control over application performance and energy. These results can be seen in Table IV, where Cluster 0 is the lowest performing cluster and Cluster 3 is the highest performing cluster. As one can see, the most important frequency to tune is the CPU, as mentioned in our previous results. Another interesting observation is that there are many frequency groups which consistently are in the highest performing cluster. This means that we can select the frequency group in this cluster which consumes the lowest power, while still achieving similar performance. We hope to utilize a clustering-based statistical analysis similar to this to develop a DVFS scheme in the Linux governor to control the big CPU, the little CPU, the GPU, and the memory frequencies in tandem.

Table 4: RODINA GENERATED FREQUENCY GROUPS

	GPU	CPU	Mem
Cluster 0	177 Mhz	1.2 Ghz	633 Mhz
Cluster 1	266 Mhz	1.2 Ghz	933 Mhz
	350 Mhz	1.2 Ghz	933 Mhz
Cluster 2	350 Mhz	1.6 Ghz	825 Mhz
Cluster 3	420 Mhz	2.0 Ghz	825 Mhz
	480 Mhz	2.0 Ghz	933 Mhz
	543 Mhz	2.0 Ghz	933 Mhz
	600 Mhz	2.0 Ghz	933 Mhz
	420 Mhz	2.0 Ghz	933 Mhz
	480 Mhz	2.0 Ghz	933 Mhz
	543 Mhz	2.0 Ghz	933 Mhz
	600 Mhz	2.0 Ghz	933 Mhz

E. CHAI

Unlike other benchmark suites, the Chai benchmark suite [17] contains collaborative benchmarks with kernels that execute on both the CPU and the GPU simultaneously. This allows us to explore the impact of frequency scaling across the SoC when the entire SoC is under load. Unlike the Rodinia benchmarks, we expect to see a variety of performance trends across benchmarks. We ran as many of the Chai benchmarks as would compile and execute on our evaluation platform. Our selected subset for analysis include: Breadth-First Search (BFS), Image Histogram (Input Partitioning) (HSTI), Image Histogram (Output partitioning) (HSTO), In-place

Transposition (TRNS), and Single-Source Shortest Path (SSSP). These applications collaborate through task partitioning (SSSP and BFS) and data partitioning (TRNS, HSTI, and HSTO).

E.1 GPU FREQUENCY SWEEP

To form a basic understanding of application behavior as hardware frequencies are changed, we perform a sweep of both the GPU and the CPU frequencies for each of the benchmarks. We leverage the ability to change the CPU and the GPU frequencies on the fly in Linux to manually tweak frequencies. We then run each benchmark and report timing and performance data. To smooth out the resulting data, we run each benchmark under each setting six times and report the average of each performance metric. We sweep across all possible values of GPU frequency (177 - 600 MHz) and evaluate at three different CPU frequencies — low, medium, and high. The low CPU frequency sets the big and little CPU cores at 1.2 GHz and 1 GHz, medium at 1.6 GHz and 1.2 GHz, and high at 2.0 GHz and 1.4 GHz, respectively. Looking across benchmarks, we notice three trends of behavior as we scale frequency parameters:

1) CPU-bound: The CPU-bound benchmarks (BFS and HSTI) demonstrate little performance improvement from increasing GPU frequency but do demonstrate performance improvement from increasing CPU frequency. As seen in Figure 15, there is no point where increasing GPU frequency provides a better performance improvement than that gained from increasing CPU frequency. Moreover, clocking the GPU at a higher frequency has little impact on performance and decreases system efficiency by consuming additional power. When running these benchmarks, a well-performing governor should prioritize CPU frequency increases and GPU frequency decreases to optimize performance.

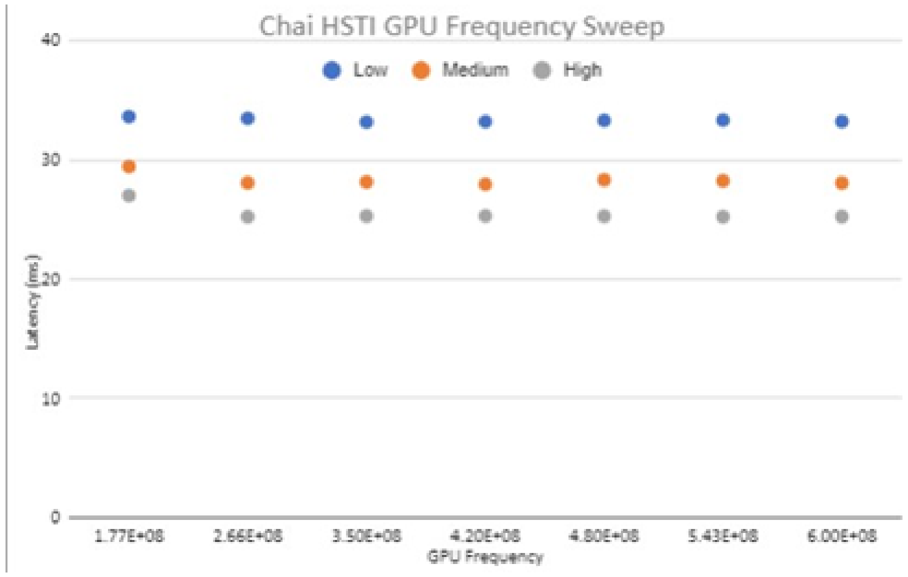


Figure 75: GPU Frequency Sweep for HSTI

2) GPU-bound: A foil to the CPU-bound benchmarks, GPU-bound benchmarks (TRNS) demonstrate performance improvement from increasing GPU frequency and little performance improvement from increasing CPU frequency. As seen in Figure 16, there is no point where increasing CPU frequency provides a meaningful increase to performance. A well-performing governor, should therefore prioritize increasing GPU frequency and decreasing CPU frequency to perform optimally. We also note that even for this GPU-bound benchmark, there is performance saturation as we further increase GPU frequency, suggesting that an optimal operating point may not be running the GPU at max settings.

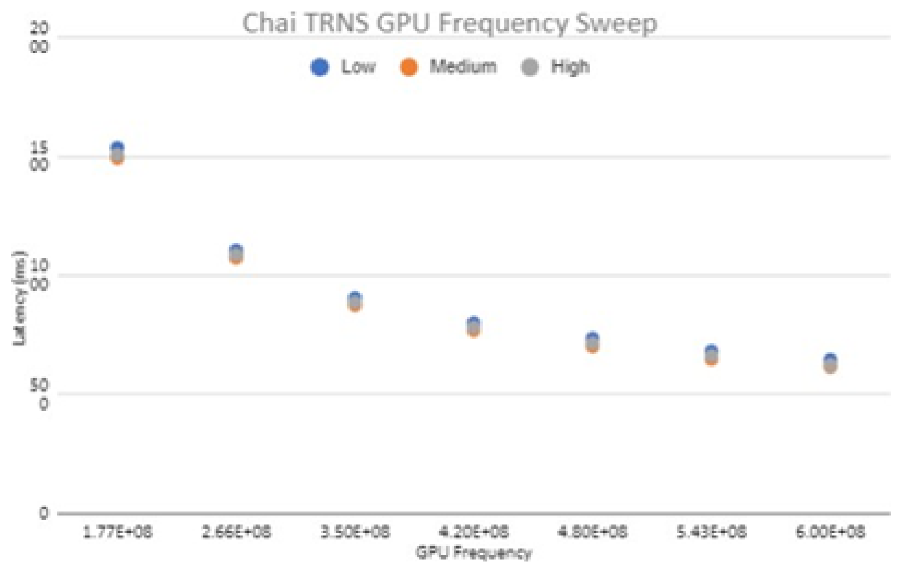


Figure 86: CPU Frequency Sweep for TRNS

3) Mixed-bound: Finally, the mixed-bound applications (SSSP and HSTO) demonstrate behavior that changes depending on the current operating point. As seen in Figure 17, these benchmarks exhibit the greatest performance improvement from first increasing GPU frequency and then increasing CPU frequency. There is performance saturation once CPU and GPU frequencies reach a certain threshold, suggesting that running at max settings provides the same performance at a power loss.

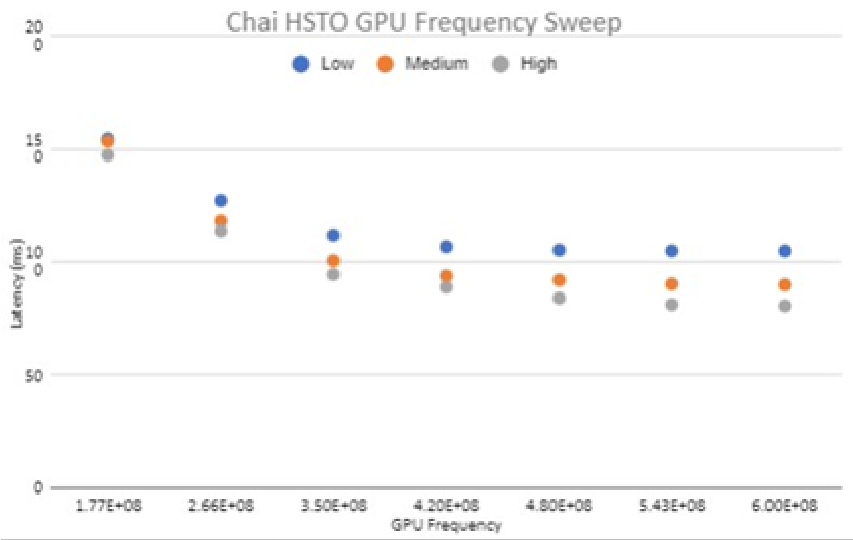


Figure 97: GPU Frequency Sweep for HSTO

E.2 MEMORY FREQUENCY SWEEP

In addition to sweeping across CPU and GPU frequencies, we leveraged the SoC's ability to over- and under-clock the memory frequency to explore the impact of memory frequency on performance at a variety of operating points. For most applications, changing memory frequency monotonically increases performance. There are points at a particular energy or power budget where increasing memory frequency will provide a larger performance increase than increasing GPU or CPU frequency, but a larger performance gain can be had by increasing CPU or GPU frequency if power is not a concern. Other applications demonstrate interesting behavior as memory frequencies are swept. For instance, HSTO demonstrates operating points where running the CPU and the GPU at lower frequencies but increasing the memory frequency provides a higher performance than increasing the CPU and the GPU frequencies as seen in Figure 18. Notice that at 825 MHz memory frequency, running with a medium CPU frequency and low GPU frequency outperforms running with a high CPU frequency and low GPU frequency. As memory frequency increases, the high CPU frequency operating point outperforms the medium CPU frequency operating point. Unlike with the GPU and the CPU frequencies, there is no clear-cut classification for application behaviors across memory frequencies.

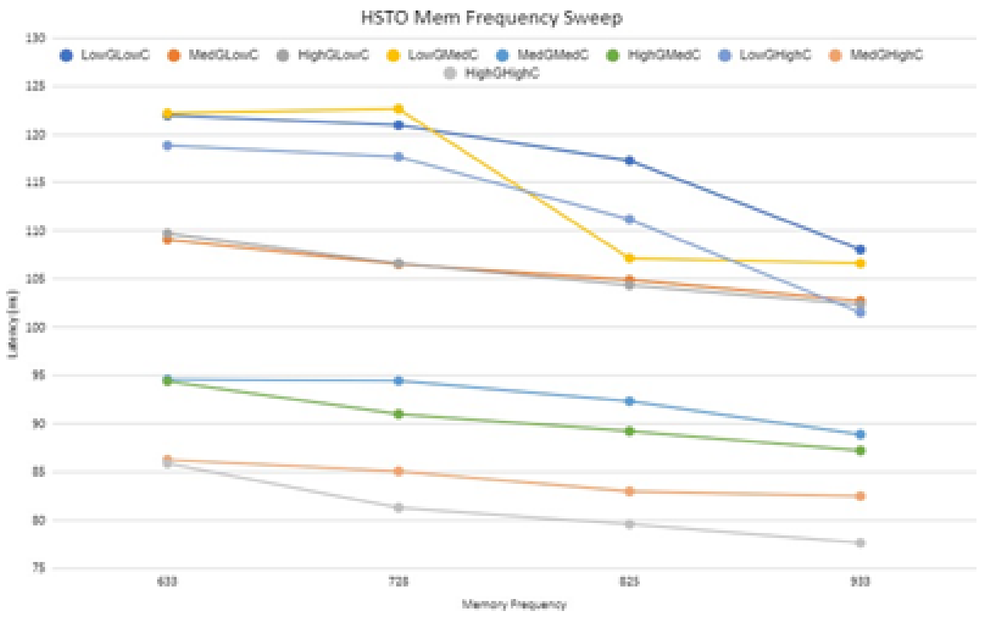


Figure 108: Memory Frequency Sweep for HSTO

E.3 CLUSTERING

Throughout this exploration, our goal is to find operating points of equivalent performance to cluster into a single operating point. Clustering these equivalent operating points enables the governor to select the most efficient state at the desired performance level. Transitions between clusters can then be reduced from the entire state space to a constrained state space. State changes can then demonstrate DVFS changes across the entire SoC rather than a single component. We begin this clustering process by examining equivalent clusters for each application at a time. To derive equivalent performance clusters, we employ KNN clustering on the performance data. To select the number of clusters, we employ the elbow method and identify that four clusters is optimal as shown in Figure 19. Higher cluster identification number indicates higher cluster performance.

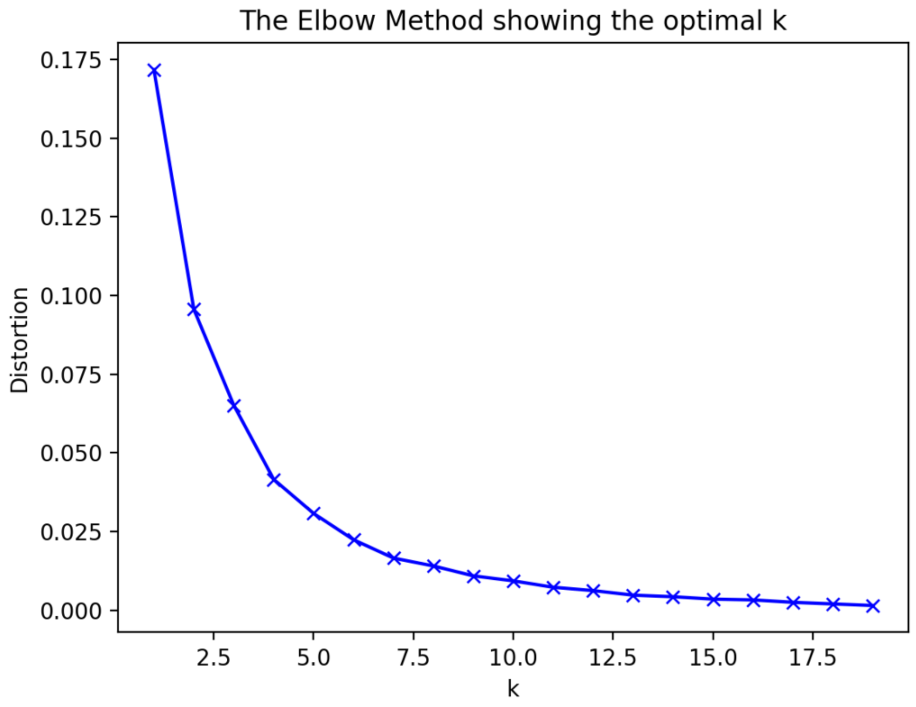


Figure 119: Plot Showing Computation of Optimal k Using the Elbow Method

While this clustering approach is well suited for identifying performance points for each individual application, it is ill-suited at deriving a generalized set of operating points. In an effort to generalize this analysis to a broader class of applications, we explore cross-application trends. To accomplish this, we identify operating points that consistently appear in performance classes and tie those together. Unsurprisingly, there are few consistent operating points within clusters due to the heterogeneous behavior when compared to benchmarks in Rodinia as seen in Table V. If we increase our tolerance to deviation between groups to allow for up to one application to disagree with the cluster, we see a more versatile set of clusters as shown in Table VI. Although certain trends exist across applications, there is no overarching set of clusters that apply well to all applications. Therefore, designing a consistently well-performing governor is challenging when running a diverse set of applications. Rather than design a single, all-encompassing governor, a better approach would involve designing a "smart-governor" that changes based on the current workload. Exploration of this concept is left for future work. Moreover, our evaluation fails to account for power usage at each operating point when executing Chai. Without these power numbers, our analysis only captures one view of the overall system.

Table 5: CHAI GENERATED FREQUENCY GROUPS

	GPU	CPU	Mem
Cluster 0	177 Mhz 177 Mhz	1.2 Ghz 1.2Ghz	633 Mhz 728 Mhz
Cluster 1			
Cluster 2			
Cluster 3	600 Mhz 600 Mhz	2.0 Ghz 2.0 Ghz	933 Mhz 825 Mhz

Table 6: CHAI GENERATED FREQUENCY GROUPS

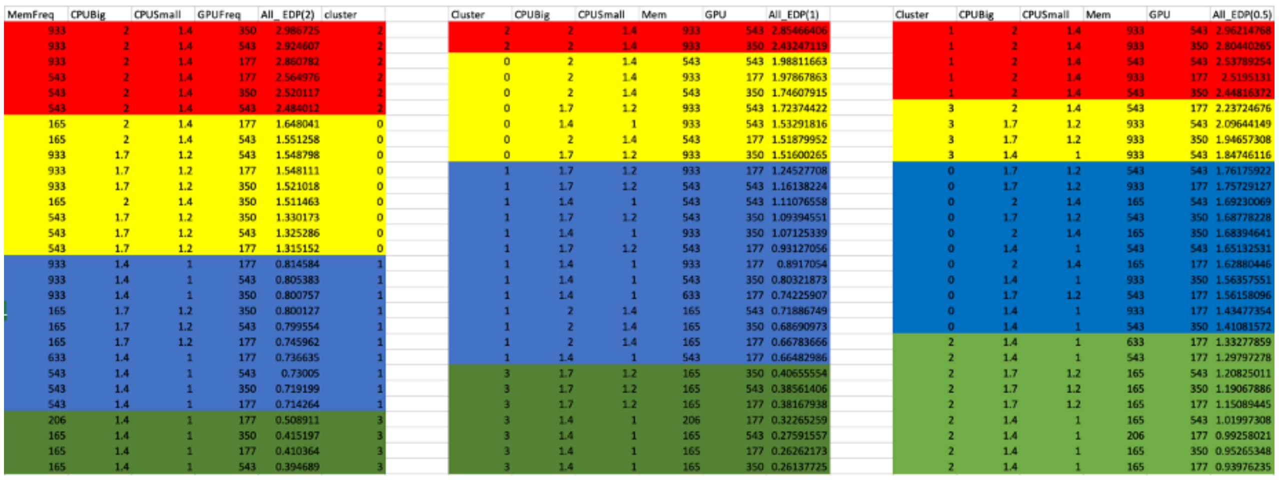
	GPU	CPU	Mem
Cluster 0	177 Mhz 177 Mhz 177 Mhz 177 Mhz	1.2 Ghz 1.2Ghz 1.6Ghz 1.2Ghz	633 Mhz 825 Mhz 633 Mhz 728 Mhz
Cluster 1	177 Mhz 177Mhz	1.6Ghz 1.6Ghz	825Mhz 933Mhz
Cluster 2	420 Mhz 420 Mhz	2.0 Ghz 1.6 Ghz	633 Mhz 825 Mhz
Cluster 3	600 Mhz 600 Mhz 600 Mhz 420 Mhz 420 Mhz 600 Mhz	2.0 Ghz 2.0 Ghz 1.6 Ghz 2.0 Ghz 2.0 Ghz 2.0 Ghz	933 Mhz 728 Mhz 933 Mhz 933 Mhz 825 Mhz 825 Mhz

F. MULTI-COMPONENT DVFS POLICY COMPARISON

Given our analysis, we utilize power and performance data across all benchmarks to construct frequency clusters which provide similar power and performance characteristics. We create the frequency clusters by using k-means clustering on an energy-delay product ($EDP(v)$) metric:

$$EDP(v) = \sum_{r \in Resources} power_r \times performance_r^v$$

where $power_r$ represents the power consumed by resource r, and $performance_r$ is the corresponding performance value (which could be benchmark score or completion time). The parameter v allows us to define system constraints. A value $v < 2$ shows that power is a more important metric than performance, while $v > 2$ represents a more performance focused system.



MemFreq	CPUBig	CPUSmall	GPUFreq	All_ EDP(2)	cluster
933	2	1.4	350	2.986725	2
933	2	1.4	543	2.924607	2
933	2	1.4	177	2.860782	2
543	2	1.4	177	2.564976	2
543	2	1.4	350	2.520117	2
543	2	1.4	543	2.484012	2
165	2	1.4	177	1.648041	0
165	2	1.4	543	1.551258	0
933	1.7	1.2	543	1.548798	0
933	1.7	1.2	177	1.548111	0
933	1.7	1.2	350	1.521018	0
165	2	1.4	350	1.511463	0
543	1.7	1.2	350	1.330173	0
543	1.7	1.2	543	1.325286	0
543	1.7	1.2	177	1.315152	0
933	1.4	1	177	0.814584	1
933	1.4	1	543	0.805383	1
933	1.4	1	350	0.800757	1
165	1.7	1.2	350	0.800127	1
165	1.7	1.2	543	0.799554	1
165	1.7	1.2	177	0.745962	1
633	1.4	1	177	0.736635	1
543	1.4	1	543	0.73005	1
543	1.4	1	350	0.719199	1
543	1.4	1	177	0.714264	1
206	1.4	1	177	0.508911	3
165	1.4	1	350	0.415197	3
165	1.4	1	177	0.410364	3
165	1.4	1	543	0.394689	3

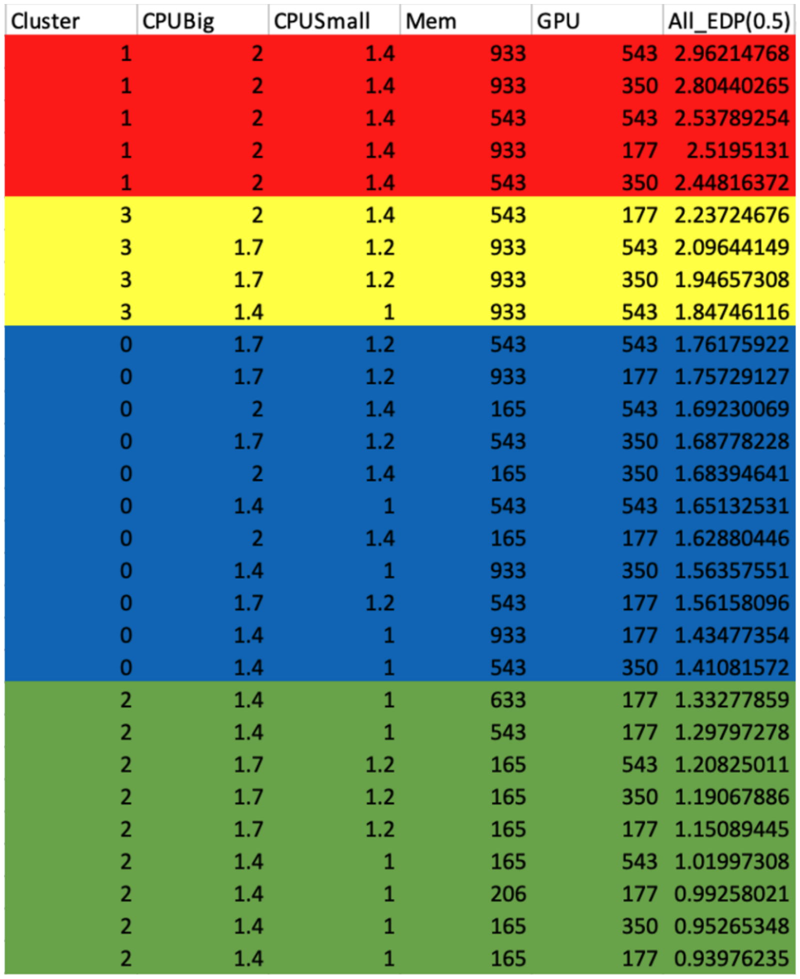
Cluster	CPUBig	CPUSmall	Mem	GPU	All_EDP(1)
2	2	1.4	933	543	2.85466406
2	2	1.4	933	350	2.43247119
0	2	1.4	543	543	1.98811663
0	2	1.4	933	177	1.97867863
0	2	1.4	543	350	1.74607915
0	1.7	1.2	933	543	1.72374422
0	1.4	1	933	543	1.53291816
0	2	1.4	543	177	1.51879952
0	1.7	1.2	933	350	1.51600265
1	1.7	1.2	933	177	1.24527708
1	1.7	1.2	543	543	1.16138224
1	1.4	1	543	543	1.11076558
1	1.7	1.2	543	350	1.09394551
1	1.4	1	933	350	1.07125339
1	1.7	1.2	543	177	0.93127056
1	1.4	1	933	177	0.8917054
1	1.4	1	543	350	0.80321873
1	1.4	1	633	177	0.74225907
1	2	1.4	165	543	0.71886749
1	2	1.4	165	350	0.68690973
1	2	1.4	165	177	0.66783666
1	1.4	1	543	177	0.66482986
3	1.7	1.2	165	350	0.40655554
3	1.7	1.2	165	543	0.38561406
3	1.7	1.2	165	177	0.38167938
3	1.4	1	206	177	0.32265259
3	1.4	1	165	543	0.27591557
3	1.4	1	165	177	0.26262173
3	1.4	1	165	350	0.26137725

Cluster	CPUBig	CPUSmall	Mem	GPU	All_EDP(0.5)
1	2	1.4	933	543	2.96214768
1	2	1.4	933	350	2.80440265
1	2	1.4	543	543	2.53789254
1	2	1.4	933	177	2.5195131
1	2	1.4	543	350	2.44816372
3	2	1.4	543	177	2.23724676
3	1.7	1.2	933	543	2.09644149
3	1.7	1.2	933	350	1.94657308
3	1.4	1	933	543	1.84746116
0	1.7	1.2	543	543	1.76175922
0	1.7	1.2	933	177	1.75729127
0	2	1.4	165	543	1.69230069
0	1.7	1.2	543	350	1.68778228
0	2	1.4	165	350	1.68394641
0	1.4	1	543	543	1.65132531
0	2	1.4	165	177	1.62880446
0	1.4	1	933	350	1.56357551
0	1.7	1.2	543	177	1.56158096
0	1.4	1	933	177	1.43477354
0	1.4	1	543	350	1.41081572
2	1.4	1	633	177	1.33277859
2	1.4	1	543	177	1.29797278
2	1.7	1.2	165	543	1.20825011
2	1.7	1.2	165	350	1.19067886
2	1.7	1.2	165	177	1.15089445
2	1.4	1	165	543	1.01997308
2	1.4	1	206	177	0.99258021
2	1.4	1	165	350	0.95265348
2	1.4	1	165	177	0.93976235

Figure 20: Different Clusters for $v = 2, 1, 0.5$

Figure 20 shows that cluster composition changes based on the v value. Each color represents a set of frequencies exhibiting a similar power-performance trade-off. The clusters in the top (red and yellow) represent performance-focused configuration, while those in the bottom (blue and green) represent more power-focused.

For higher values of v (≥ 2), we observe an increasing relationship among all the component frequencies. This pattern is consistent with an increasing relationship between performance and frequency. However, we observe complex configurations when we focus on power while trying to retain performance (lower v (≤ 1) values). In Figure 21, the blue cluster shows that lower CPU frequencies correspond to medium and high memory frequency, while higher CPU frequencies map to low memory and GPU frequencies. This setup allows the system to retain performance of a single component while reducing power consumption of other resources.



Cluster	CPUBig	CPUSmall	Mem	GPU	All_EDP(0.5)
1	2	1.4	933	543	2.96214768
1	2	1.4	933	350	2.80440265
1	2	1.4	543	543	2.53789254
1	2	1.4	933	177	2.5195131
1	2	1.4	543	350	2.44816372
3	2	1.4	543	177	2.23724676
3	1.7	1.2	933	543	2.09644149
3	1.7	1.2	933	350	1.94657308
3	1.4	1	933	543	1.84746116
0	1.7	1.2	543	543	1.76175922
0	1.7	1.2	933	177	1.75729127
0	2	1.4	165	543	1.69230069
0	1.7	1.2	543	350	1.68778228
0	2	1.4	165	350	1.68394641
0	1.4	1	543	543	1.65132531
0	2	1.4	165	177	1.62880446
0	1.4	1	933	350	1.56357551
0	1.7	1.2	543	177	1.56158096
0	1.4	1	933	177	1.43477354
0	1.4	1	543	350	1.41081572
2	1.4	1	633	177	1.33277859
2	1.4	1	543	177	1.29797278
2	1.7	1.2	165	543	1.20825011
2	1.7	1.2	165	350	1.19067886
2	1.7	1.2	165	177	1.15089445
2	1.4	1	165	543	1.01997308
2	1.4	1	206	177	0.99258021
2	1.4	1	165	350	0.95265348
2	1.4	1	165	177	0.93976235

Figure 21: Frequency Clusters at $EDP(v = 0.5)$

Typically, the default DVFS policies analyze the utilization of each resource independently and sets its frequency based on the defined policies. Here, we consider two policies: a performance policy, which sets all the resources to their maximum frequency to extract the maximum possible performance without considering the power requirement; and a more dynamic policy, on-demand, which sets the frequency based on current utilization. This method allows us to reduce power consumption during idle periods, while meeting the performance requirements.

For different system requirements, such as power or performance we input the different frequency clusters shown in Figure 20 into our policy. As we observe in Figures 22 and 23, different frequency clusters result in either significant power savings (20%) from the most conservative system (policy_Pow), or minimal performance degradation (3%) while reducing power consumption by 45% in policy_Perf.

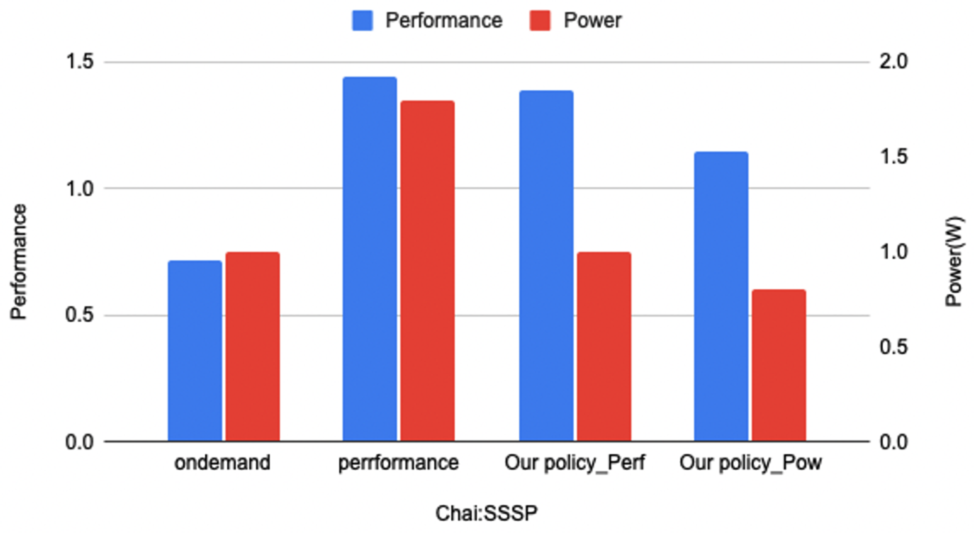


Figure 22: Comparing the Power and Performance of Our Policy with Existing Methods for SSSP

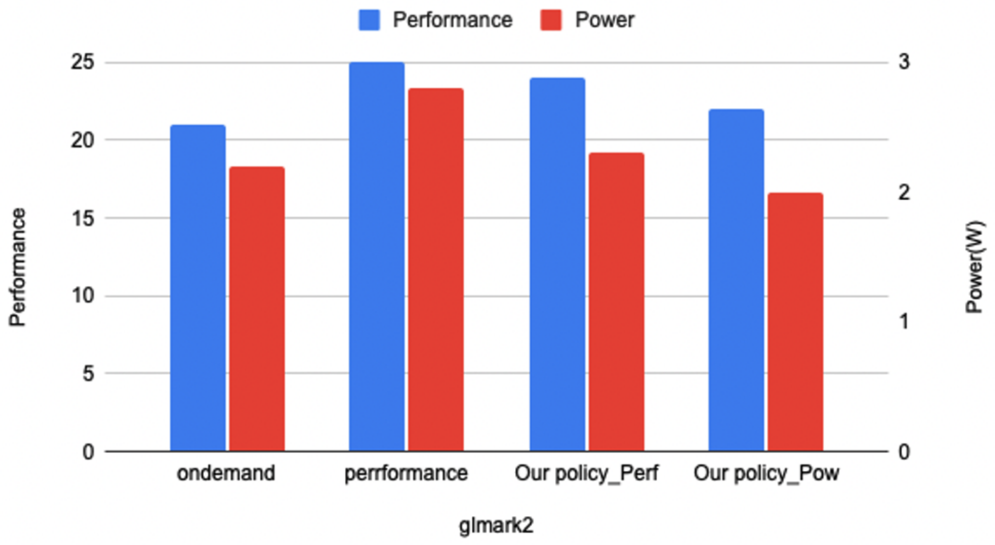


Figure 23: Comparing the Power and Performance of Our Policy with Existing Methods for glmark2

ANTICIPATED OUTCOMES AND IMPACTS

We have successfully met our research goal of achieving 10% or more power savings. Depending on the policy we ran, we saw savings as high as 45% even with added computing overhead of 6-11% from running our scripts. Since a 10% energy savings in ICT devices would significantly reduce their effect on the global climate footprint, we feel that our method has the potential to have an even greater impact on the climate than originally thought. Energy suppliers would appreciate this reduction as it would help add resilience to power grid outages through improved operation of mobile devices, satellites, and Internet of Things devices. In addition, our method could be used to increase the performance of systems while keeping their power consumption constant. In fact, our method could benefit Sandia's Global Security and National Security Program by reducing energy consumption providing resilience to mobile sensing and military devices. Through its application to decision support and remote sensing systems, it could improve the performance of these systems possibly leading to more time to make critical decisions. We have also been looking at ways to integrate our method into already existing APIs that focus on power usage of systems giving users another avenue to modify system configuration to gain power savings.

Our research success will allow system wide optimal energy policies to be created that reduce the energy consumption of systems. This would have the impact of creating a more environmentally responsible energy footprint addressing S02 of the DOE's science and energy goal. Additionally, it would have some application to NNSA's crosscut 5 by providing a means of creating modern energy conscious IT systems. We plan to publish our results in a journal like the IEEE Transactions on Parallel and Distributed Computing and also plan to submit a full LDRD to further investigate a temporal analysis of application resource requirements for efficient scheduling and to develop a finer-grained DVFS policy for heterogeneous systems.

CONCLUSION

In this project, we studied the impact of operating frequencies of various components of SoCs on the performance of various benchmarks. To analyze impact, we ran variety of benchmarks which focused on either each component individually (being CPU, GPU, or memory-bound) or a more general workload which utilized all components. These benchmarks allowed us to break down the relationship among each component, while also providing an overview of the system-wide impact. We proposed a dependency graph which can provide us with clusters of optimal performance, while ensuring minimal wasted power. Across various benchmarks we do observe certain clusters (as shown in Table IV and VI) where lower and higher frequencies of each components map together, respectively, while observing a similar impact of power on performance. We can use our methods to design more energy aware clusters which account for power-performance trade-off. We also observed the disparity in the power and performance across various components, where changes in the GPU configuration provide more gains in power followed by memory and the CPU. These observations provide a foundational basis for designing system-wide DVFS policies, which can be used to make some application agnostic decisions for an energy-aware system. Future research may utilize these studies to evaluate a wider variety of workloads to verify our observation and design a DVFS governor which can decide an optimal operating state and devise an efficient transition strategy which ensures minimal energy waste.

REFERENCES

[1] P. Czarnul, J. Proficz, and A. Krzywaniak, "Energy-aware high- performance computing: Survey of state-of-the-art tools, techniques, and environments," Scientific Programming, vol. 2019, pp. 1–19, 2019.
[2] R. F. Barrett, S. Borkar, S. S. Dosanjh, S. D. Hammond, M. A. Heroux, X. S. Hu, J. Luitjens, S. G. Parker, J. Shalf, and L. Tang, "On the role of co-design in high performance computing," in Transition of HPC Towards Exascale Computing - Selected Papers from the High Performance Computing Workshop, Cetraro, Italy, June 25-29, 2012, ser. Advances in Parallel Computing, E. H. D'Hollander,
J. J. Dongarra, I. T. Foster, L. Grandinetti, and G. R. Joubert, Eds., vol. 24. IOS Press, 2012, pp. 141–155. [Online]. Available: https://doi.org/10.3233/978-1-61499-324-7-141
[3] S. Bhalachandra, A. Porterfield, S. L. Olivier, and J. Prins, "An adaptive core-specific runtime for energy efficiency," 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 947–956, 2017.
[4] L. Tan, S. L. Song, P. Wu, Z. Chen, R. Ge, and D. J. Kerbyson, "Investigating the interplay between energy efficiency and resilience in high performance computing," in 2015 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2015, Hyderabad, India, May 25-29, 2015. IEEE Computer Society, 2015, pp. 786–796. [Online]. Available: https://doi.org/10.1109/IPDPS.2015.108
[5] G. von Laszewski, L. Wang, A. J. Younge, and
X. He, "Power-aware scheduling of virtual machines in dvfs-enabled clusters." in CLUSTER. IEEE Computer So- ciety, 2009, pp. 1–10. [Online]. Available: http://dblp.uni-trier.de/db/conf/cluster/cluster2009.htmlLaszewskiWYH09
[6] V. A. Korthikanti and G. Agha, "Analysis of parallel algorithms for energy conservation in scalable multicore architectures," in ICPP 2009, International Conference on Parallel Processing, Vienna, Austria, 22-25 September 2009. IEEE Computer Society, 2009, pp. 212–219.
[7] G. D. Costa and J.-M. Pierson, "Dvfs governor for hpc: Higher, faster, greener," in 2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, 2015, pp. 533–540.
[8] P. Czarnul and P. Ros´ciszewski, "Optimization of execution time under power consumption constraints in a heterogeneous parallel system with gpus and cpus," in Distributed Computing

and Networking, M. Chatter- jee, J.-n. Cao, K. Kothapalli, and S. Rajsbaum, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 66–80.
[9] J. Haj-Yahya, M. Alser, J. Kim, A. G. Yag˘lıkc¸ı, N. Vijaykumar,
E. Rotem, and O. Mutlu, “Sysscale: Exploiting multi-domain dynamic voltage and frequency scaling for energy efficient mobile processors,” in Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture, ser. ISCA ’20. IEEE Press, 2020, p. 227–240. [Online]. Available: https://doi.org/10.1109/ISCA45697.2020.00029
[10] R. Nath and D. Tullsen, “The crisp performance model for dynamic voltage and frequency scaling in a gpgpu,” in Proceedings of the 48th International Symposium on Microarchitecture, ser. MICRO-48. New York, NY, USA: Association for Computing Machinery, 2015, p. 281–293. [Online]. Available: https://doi.org/10.1145/2830772.2830826
[11] Q. Deng, D. Meisner, L. Ramos, T. F. Wenisch, and R. Bianchini, “Memscale: Active low-power modes for main memory,” in Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems, ser. ASPLOS XVI. New York, NY, USA: Association for Computing Machinery, 2011, p. 225–238. [Online]. Available: https://doi.org/10.1145/1950365.1950392
[12] K. K. Chang, A. G. Yag˘lıkc¸ı, S. Ghose, A. Agrawal, N. Chatterjee,
A. Kashyap, D. Lee, M. O’Connor, H. Hassan, and O. Mutlu, “Understanding reduced-voltage operation in modern dram devices: Experimental characterization, analysis, and mechanisms,” Proc. ACM Meas. Anal. Comput. Syst., vol. 1, no. 1, jun 2017. [Online]. Available: https://doi.org/10.1145/3084447
[13] S. Lee, K.-D. Kang, H. Lee, H. Park, Y. Son, N. S. Kim,
and D. Kim, “Greendimm: Os-assisted dram power management for dram with a sub-array granularity power-down state,” in MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture, ser. MICRO ’21. New York, NY, USA: Association for Computing Machinery, 2021, p. 131–142. [Online]. Available: https://doi.org/10.1145/3466752.3480089
[14] Q. Deng, D. Meisner, A. Bhattacharjee, T. F. Wenisch, and
R. Bianchini, “Coscale: Coordinating cpu and memory system dvfs in server systems,” in Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture, ser. MICRO-45. USA: IEEE Computer Society, 2012, p. 143–154. [Online]. Available: https://doi.org/10.1109/MICRO.2012.22
[15] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and
K. Skadron, “Rodinia: A benchmark suite for heterogeneous computing,” in 2009 IEEE International Symposium on Workload Characterization (IISWC), 2009, pp. 44–54.
[16] S. Che, J. W. Sheaffer, M. Boyer, L. G. Szafaryn, L. Wang, and
K. Skadron, “A characterization of the rodinia benchmark suite with comparison to contemporary cmp workloads,” in IEEE International Symposium on Workload Characterization (IISWC’10), 2010, pp. 1–11.

[17] J. Go´mez-Luna, I. E. Hajj, L.-W. Chang, V. Garc´ıa-Floreszx, S. G. de Gonzalo, T. B. Jablin, A. J. Pen˜a, and W.-m. Hwu, "Chai: Collabo- rative heterogeneous applications for integrated-architectures," in 2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2017, pp. 43–54.

LABORATORY DIRECTED RESEARCH & DEVELOPMENT

WHERE INNOVATION BEGINS

ADDENDUM:

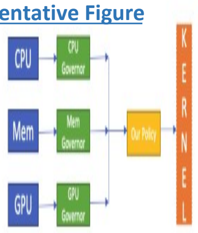
Runtime Systems for Energy Efficiency in Advanced Computing Systems, 227331

Curtis Madsen (6323), Tian Ma (6321), Professor Gul Agha (UIUC)

Purpose, Approach, and Goal

Motivation: Energy consumed by computing systems has grown over the last decade. Current dynamic voltage and frequency scaling (DVFS) methods are designed to work on individual components and are unaware of other resource requirements and power demands of the complete system.
Hypothesis: Parallelism in applications can be leveraged to develop latency aware fine-grained DVFS for heterogeneous systems reducing energy consumption
R&D Approach: Create state transition models which combine static information about application behavior with dynamic information about the operating environment. Estimate energy use by state transition time and learn the behavior of applications to predict transitions and optimize runtime energy efficiency.
One Key Goal: Achieve energy savings of at least 10% over current DVFS methods.

Representative Figure



We developed a system-wide DVFS policy that sits between the governors for each component and the kernel and modifies the frequencies of each component to obtain energy savings and better performance for the system.

Key R&D Results and Significance

Summarize your R&D
Build dependency graph among all the components in the system:

- Run benchmarks and collect utilization numbers for each component.
- Characterize the interactions among different components and determine how their operating points impact performance, utilization, and energy.
- Use this information to construct the dependency graph and cluster into equivalence classes.

Use dependencies to design a system-wide governor to improve energy efficiency and quality of service of the system.

- Each new policy is driven by one of the components (CPU, GPU, Memory) and the other components' frequencies are modified based on the clustering.

The result for the one key goal
Depending on the policy we ran, we saw **savings as high as 35%** even with an added computing overhead of 6-11% from running our scripts.

Lessons learned
Using Python for scripting adds overhead and is not suitable for energy-efficient applications. Could use a different prototyping language like Bash.
Attempted to implement governors for non-standard resources which is very difficult without source code. Should have started using scripts earlier.

Follow-on plans/activities
Clean up scripts to reduce overhead and make more generalized for multiple systems.
Utilize approach to increase system performance while maintaining a specified power usage.
Obtain funding from a CIS LDRD or the DOE Office of Science ASCR to target more mission applications.

Impact of follow-on plans
This work benefits Sandia's Global Security and National Security Program by reducing energy consumption providing resilience to remote sensors, mobile devices, and military devices.

Publications, awards, staff development & IP
IEEE Transactions on Parallel and Distributed Computing (in preparation)
Supported an early career employee, Curtis Madsen, who was also the PI.

R&D Summary (Methods, Results and Discussions)

To save energy, Dynamic Voltage and Frequency Scheduling (DVFS) is implemented on many current devices:

- DVFS is a power management technique used to adjust the voltage and frequency of a computing device's processor to conserve energy when the component is idle
- Modern DVFS can be run at different operating points on a per-component basis[1]
- However, transitions are performed instantaneously without consideration for other components[2], which can lead to loss of energy

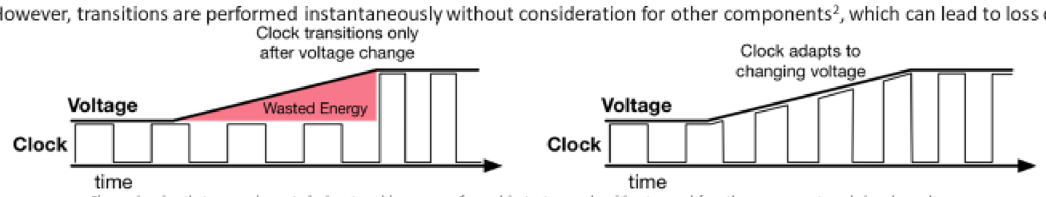

Clock transitions only after voltage change
Voltage
Wasted Energy
Clock
time
Clock adapts to changing voltage
Voltage
Clock
time

Figure showing that energy is wasted when transitions are performed instantaneously without regard for other components and changing voltages

Our approach determines equivalence classes of operating point groups among different components and parallelizes DVFS requests to better adapt the frequencies of each component

To achieve this, we built dependency graphs between components by running benchmarks for all possible configurations of voltage and frequency for each component and by collecting the benchmark and utilization numbers for each run allowing us to characterize the interactions among different components

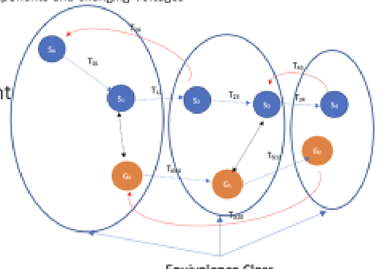


Figure depicting determining the equivalence classes from a dependency graph

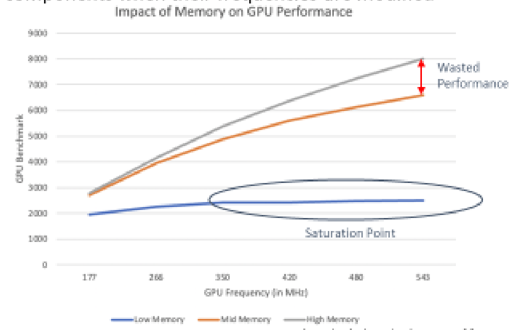
(1) B. Acun et al. Fine-Grained Energy Efficiency Using Per-Core DVFS with an Adaptive Runtime System. 2019 IGSC. doi: 10.1109/IGSC48788.2019.8957174
(2) Q. Fettes et al. "Dynamic Voltage and Frequency Scaling in NoCs with Supervised and Reinforcement Learning Techniques," in *IEEE Transactions on Computers*, 1 March 2019, doi: 10.1109/TC.2018.2875476

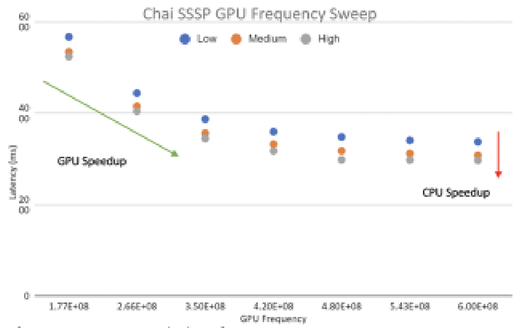
LABORATORY DIRECTED RESEARCH & DEVELOPMENT

WHERE INNOVATION BEGINS

R&D Summary (Methods, Results and Discussions)

Utilizing the AnTuTu, Chai, and Rodinia benchmarks, we were able to analyze dependencies between different components when their frequencies are modified





Plots depicting the impact of frequency changes of one component on another's performance

Combining these analyses, we were able to see how the frequency of each component impacted the performance of the other components

We tried multiple clustering options based on various methods and parameters:

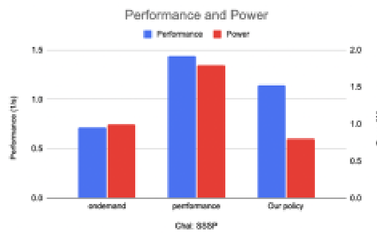
- Power-based: We clustered operating points based on power consumption
- Performance: Operating points were clustered based on the performance score
- Power-perf: We used multi-dimensional clusters, such as DBSCAN
- Energy-Delay Product(ν): We used energy-delay product with ν determining the balance of power and performance

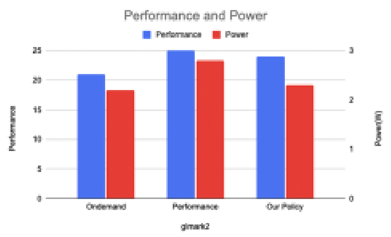
	MemPerf	CPUPerf	GPUPerf
MemFreq	0.921	0.632	0.776
CPUFreq	0.152	0.737	0.070
GPUFreq	-0.130	0.041	0.538

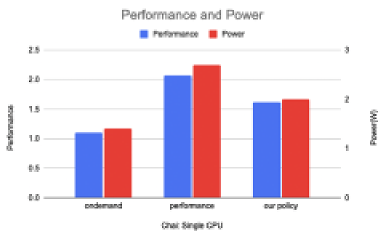
Table showing the impact of one component's frequency on the performance of the other components

We settled on a state transition policy based on the system-wide resource utilization and efficiency requirement ν and implemented this policy using scripts allowing for frequency modifications in parallel

R&D Summary (Methods, Results and Discussions)







Plots comparing the performance and power usage of a power-saving policy, a high performance policy, and our policy

In testing our policy against existing DVFS policies, we found that we were often able to achieve similar performance to high performance policies but with power usage closer to power-saving policies

We often observed energy savings around 15-20%, but in some cases, we observed an energy savings of 35% even though our scripts (written in Python) added an overhead of about 6-11% to the computation

Lessons Learned:

- It is difficult to modify the DVFS policy at the kernel level for components where we don't have source code
  - We spent a fair bit of time attempting to make these modifications directly but found it easier (and still very effective) to change the frequency with scripts making these requests
- The overhead from running Python code is not trivial though we were still able to see great gains despite this overhead

Project Legacy

Key Technical Accomplishment

- Achieved greater than 10% (in some cases 35%) power savings over existing DVFS methods

This work benefits Sandia's Global Security and National Security Program by reducing energy consumption providing resilience to remote sensors, mobile devices, and military devices.

- Worked with Professor Gul Agha, an Academic Alliance professor at University of Illinois Urbana-Champaign
- Discussed application of our power saving methods to decision support systems with members of center 6300
- Communicated with the Power API[1] Team to learn about how our methods might be applied to their workloads
- Encouraged by IAT member to submit full LDRD to the CIS call in ACS

We plan to submit a full CIS LDRD in FY24 to further develop our approach for application to mobile sensing missions.

- Redesign our scripts to run on decision support system hardware and focus on both power savings and performance
  - Sandia will determine hardware specifications and software APIs available in current systems by end of 2022
  - UIUC will implement a mode where performance is optimized for a desired power usage by early 2023

We wish we could have produced software that would automatically perform the frequency clustering and equivalence class computation for any given system. We would have also liked to convert the scripts to something that would run with less overhead such as Bash scripts.

(1) R. E. Grant et al, "Standardizing Power Monitoring and Control at Exascale," in Computer, vol. 49, no. 10, pp. 38-46, Oct. 2016, doi: 10.1109/MC.2016.308

LDRD Project Metrics

Presentations and Publications

- IEEE Transactions on Parallel and Distributed Computing (in preparation)

Intellectual Property

- Requires further evaluation and demonstration of utility; however, algorithm shows promise as being applicable on extending battery life of mobile devices

Tools and Capabilities

- Developed a suite of Python scripts that can be fed a collection of clustered frequency equivalence classes for each component in a heterogeneous system to produce and run a policy that will automatically adjust the frequencies to obtain greater performance and power savings

Staff Development

- This LDRD supported an early career employee, Curtis Madsen, who was also the PI on the project

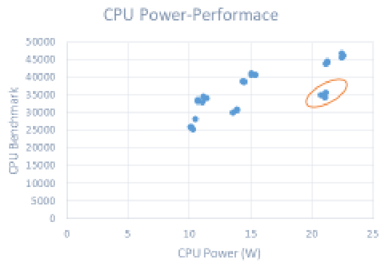
Awards

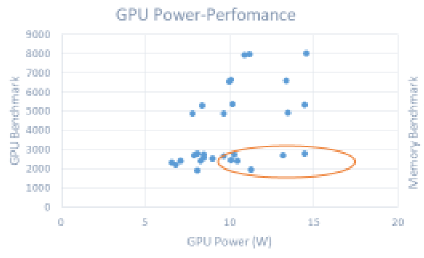
- None

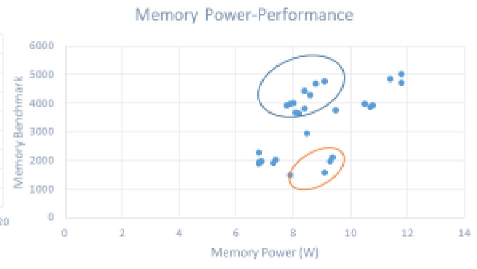
LABORATORY DIRECTED
RESEARCH & DEVELOPMENT
WHERE INNOVATION BEGINS

Backup Slides

When analyzing the performance and power usage of each component, we identified cases where each component either underperformed or overperformed given the power used for different frequency values of the other components







Plots showing where components performed poorly at higher power usage (orange regions) and where components had great performance at lower power usage (blue regions) for varying frequencies of the other system components

Backup Slides

Different clustering options resulted in combinations of frequencies being placed in different equivalence classes as shown in the example below

[177, 1.4, 165]	0
[350, 1.4, 165]	0
[543, 1.4, 165]	0
[177, 1.7, 165]	1
[350, 1.7, 165]	1
[543, 1.7, 165]	1
[177, 1.4, 543]	2
[177, 1.4, 933]	2
[350, 1.4, 543]	2
[350, 1.4, 933]	2
[543, 1.4, 543]	2
[543, 1.4, 933]	2
[177, 2.0, 165]	2
[350, 2.0, 165]	2
[543, 2.0, 165]	2
[177, 1.7, 543]	3
[177, 1.7, 933]	3
[350, 1.7, 543]	3
[350, 1.7, 933]	3
[543, 1.7, 543]	3
[543, 1.7, 933]	3
[177, 2.0, 543]	4
[177, 2.0, 933]	4
[350, 2.0, 543]	4
[350, 2.0, 933]	4
[543, 2.0, 543]	4
[543, 2.0, 933]	4

Performance based cluster

[177, 1.4, 165]	0
[350, 1.4, 165]	0
[543, 1.4, 165]	0
[177, 1.4, 543]	0
[177, 1.4, 933]	0
[350, 1.4, 543]	0
[350, 1.4, 933]	0
[543, 1.4, 543]	0
[543, 1.4, 933]	0
[177, 1.7, 165]	1
[350, 1.7, 165]	1
[177, 1.7, 543]	1
[543, 1.7, 543]	1
[177, 1.7, 933]	2
[543, 1.7, 933]	2
[177, 2.0, 165]	3
[350, 2.0, 165]	3
[543, 2.0, 165]	3
[177, 2.0, 543]	3
[177, 2.0, 933]	4
[543, 2.0, 933]	4

Power based cluster