5-2018

Power-cognizant Computing

Philip Gerald Vaccaro
University of Tennessee, pvaccaro@vols.utk.edu

To the Graduate Council:

I am submitting herewith a thesis written by Philip Gerald Vaccaro entitled "Power-cognizant Computing." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Jack Dongarra, Major Professor

We have read this thesis and recommend its acceptance:

Michael R. Jantz, Stanimire Z. Tomov

Accepted for the Council:
Dixie L. Thompson

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

Power-cognizant Computing

A Thesis Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Philip Gerald Vaccaro

May 2018

This labor of a rather lethal concoction of emotions is dedicated to the one-of-a-kind feline soul whose light burned too bright for our world. I realize this may not be the standard subject of a dedication, but that cat was one of the few things that kept me going through one of the hardest stretches of time in my life. Without the loving bond we shared I honestly don't know if I would even be typing this. I dedicate this culmination of my work to the angelic spirit that reminded me how beautiful life can be. Love you Beanie girl.

Acknowledgments

I would like to thank my wonderful girlfriend Chloé for all of her love and support. My family has also been an unrelenting force of positive through my whole life and academic career. I can never thank them enough. A very special thank you is dedicated to the University and the Innovative Computing Lab. I sincerely thank my adviser Professor Jack Dongarra for allowing me the opportunity to do this exciting work.

Abstract

The path towards exa-scale computing systems has yielded systems with performance bottlenecks that are increasingly hard to discover. Over time a steadily rising number of hardware performance metrics have been made available to application developers for optimization purposes. One emerging metric that has been singled out as being useful is power consumption. So far, understanding how different components of complex heterogenous computing systems consume energy has proved useful in investigating optimal power requirements for applications. This work can be broken up into two main areas: power profiling and power-cognizance. To gain insights regarding power consumption of complicated systems, there must be a consistent software interface for reliably profiling applications and collecting power metrics for analysis. That serves as the foundation for making inferences about real-world power requirements for different categories of applications. Using power profiles of applications to deduce trends in power consumption could provide a means for optimizing in pursuit of power-cognizance. The results of efforts in both power profiling and power-cognizance research are presented. Power profiling pursuits materialized as a component that is now a part of the Performance API (PAPI), known as powercap. Efforts in power-cognizance research took the form of utilizing powercap to collect power measurements and enforce power limits at run-time. The intention of this work is to explore the computing landscape of the future with the goal of drawing meaningful conclusions about application behavior in power-managed environments.

Table of Contents

1 Introduction **1**

1.1 Managing Power 1

1.1.1 Running Average Power Limit (RAPL) 1

1.2 Intel RAPL MSR Interface 2

1.3 Intel Powercap Sysfs Interface 2

1.4 Thesis Statement 3

2 Software-defined Power Management **4**

2.1 Performance API (PAPI) Powercap Integration 4

2.2 Dynamic Power Management 4

2.3 Computational Characteristics of Applications 5

2.4 Optimization Implications 5

2.5 Modeling Power 6

3 Characterizing Applications **7**

3.1 Goals 7

3.2 Representative Applications 7

3.3 Power Profiling 9

3.3.1 Power Profiling Framework 9

3.3.2 Power Profiling Environment 9

3.4 Assessing Power Profiles 10

4 Power and Computational Intensity **11**

4.1 Understanding Compute-bound Power Profiles 11
4.2 Compute-bound Implications 12
4.3 Understanding Memory-bound Power Profiles 14
4.4 Memory-bound Single Limit Analysis 17
4.5 Memory-bound Multi-limit Analysis 21
4.6 Complex Power Profiles 23

5 Power Management Moving Forward **27**
5.1 Understanding Power Consumption 27
5.2 Analyzing with Hardware Counters 27
5.2.1 Compute-bound Implications 28
5.2.2 Memory-bound Implications 30
5.2.3 Interleaved Implications 32
5.3 Moving Forward 34

Bibliography **35**

Vita **37**

List of Tables

3.1 Representative Applications 8

List of Figures

4.1 Power profile of an 18K DGEMM executing on DDR4. 12
4.2 Power profile of an 18K DGEMM executing on MCDRAM. 13
4.3 Power profile of an 18K DGEMM executing on DDR4 under a 215 Watt powercap. 13
4.4 Power profile of an 18K DGEMM executing on DDR4 under a 155 Watt powercap. 14
4.5 Power profile of an 18K DGEMM executing on DDR4 under multiple powercap limits. 15
4.6 Power profile of an 18K DGEMV executing on DDR4. 16
4.7 Power profile of an 18K DGEMV executing on MCDRAM. 17
4.8 Power profile of an 18K DGEMV executing on DDR4 under a 215 Watt powercap. 18
4.9 Power profile of an 18K DGEMV executing on MCDRAM under a 155 Watt powercap. 18
4.10 Power profile of an 18K DAXPY executing on DDR4 under a 215 Watt powercap. 20
4.11 Power profile of an 18K DAXPY executing on MCDRAM under a 155 Watt powercap. 20
4.12 Power profile of an 18K DGEMV executing on MCDRAM under multiple powercap limits. 21
4.13 Power profile of an 18K DAXPY executing on MCDRAM under multiple powercap limits. 22
4.14 Power profile of a 12K DGESVD executing on DDR4 under a 215 Watt powercap. 23

4.15 Power profile of a 12K DGESVD executing on MCDRAM under a 215 Watt powercap. 25
4.16 Power profile of a 12K DGESDD executing on MCDRAM under a 215 Watt powercap. 25

5.1 Power profile of an 18K DGEMM executing on DDR4 under multiple powercap limits alongside hardware counters. 28
5.2 Power profile of an 18K DGETRF executing on MCDRAM under multiple powercap limits alongside hardware counters. 29
5.3 Power profile of an 18K DGEMV executing on DDR4 under multiple powercap limits alongside hardware counters. 31
5.4 Power profile of a Jacobi executing on MCDRAM under multiple powercap limits alongside hardware counters. 32
5.5 Power profile of an 12K DGESDD executing on MCDRAM under multiple powercap limits alongside hardware counters. 33
5.6 Power profile of an 12K DGESDD executing on MCDRAM under multiple powercap limits alongside hardware counters. 34

Chapter 1

Introduction

1.1 Managing Power

In order to investigate power consumption of applications, a means of measuring and managing power must be employed. In modern computing systems, hardware has the ability to provide users access to power-related metrics and settings. Metrics are useful for logging energy consumption for applications. By collecting energy consumption metrics over time, the power being consumed at each time-step can also be logged. Collecting power data in this way produces a power profile curve for a given application. The power settings of concern are associated with power limits for different hardware components in a system. Upon changing a power limit setting for a hardware component, the limit should be reflected in the power consumption measurements. A valid power management paradigm supporting power limiting must ensure that power consumption measurements reflect any power limit updates in order to ensure accuracy.

1.1.1 Running Average Power Limit (RAPL)

As described in Intel 64 and IA-32 Architectures Software Developers Manual [2], modern Intel processors employ the RAPL model in order to expose power measurements and settings. This feature provides a means of modeling power consumption via an internal methodology based on hardware-specific attributes. Power limiting is accomplished in

RAPL by accepting user-specified power limits for different hardware components as inputs. Enforcing power limits in RAPL is enacted by way of Dynamic Frequency Scaling (DVFS) [5]. Employing DVFS makes it possible for processors to operate at a frequency that falls in a processor-specific range of frequency settings. Adjusting the frequency that a processor is operating at has a direct impact on the power consumed by a given processor.

1.2 Intel RAPL MSR Interface

Intel provides access to the RAPL model via model-specific registers (MSRs). These MSRs are restricted system files that require root privileges to access. Each core in a processor has an individual MSR file that corresponds to power measurements and settings. Every MSR is divided up into sections that are denoted by documented offsets. These MSR offsets serve as indexes to relevant bytes that correspond to specific power metrics and management settings. For example, every Intel processor supporting RAPL provides access to an MSR at an offset which corresponds to a package-wide energy counter. Sampling this package-wide energy counter at regular sample intervals allows instantaneous power consumption to be collected over the course of an applications execution. The main drawback to using the RAPL MSRs for power management is the fact that it requires the highest level of access privileges to even read an energy counter. However, the concept of a standard model that provides users access to reliable power metrics and settings is key for finding a general solution for energy-efficient computing.

1.3 Intel Powercap Sysfs Interface

The powercap interface provided by Intel provides users an intuitive abstraction layer for accessing RAPL MSR content. In this paradigm, power measurements and settings are exposed through system files that are much less restricted to access when compared with RAPL MSRs. Any user on a powercap-supported system is able to read power measurements without elevated privileges. Also, the user need only have write permission for the power limit related files in order to enforce power limits. The powercap functionality is made

available through system-wide files that correspond to a single power management aspect. For example, on systems that support package and DRAM power measurements, a single aptly named file corresponds to the current energy count for each individual hardware component.

1.4 Thesis Statement

The importance of power consumption in the computing landscape will only increase as hardware technology continues to evolve. Modern computing systems contain many hardware components, each with individual power consumption requirements. To get the most out of these powerful systems, insight must be gained concerning high-performance applications and their energy efficiency. Understanding how a diverse set of applications consume power is key to deducing whether or not energy savings can be realized while maintaining performance. The goal of this work is to describe how hardware metrics can be used as a means for modeling power consumption requirements. As a side effect, it is also shown how using power measurements alone is not sufficient for understanding consumption requirements in all cases.

Chapter 2

Software-defined Power Management

2.1 Performance API (PAPI) Powercap Integration

As part of an expansion of PAPI [6] to incorporate more power management functionality, a powercap component was added to the widely used performance profiling interface. With the addition of the new component, PAPI users gain the ability to access power measurements and settings through the standard PAPI interface. This allows for seamless integration into codes instrumented with PAPI functionality for profiling and optimization purposes. Code that includes PAPI powercap instrumentation can read energy counters and set power limits simply by accessing the corresponding PAPI powercap event. Each PAPI powercap event maps to a powercap system file which represents a given power attribute of a given hardware component. Adding this dimension to the performance profiling landscape allows for a burgeoning aspect of computing to be explored.

2.2 Dynamic Power Management

Work concerning static power limiting at the start of execution has been explored in [1], however using power measurements to categorize applications was not explored in depth. One of the main benefits of using power management functionality as part of performance profiling is the ability it provides for setting power limits at run-time. Dynamically setting power limits over the course of execution of an application allows users to apply power

limits and observe the effects at the same time. This can be accomplished by using PAPI powercap functionality to sample power measurements at regular intervals and set power limits when desired. Thus, running an application under multiple power limits can shed light on its intrinsic computational characteristics since power consumption can simultaneously be logged for later analysis.

2.3 Computational Characteristics of Applications

Applications tend to fall in one direction on a two-way computational intensity spectrum. Those containing many memory access per operation are said to be memory-bound in nature, while those performing few memory accesses per operation are referred to as compute-bound. These two categories of computationally intensity have inherently different energy consumption requirements due to the fact that a processor waiting on data to be fetched is essentially operating idly. If the predominant computational factor for an application is memory bandwidth, a processor can consume unnecessarily high amounts of power. Enforcing lower power limits during these periods of execution for these applications can be used as a way of deducing whether or not it is a good candidate for energy savings. On the other hand, compute-bound applications will operate at near peak power levels since the bottleneck is the processor. Since current processors operate on data at a faster rate than memory hardware can retrieve it, optimal energy efficiency can usually be attained at near peak power levels for compute-bound applications. This implies that the benefits of power limiting compute-bound codes are potentially small; however, it still may be the case that energy savings actually can be had for that computational category.

2.4 Optimization Implications

Having a reliable mechanism for collecting power measurements provides users the opportunity to characterize different applications based on their power profiles. For example, on a system supporting package and DRAM power measurements, an application could be profiled for power consumption in an attempt to understand energy consumption of the DRAM

hardware component. If an application instrumented with PAPI powercap functionality indicates DRAM power consumption is at the DRAM power limit, it is likely that application is memory-bound in nature. These cases often result in the package power consumption being below the maximum power limit of a given processor. Furthermore, the package power limit consumption during the execution of memory-bound applications can still be higher than is actually necessary to maintain performance.

2.5 Modeling Power

The ultimate goal of these pursuits is a real-time strategy for power management during the execution of target applications. Using cycle-related hardware counter suggest an optimal power limit is feasible given a commonality between applications of each computational intensity category. Hardware counters can be sampled simultaneously with power metrics on a system. This allows for decision calculations to be made, on a per-sample basis, by analyzing the hardware counter relationships at each time slice. Subsequently, a per-sample decision for an optimal power limit could be determined at the rate performance metrics can be sampled. Thus, if the hardware counters can be reliably used as a power management decision factor, there is relatively low overhead to pursuing possible solutions.

Chapter 3

Characterizing Applications

3.1 Goals

A key aspect of application analysis via power profiling is the ability to reliably deduce computational tendencies. In order to accomplish this, it is useful to establish categories of computational intensity for the purposes of analysis. The categories used for this analysis fall into three groups: memory-bound, compute-bound, and interleaved. Interleaved applications present a complex power consumption pattern since they consist of interwoven sections of both memory- and compute-bound segments of execution. This fact underscores the benefit of having the ability to dynamically enforce power limits through PAPI powercap instrumentation. Without this functionality, a power limit could only be maintained at the scope of an entire program instead of the granularity of individual code segments. By choosing representative applications that fall into each category for analysis, a diverse set of data can be established and analyzed. Analysis of this data can be used to aide in the pursuit of observing power trends among codes of the same computational category.

3.2 Representative Applications

The applications chosen for characterization are implementations of common linear algebra algorithms that are widely used in the High-performance Computing community. Certain applications were selected specifically due to their strong tendency to either be intuitively

memory-bound or compute-bound in nature. However, applications that exhibit interleaved segments of both type over time also had to be chosen to capture power profiles of the third aforementioned category. The different computational categories and the associated representative applications are shown in the 3.1.

Table 3.1: Representative Applications

Compute-bound	Memory-bound	Interleaved
DGEMM	HPCG	DGESDD
DGETRF	Jacobi	DGESVD
DGETRFBIS	DGEMV	-
-	DAXPY	-

In an attempt to round out each category, linear algebra applications were chosen from the widely used Intel Math Kernel Library (MKL) [4]. The **compute-bound** candidates were selected because the bulk of their execution time is spent on many operations when compared to number of memory accesses. The kernels DGEMM, DGETRF, and DGETRFBIS exhibit the computational behavior of performing many mathematical operations in conjunction with minimal movement of large data segments. Applications that tend to follow this computational pattern are only performance limited by the capabilities of the processor. Conversely, the applications HPCG, Jacobi, DGEMV, and DAXPY are categorized as **memory-bound** because their computational pattern consists of mathematical operations that depend upon many data movement operations in order to complete. However, it is not to say that compute-bound applications do not also require many data movement operations as well. The key distinction between compute- and memory-bound codes is that in the latter case the overarching performance bottleneck is the inability for the memory bandwidth of a system to deliver data to the processor in a timely manner. Data size highly influences whether or not the memory hardware of a system will be able to keep up with the processor under the stress of a given executing application. Moving small data elements many times has much less unpleasant consequences than the situation where an application is moving large data elements with even moderate frequency. The **interleaved** category containing DGESDD and DGESVD represents linear algebra applications that consist of both compute-

and memory-bound computational sections of execution. For example, DGESVD has a long memory-bound section at the start of execution followed by an alternating mix of compute- and memory-bound sections that run until completion.

3.3 Power Profiling

As previously mentioned, a power profile of an application can yield valuable information which can help identify trends in power consumption patterns. This power profiling process was carried out for the aforementioned representative linear algebra applications and the results were analyzed. This was done in the pursuit of deducing power consumption commonality among applications thought to fall into the same computational category.

3.3.1 Power Profiling Framework

A framework for performance counter sampling was developed in order to facilitate the profiling of applications for power consumption. The framework utilizes PAPI powercap component functionality to sample user-specified at a regular sample interval. The benefit to using PAPI powercap is that it also opens the door to collecting measurements of other counters that are made available through its interface. Because the same interface can be used to collect many different events with PAPI, it can be used as a foundation for a light-weight sampling interface for logging performance data over time. This approach is very important when dealing with power measurements due to the fact that the raw counter values are provided as energy in Joules, not power. Thus, an accurate sampling interface is key for accurate computations of instantaneous power, which is derived as energy over a given sample interval of time.

3.3.2 Power Profiling Environment

Research utilizing this performance profiling framework was conducted on a single Intel Knights Landing (KNL) socket containing two non-uniform memory access (NUMA) nodes. The first NUMA node consists of a DDR4 memory component and the second NUMA node

has an MCDRAM memory component [3]. This is significant due to the fact that the maximum bandwidth capacity for MCDRAM is four times that of DDR4. Tangentially related is the existence of a utility known as *numactl*, which serves as a way of designating an application to execute using the memory device on a specific NUMA node. The utility can accept a NUMA node identification number as well as path to a program to be executed on said NUMA node. This allows for attaining performance counter profiles for individual NUMA nodes and performing power analysis at an even finer level of detail in terms of specific hardware components.

3.4 Assessing Power Profiles

Scores of power profiles were collected for all of the applications listed in 3.1. Results from applications of each category are shown and described in order to aide in the analysis of power consumption requirements for applications of various computational intensities. An example of a power profile consisting of package and DRAM power is shown in 4.1. This visualization shows how power at the package and DRAM level is consumed at 100 millisecond sample intervals over the course of the execution of a DGEMM with a matrix size of 18000 double precision elements using the NUMA node containing DDR4 memory. It must be pointed out that only DDR4 power consumption is accounted for in the DRAM power measurement. This is not the case for MCDRAM because it resides on the actual CPU package and is thus accounted for in the package power measurements. This difference is evident in the separate graphical representations of power profiles of a single application executing on the distinct NUMA nodes.

Chapter 4

Power and Computational Intensity

4.1 Understanding Compute-bound Power Profiles

Figures 4.1 and 4.2 show power profiles of a DGEMM being executed using DDR4 and MCDRAM, respectively. In both cases there is an initial power spike up to nearly 250 Watts followed by an immediate power throttling down to around 215 Watts. This number is significant because it happens to be the Thermal Design Point (TDP) of this particular KNL computing system, which is a hardware enforced power limit. This hardware enacted power limit ensures that the system does not exceed the TDP limit for an amount of time that would be detrimental to the hardware. Once the TDP limit has been exceeded for a time deemed to be too long, the system drops the power down to conform to meeting the TDP requirements. These initial DGEMM figures illustrate how compute-bound applications tend to have computational behavior such that maximizing the utilization of the processor provides a near optimal execution environment in terms of power. This is due to the fact that the bottleneck for these types of applications has to do with the maximum rate at which a processor can operate at instead the rate at which memory is fed to the processor to be operated on. Since the processor is wildly ahead in terms of speed, compute-bound applications essentially attempt to maximize the most powerful computing resource for the longest possible amount of time before it becomes thermally unsafe to continue.

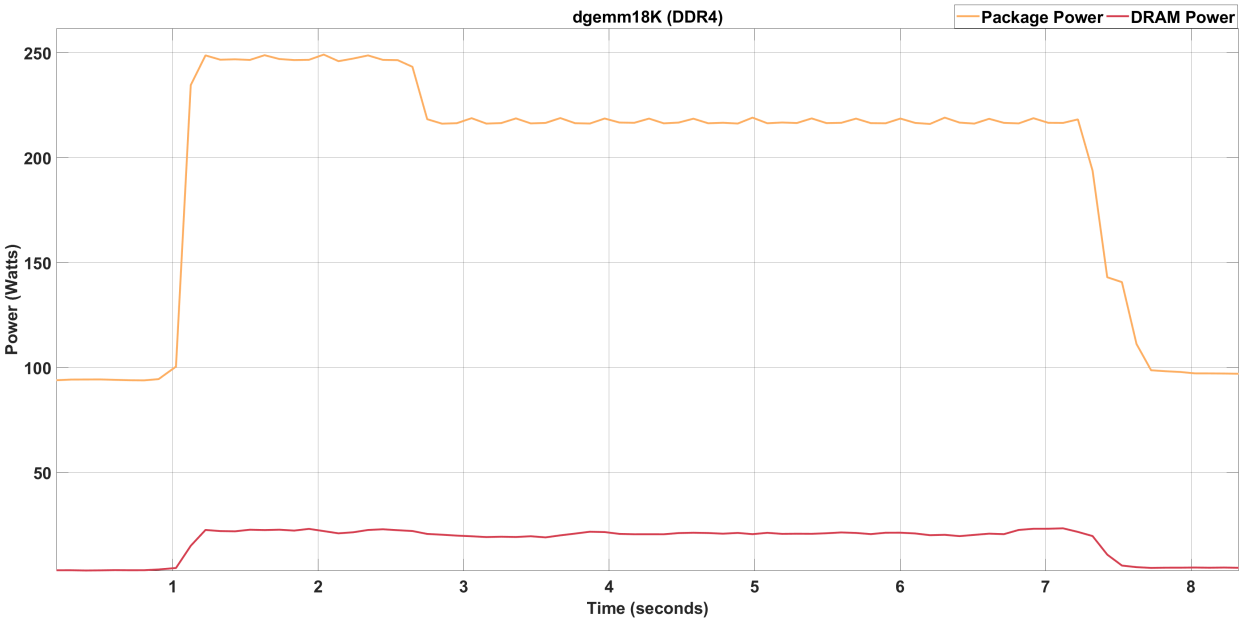

Figure 4.1: Power profile of an 18K DGEMM executing on DDR4.

4.2 Compute-bound Implications

Understanding when power limiting an application does not produce tangible benefits is key in tackling the issue of characterizing power profiles. For applications that are very compute-bound in nature, limiting the processing resources by restricting the maximum power of the system can have an adverse effect on performance. This insight is demonstrated when powercap enforces power limits on applications like DGEMM and other compute-bound applications like DGETRF. Figures 4.3 and 4.4 show power profiles of a DGEMM being executed on DDR4 at a 215 Watt power limit and 155 Watt power limit.

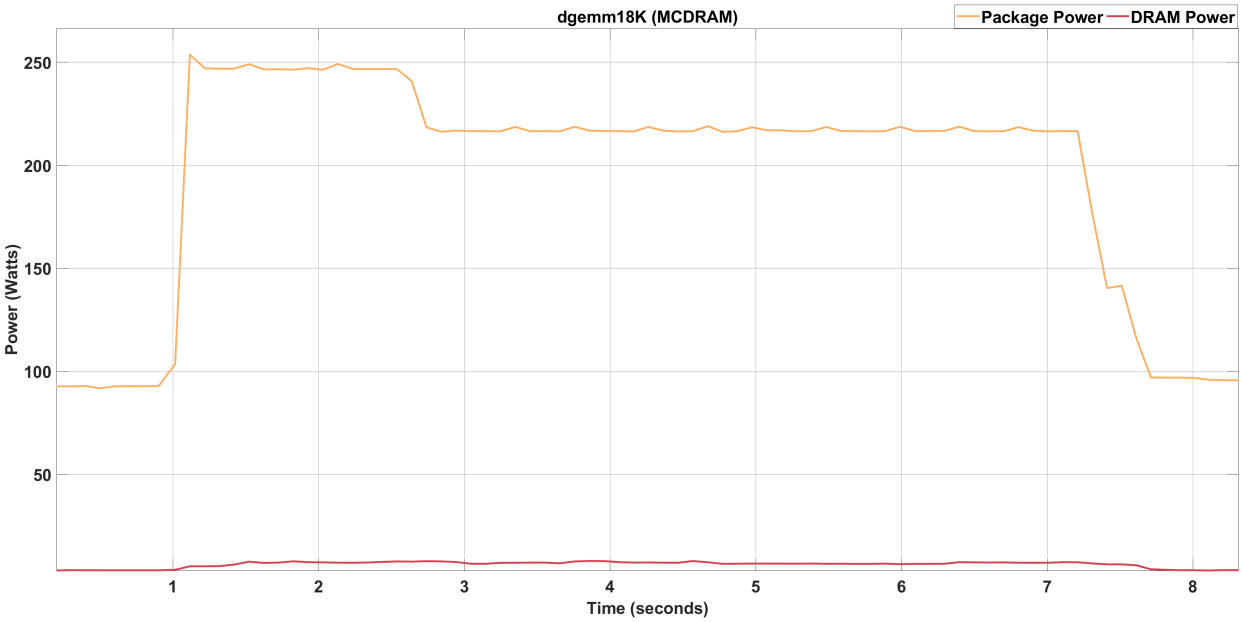

Figure 4.2: Power profile of an 18K DGEMM executing on MCDRAM.

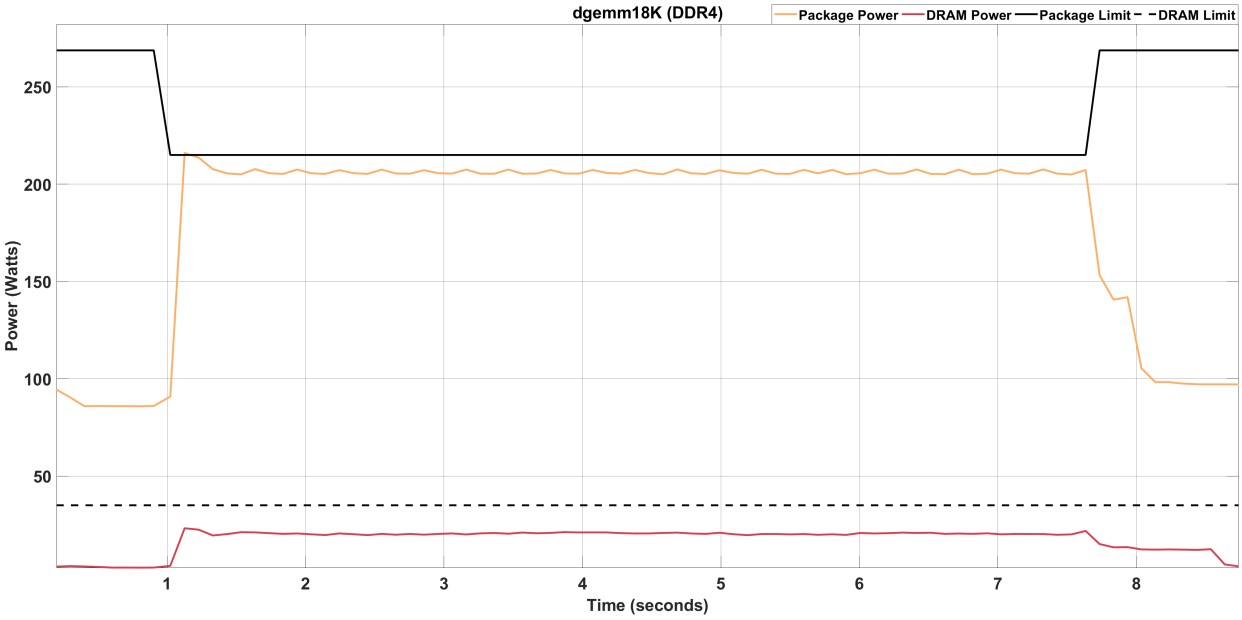

Figure 4.3: Power profile of an 18K DGEMM executing on DDR4 under a 215 Watt powercap.

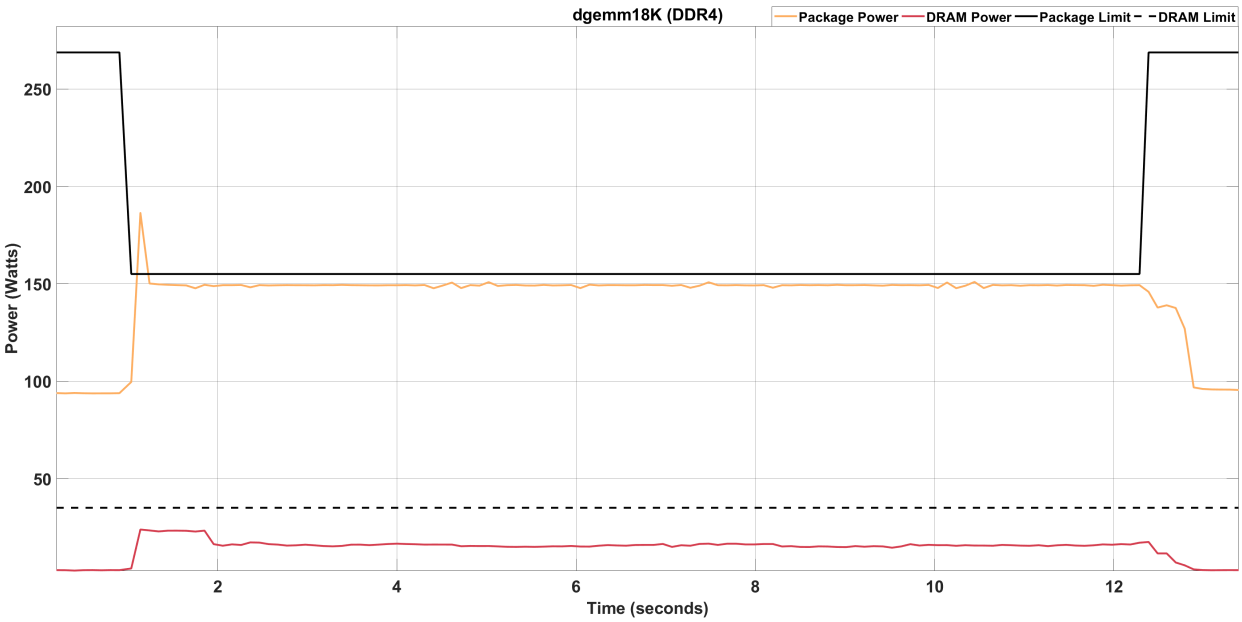

Figure 4.4: Power profile of an 18K DGEMM executing on DDR4 under a 155 Watt powercap.

Figure 4.5 illustrates what can happen when applications are executed under power limits that are too low for performance to be maintained. In this example, a DGEMM is executed at four power limits: 245, 215, 185, and 155 Watts. Note the continual increase in execution time as the power limit is decreased. The decrease in the package power limit translates into a decrease in the allowed peak frequency of the processor.

Compute-bound codes inherently depend on attaining the maximum desired frequency of the processor. Decreasing power inherently decreases processor frequency, which affects that rate at which data can be operated on. Codes classified as compute-bound can experience an overall loss in performance when executing under a power limit. However, that is not to say that finding the optimal power configuration for compute-bound codes could not result in energy savings.

4.3 Understanding Memory-bound Power Profiles

Similarly, Figures 4.6 and 4.7 show power profiles of a DGEMV being executed using DDR4 and MCDRAM, respectively. One initial observation of this data is that NUMA node selection has much more of a significant impact for DGEMV when compared to DGEMM.

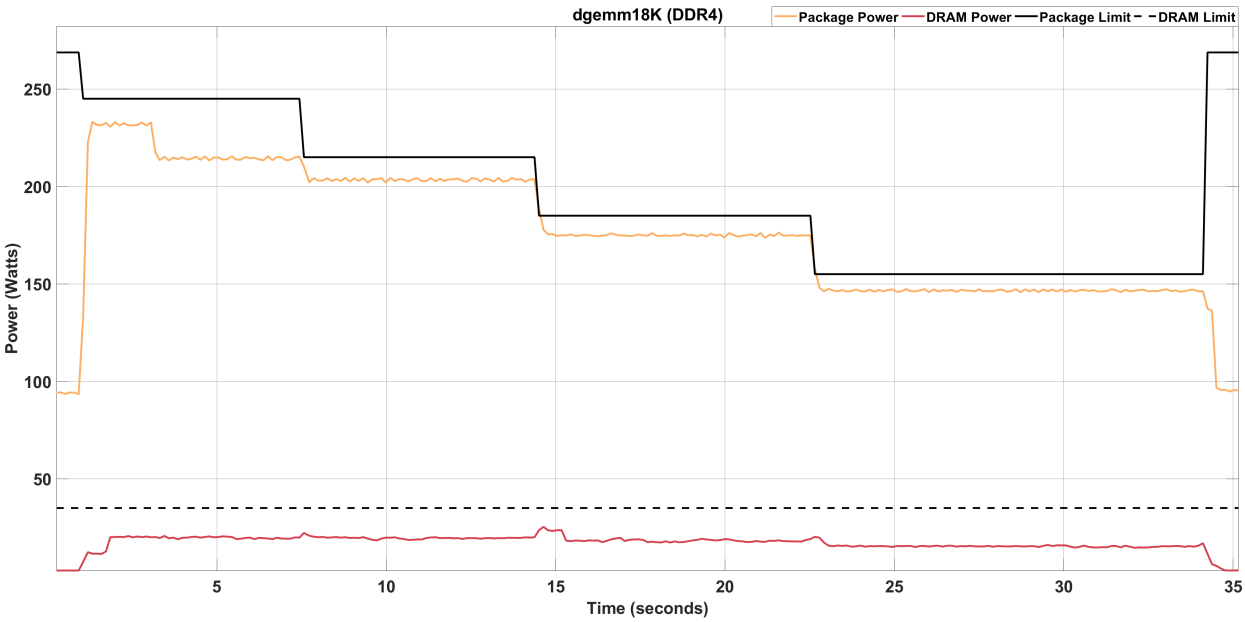

Figure 4.5: Power profile of an 18K DGEMM executing on DDR4 under multiple powercap limits.

The execution time of DGEMM in Figures 4.1 and 4.2 show that the type of NUMA node used has little effect on the performance of DGEMM at default power limits on this KNL system. This discrepancy is expressed in the execution time for the DGEMV datasets, which demonstrates MCDRAM execution time that is roughly four times faster than DDR4. Exhibiting this performance distinction when executing on different memory devices is an indication of a memory-bound application. Drawing this conclusion stems from the notion that if an application were compute-bound, then the bandwidth of the memory device in use would have little impact on the overall performance of the application.

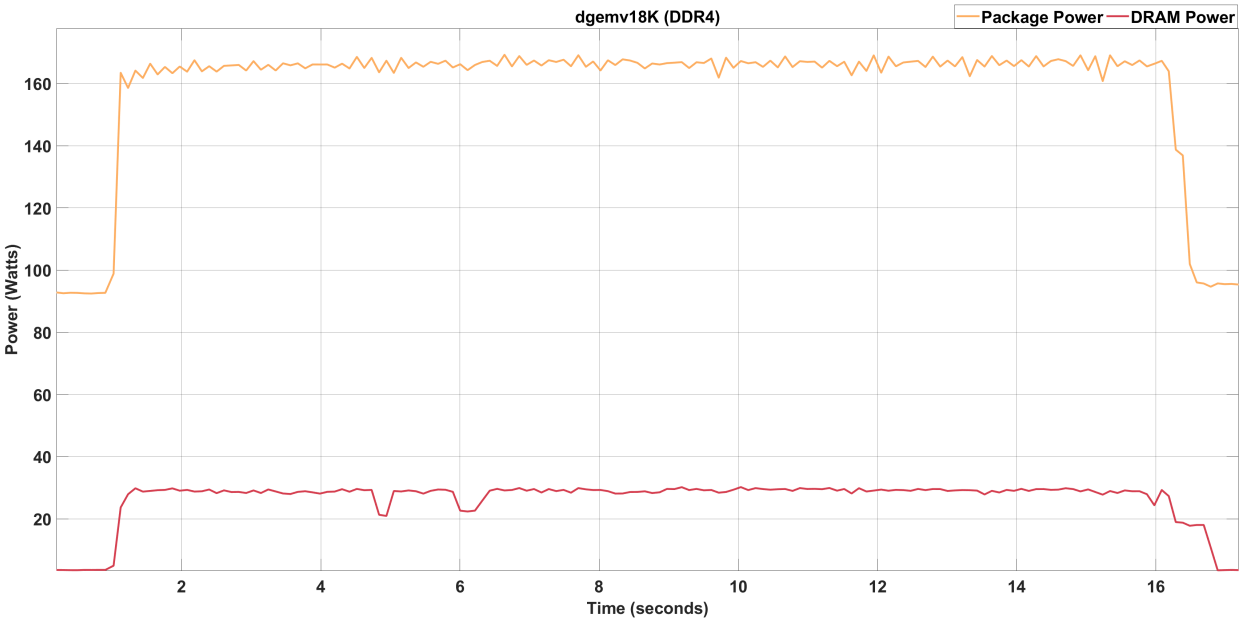

Figure 4.6: Power profile of an 18K DGEMV executing on DDR4.

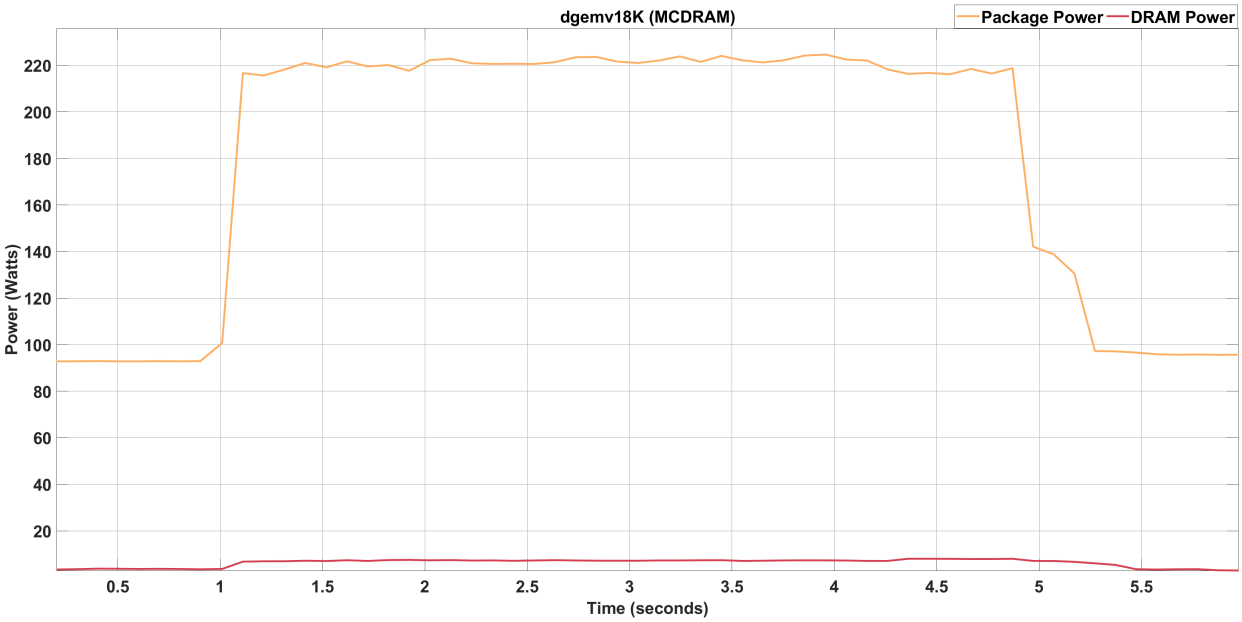

Figure 4.7: Power profile of an 18K DGEMV executing on MCDRAM.

4.4 Memory-bound Single Limit Analysis

Memory-bound applications behave differently when executed using NUMA nodes of differing bandwidth capabilities. Figures 4.8 and 4.9 show power consumption and limits of a DGEMV being executed using DDR4 and MCDRAM, respectively. Execution using MCDRAM results in a roughly four times faster run-time when compared to DDR4. These figures also depict a key distinction between power monitoring capabilities for different memory types. Since MCDRAM resides on the CPU package, its power consumption is accounted for in the package energy counter. When MCDRAM is used on a KNL system the DRAM counter is essentially useless because only DDR4 energy information is associated with it.

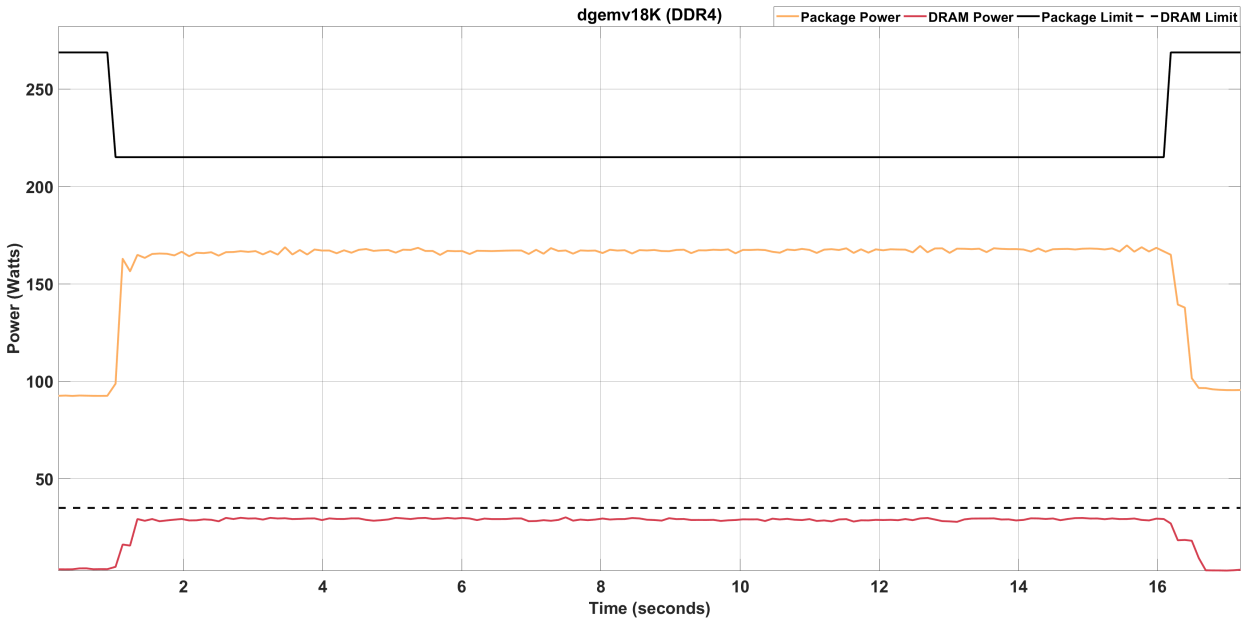

Figure 4.8: Power profile of an 18K DGEMV executing on DDR4 under a 215 Watt powercap.

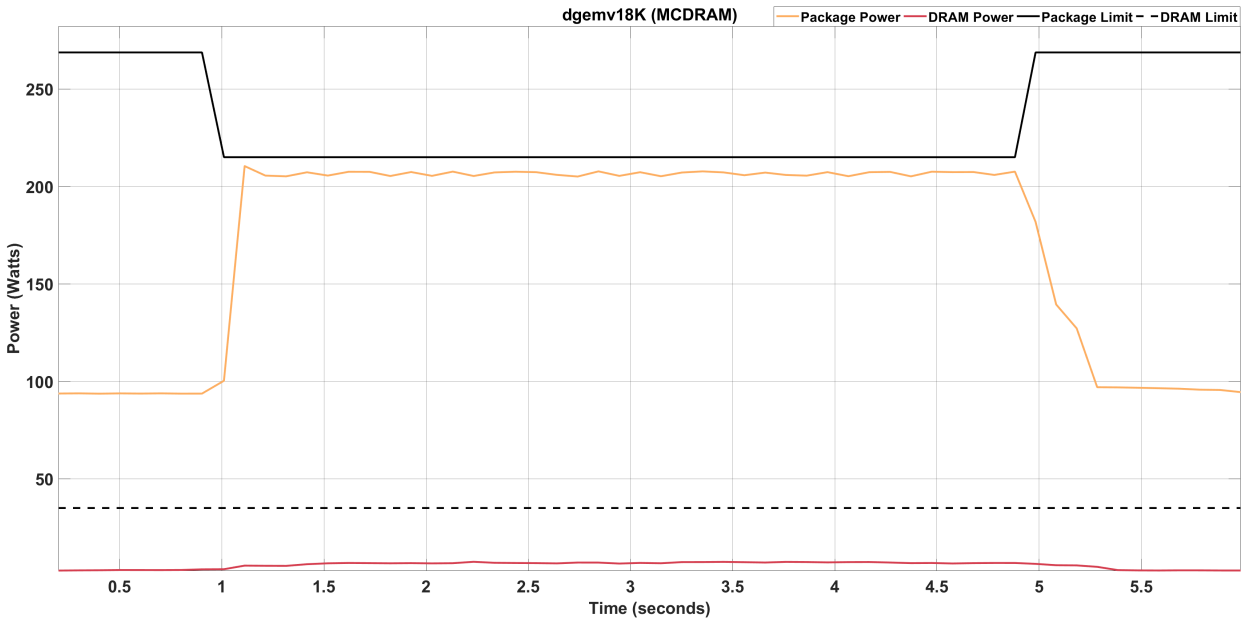

Figure 4.9: Power profile of an 18K DGEMV executing on MCDRAM under a 155 Watt powercap.

The power monitoring discrepancy between the two is is made evident in Figure 4.8 by way of the DRAM power consumption reaching very close to the peak DRAM power limit. An application that exhibits DRAM power consumption that is near the DRAM power limit can usually be classified as memory-bound in nature. Furthermore, when using DDR4 on KNL systems, applications like DGEMV bring DRAM power consumption up to the DRAM limit while package power consumption falls well below the default power limit of 268.75 Watts. Highly optimized compute-bound codes can also consume high amount of DRAM power and they also maximize the package power consumption as can be see in Figure 4.5. Computational behavior that is very similar to DGEMV can be observed in the DAXPY data depicted in Figures 4.10 and 4.11.

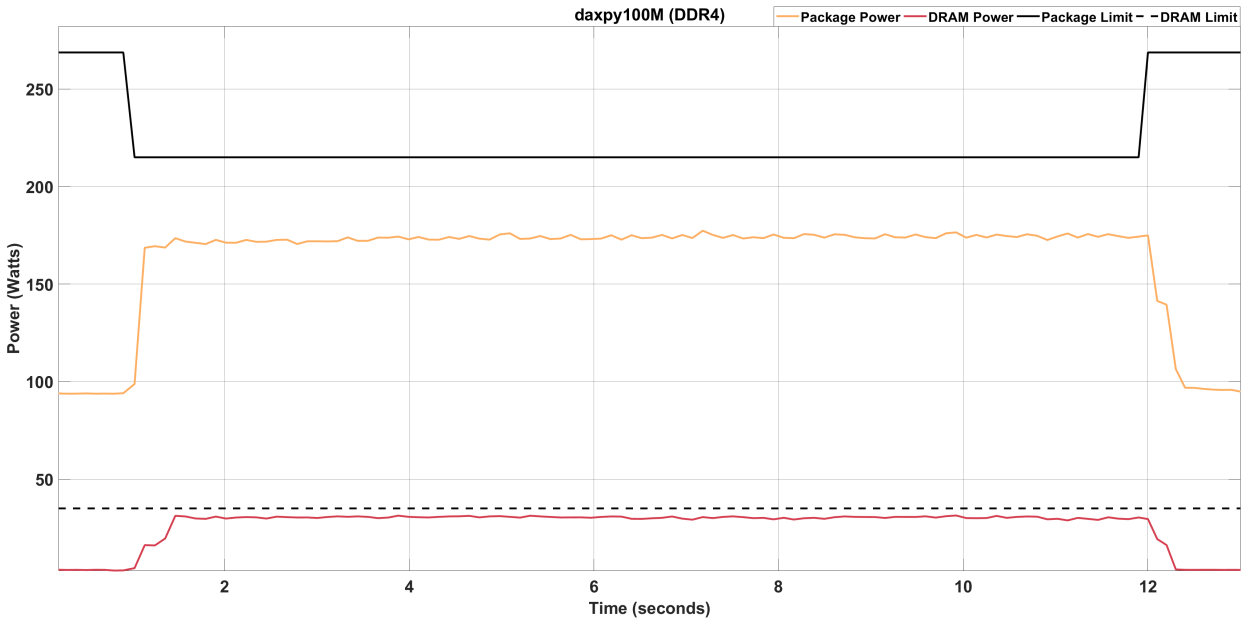

Figure 4.10: Power profile of an 18K DAXPY executing on DDR4 under a 215 Watt powercap.

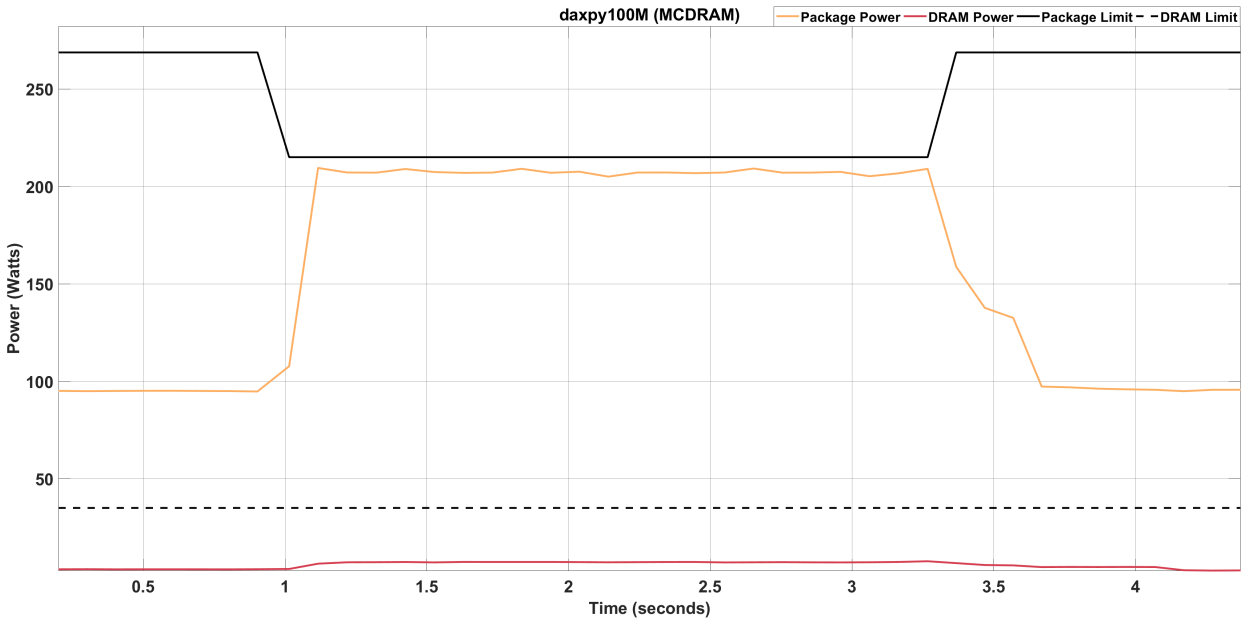

Figure 4.11: Power profile of an 18K DAXPY executing on MCDRAM under a 155 Watt powercap.

The graph showing DAXPY executing on DDR4 shows the same trend as DGEMV running on DDR4. The DRAM power limit is almost immediately reached and the package power is well below the imposed package power limit. The same behavior that DGEMV showed when running on MCDRAM can also be observed in the DAXPY graphs showing power consumption using MCDRAM. The similarities among memory-bound codes is key to isolating characteristics in the power data that can be exploited for energy savings.

4.5 Memory-bound Multi-limit Analysis

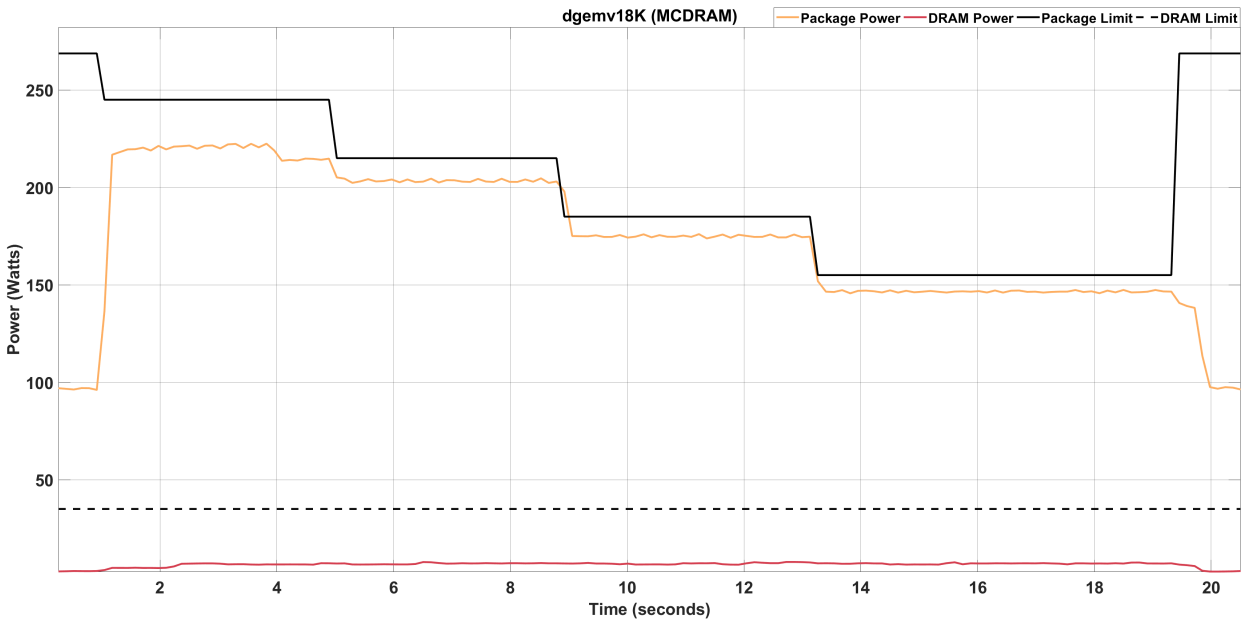

Figure 4.12: Power profile of an 18K DGEMV executing on MCDRAM under multiple powercap limits.

Analyzing multi-limit power profiles of memory-bound applications can help to shed light on common behavior. This approach allows for analysis of multiple executions of the same application under different power limits at the same time. Figures 4.12 and 4.13 display power profiles of back to back executions of each application. Each step in the package power limit line represents the setting of a new power limit via the power profiling framework. The power limits of 245, 215, 185, and 155 Watts are applied to each application in that order. Only the MCDRAM data for the runs is shown because the package power consumption using DDR4 is too low to have enough viable limits to depict the behavior. If the package

power reaches too close to the system idle power of 90 Watts, performance is dramatically reduced to the point of unreliability. The changes in run-time for multi-limit executions on MCDRAM illustrate the change in execution experienced by these memory-bound codes under software-enforced power limits.

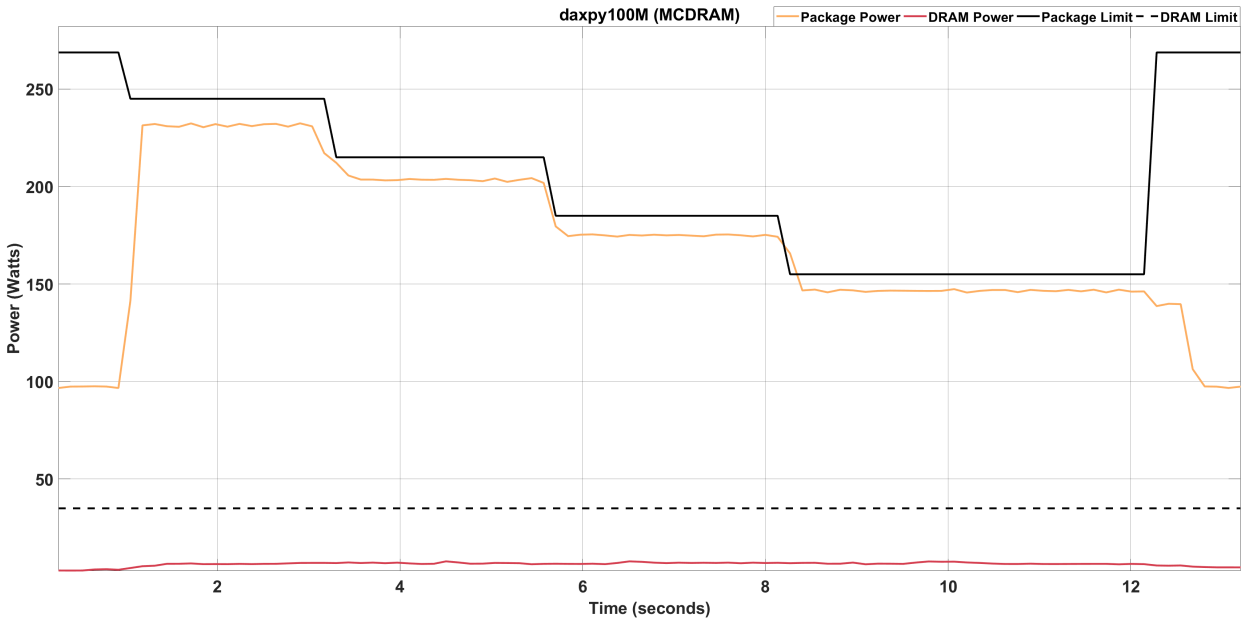

Figure 4.13: Power profile of an 18K DAXPY executing on MCDRAM under multiple powercap limits.

Figure 4.12 shows that DGEMV experiences a nontrivial increase in run-time under a 155 Watt power limit. Under the three higher power limits the performance does not take a substantial hit and it is clear the power is being saved. The notion of power savings becomes clear when analyzing the package power under the 215 and 185 Watt power limits. The package power line can be seen to immediately drop when both of those limits are applied, yet there is not a significant effect on performance despite the lesser amount of power being consumed. Thus, running DGEMV on a single socket KNL system using MCDRAM can be run at a package power limit of 245, 215, or 185 Watts and achieve performance that is nearly interchangeable. There is a slight drop in performance when running under the 185 Watt power limit, but this may be acceptable depending on the use case.

In Figure 4.13 it is clear that DAXPY exhibits a very similar power profile to that of DGEMV. Only at the power limit of 155 Watts is there an apparent decrease in performance that is non-negligible when using MCDRAM. This result confirms that applications which

experience near peak DRAM power consumption when using DDR4, can be tested at lower package power limits using MCDRAM in pursuit of energy savings. This is evident in the multi-limit DAXPY plot as it depicts the package power consumption immediately responding to the enforced power limits, while maintaining performance for the first three package power limits.

4.6 Complex Power Profiles

Ideally all applications only exhibited either compute- or memory-bound behavior and not both. However, there are many applications that go through compute- and memory-bound segments of execution. This category of applications was introduced in an earlier section as interleaved applications.

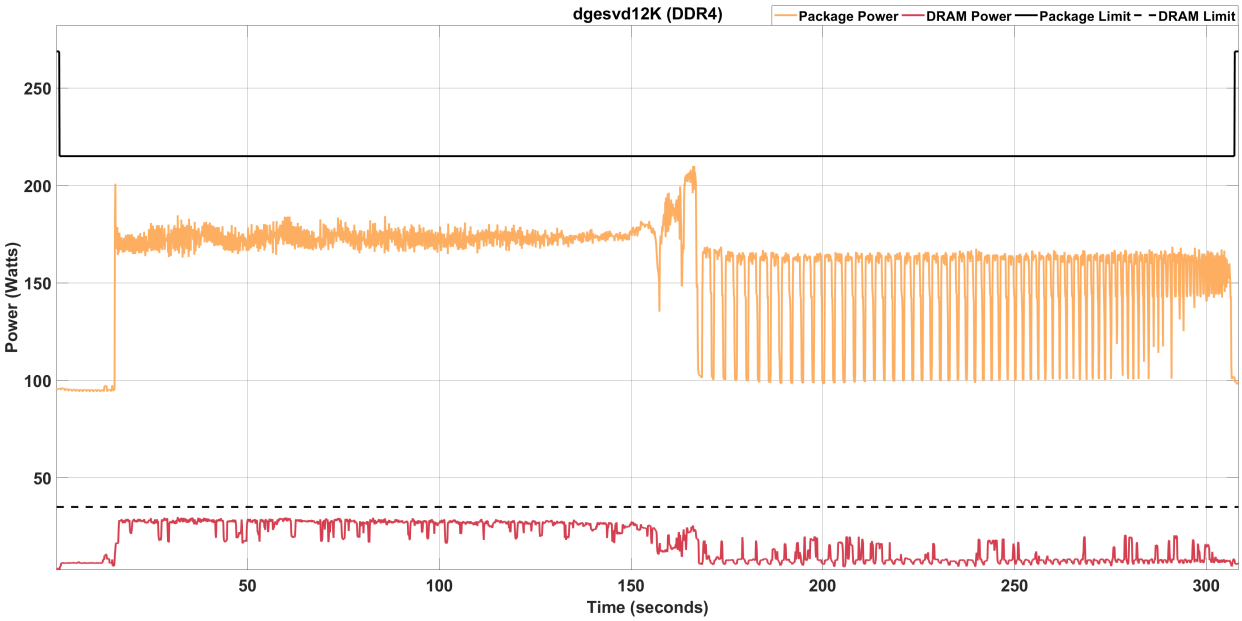

Figure 4.14: Power profile of a 12K DGESVD executing on DDR4 under a 215 Watt powercap.

Figure 4.14 shows the DGESVD routine being run under the TDP power limit of 215 Watts using DDR4. As indicated by the varying DRAM power consumption, the first half of DGESVD tends to be memory-bound, while the second half skews more compute-bound in nature. Segment of executon can be compute-bound and not consume peak package power at the same time. This is the case of DGESVD since the compute-bound segment does not consume power at a higher level than the memory-bound section. Applying a power limit to both segments can be detrimental to performance since the compute-bound section is consuming exactly the amount of power that it needs. This was not true in the case of the compute-bound application DGEMM, since the results show that applying power limits at the package level can have disastrous effects on performance.

Figure 4.15 shows a power profile for the same DGESVD application, but this execution was done using MCDRAM. The DRAM power consumption can no longer be used as a means of determining computational intensity, but previous results show which segments of execution tend to be memory- and compute-bound. There is a near 100 second speedup between the DDR4 and MCDRAM power profiles, which indicates there are segments of code that rely heavily on the speed of the available memory bandwidth. However, the speedup is not as extreme as the four times performance increase that was shown for DAXPY and DGEMV. Thus, it is safe to assume that some portions of DGESVD can benefit from energy savings when power limits are applied appropriately.

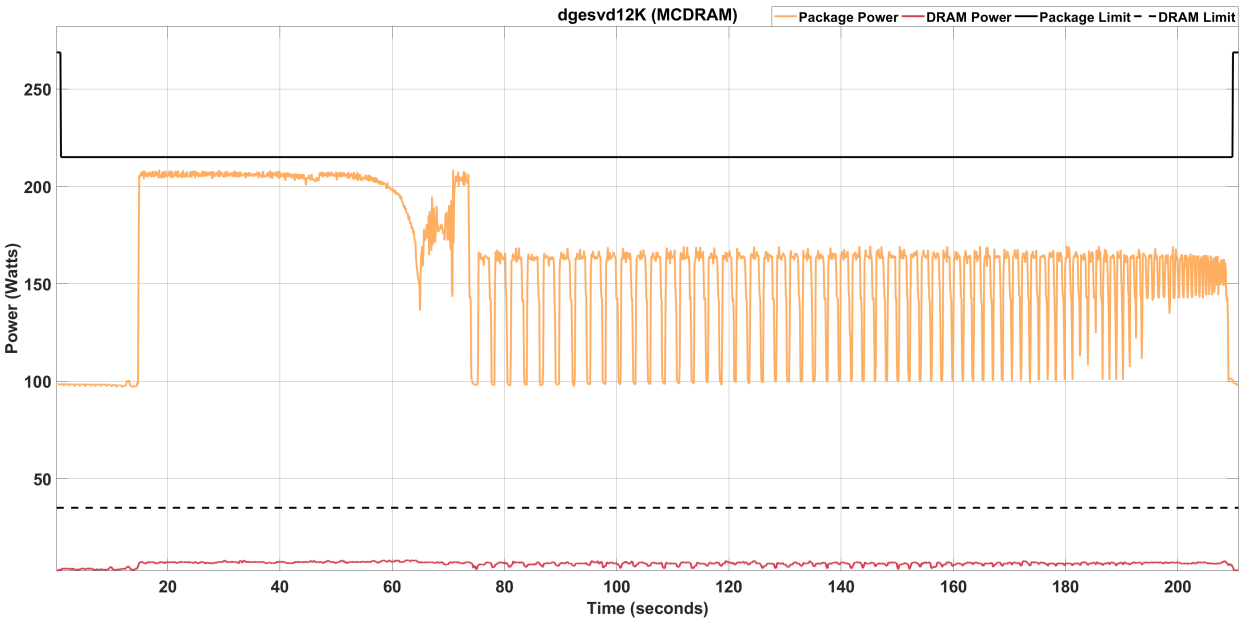

Figure 4.15: Power profile of a 12K DGESVD executing on MCDRAM under a 215 Watt powercap.

Applications like DGESVD reveal the complexity of determining optimal power limit configurations for complex interleaved codes when relying only on power consumption information of different hardware components. A power profile of another such application is shown in Figure 4.16. The DGESDD application is also an interleaved code that exhibits characteristics of both of the other two computational categories. Shown in the power profile for DGESDD is a similar trend to that of DGESVD. Both applications go through a long segment of near constant power consumption around the TDP limit of 215 Watts. This is once again a long stretch of memory-bound computational activity followed by a series of smaller segments of compute-bound behavior. Thus, the same potential energy savings are valid for DGESDD, but the challenge of accurately classifying segments of execution by computational intensity still remains.

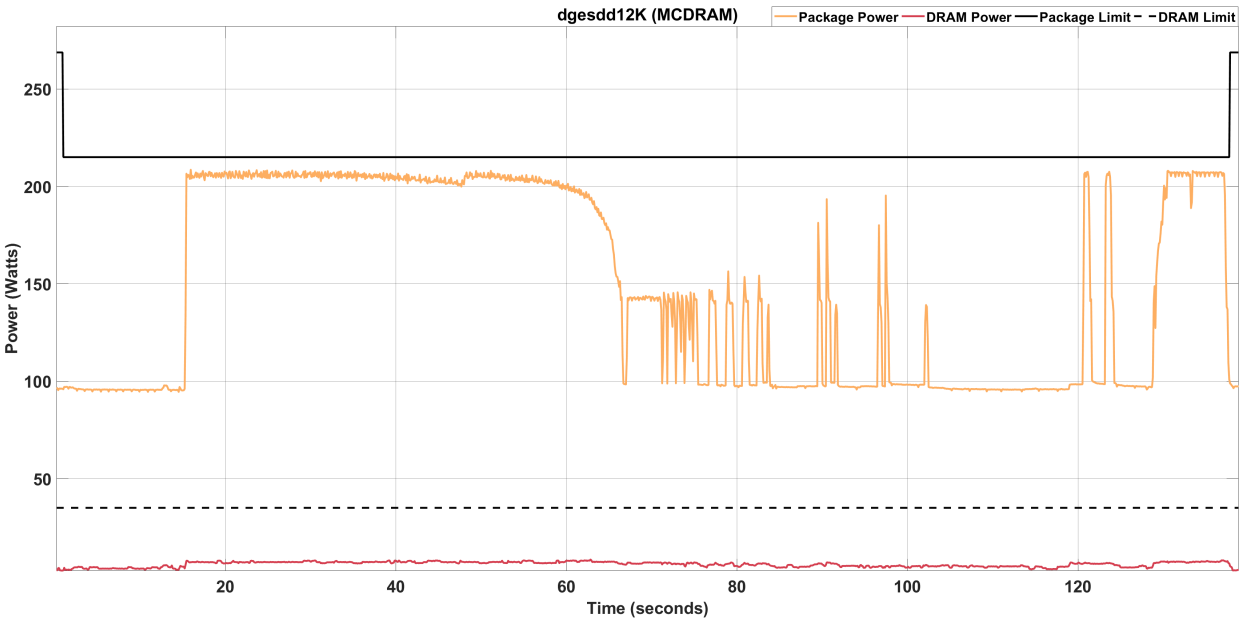

Figure 4.16: Power profile of a 12K DGESDD executing on MCDRAM under a 215 Watt powercap.

An important conclusion to draw from the analysis of power profiles from each computational category is the sheer complexity associated with power consumption patterns of complicated applications. It seems that using only power consumption as a way of characterizing the computational intensity of application may not be sufficient. Considering performance profiling options that can augment the use of power consumption as an identifier is key to attaining a more reliable means of categorization. Turning to the aid of hardware performance counters is a logical next step in the pursuit of energy savings for high-performance applications.

Chapter 5

Power Management Moving Forward

5.1 Understanding Power Consumption

The previous sections revealed how to characterize simple applications as memory- or compute-bound by analyzing their DRAM power consumption when running on DDR4 memory. Applications as complex as DGESVD and DGESDD it becomes beneficial to analyze power consumption in conjunction with performance hardware counters. An example of a potentially useful combination of hardware counters is the ratio of the number of unhalted core cycles and unhalted reference cycles. Unhalted core cycles represent the number of cycles that the core was not in a halted state based on the current processor frequency. Unhalted reference cycles represent the number of cycles spent in an unhalted state based on the nomial processor frequency. The ratio of unhalted core cycles to unhalted reference cycles can serve as an indicator of application performance under a software-enforced power limit. The ratio should be one since if the cores are actually operating at the nominal frequency. It follows that this metric could serve as an indicator of performance under a power limit.

5.2 Analyzing with Hardware Counters

The power profiling framework described logs power metrics, but it is also capable of collecting hardware metrics at the same time. This approach allows for analysis of hardware counters that are synchronized with the power consumption measurements. This allows for

drawing conclusions about the effect that power limits have on the underlying hardware because the hardware counters are a direct reflection on that.

5.2.1 Compute-bound Implications

Enforcing power limits is done via DVFS, which directly affects processor frequency, so this should be reflected in hardware metrics related to frequency. The counters for unhalted core and reference cycles are a reflection of the frequency that a processor is currently operating at. Unhalted core cycles are a reflection of the frequency that a processor is currently operating at, and unhalted reference cycles represent how the processor would be behaving at nominal processor frequency. By analyzing the ratio of the two, it can be deduced how far away a program is deviating from performance under nominal frequency settings.

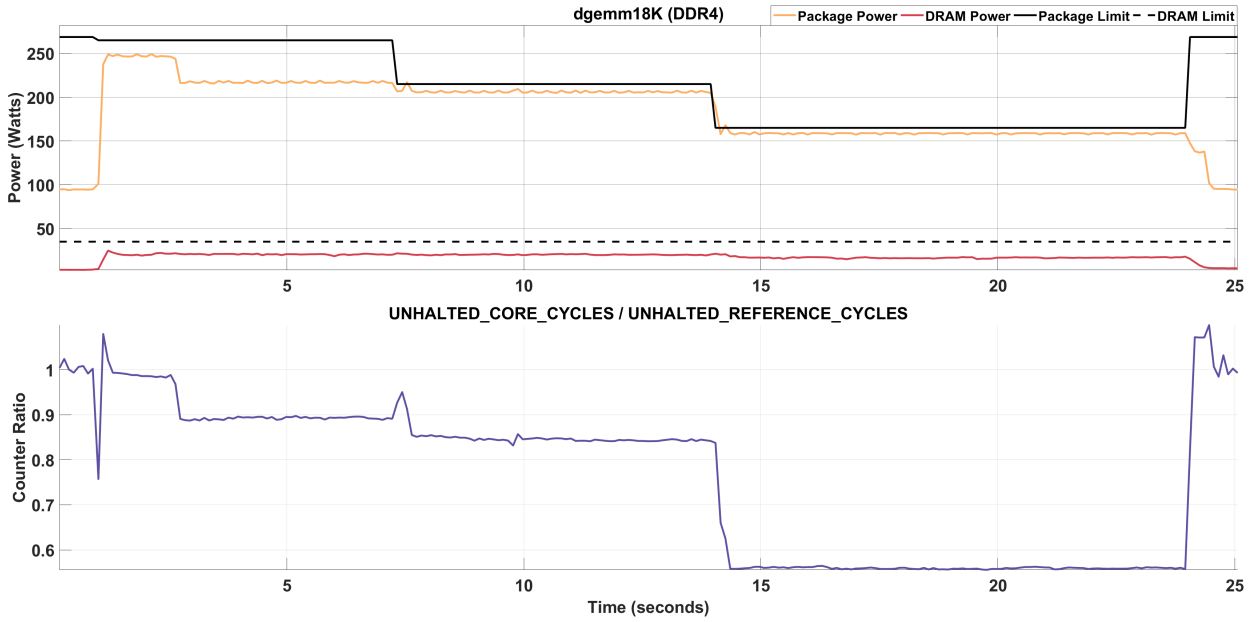

Figure 5.1: Power profile of an 18K DGEMM executing on DDR4 under multiple powercap limits alongside hardware counters.

Figure 5.1 depicts a profile of a multi-limit DGEMM executed on DDR4, displaying both power consumption and hardware counter measurements. The ratio of the counters shows a very close match to the trend exhibited by the package power consumption. When a new power limit is introduced, there is an immediate drop in the counter ratio just as there is in the power consumption measurements. The same initial spike in the counter ratio can be

seen in the package power consumption line. That spike represents the time the program was able to execute about the TDP limits before being brought down by the hardware. It seems that this counter ratio is a direct reflection on the current processor frequency, which itself is a direct reflection of the enforced power limit. Note that at the start of the program, the ratio was at one, and subsequently dropped as the package power limit was decreased. The drop in the ratio translates into a loss in performance as can be seen by examining the execution times of each DGEMM execution. The drop in performance seems to be proportional to the drop in the counter ratio indicating that the further the ratio is below one, the further the drop in performance.

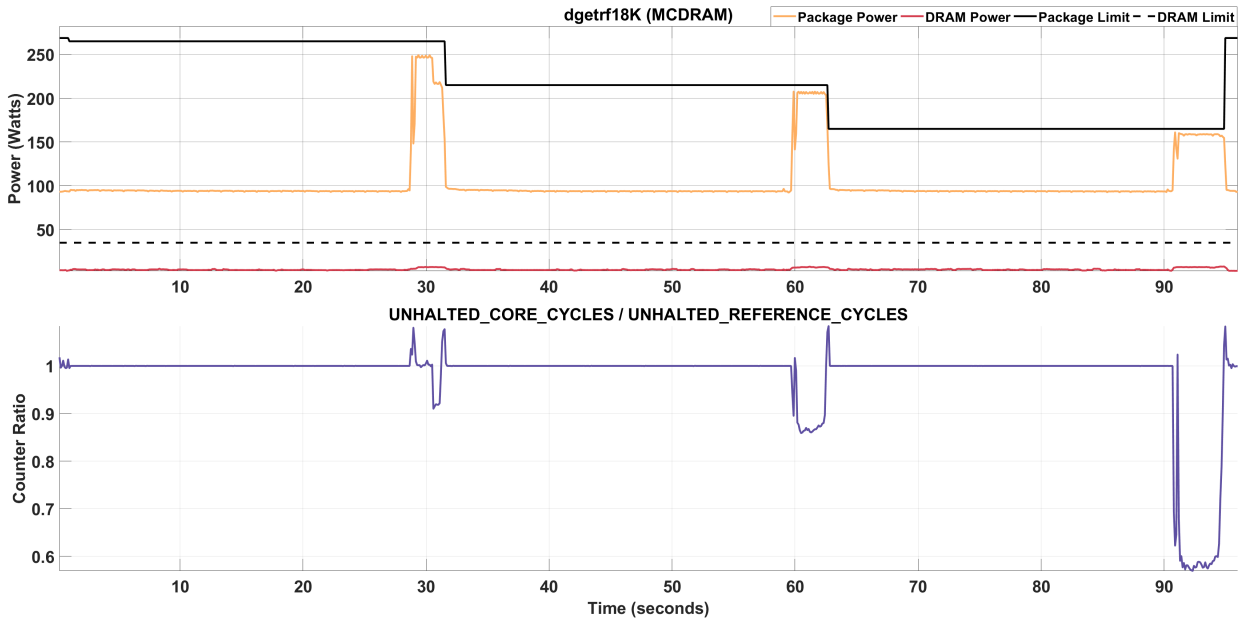

Figure 5.2: Power profile of an 18K DGETRF executing on MCDRAM under multiple powercap limits alongside hardware counters.

This result can also be related back to the concept of computational intensity categories. A profile of the application DGETRF is shown in Figure 5.2, which contains the same power consumption and hardware counter measurements as the DGEMM profile. Note that the important portion of the profile is at the end when the actual computations are happening. For an unknown reason, this particular application from Intel MKL has a long initial segment of power consumption close to the system idle power limit of 90 Watts. Nonetheless, DGETRF is also a compute-bound code and the same drop in counter ratio that was observed in the DGEMM profile is also present in this one. This is in line with

previous conclusions regarding applications of the same computational intensity category should behave similarly under power limits.

5.2.2 Memory-bound Implications

Memory-bound applications show the same drop in the ratio of unhalted core cycles to unhalted reference cycles, however; the point at which this occurs differs from compute-bound applications. Compute-bound codes show an immediate drop in performance for even small decreases of the package power limit. This is not necessarily the case for all memory-bound codes as can be observed in Figure 5.3. The DGEMV profile shows that the application only suffers a small drop in counter ratio at the package power limit of 155 Watts. In the case of DGEMM the ratio was under 0.6 at the same limit. Clearly there is different behavior in the hardware counter ratio between compute- and memory-bound applications. Since DGEMV experiences only a small drop in the ratio, there is also a minute change in performance, which is not really noticeable. This is key, because it illuminates the notion of using a hardware counter ratio to determine if a power limit is valid for a given application. Since these ratios are computed at run-time, it makes it possible to find a real-time solution for anlyzing hardware counter ratios and make informed decisions about power limits. If both cases of memory- and compute-bound applications are handled, it can be possible to achieve energy saving using this approach.

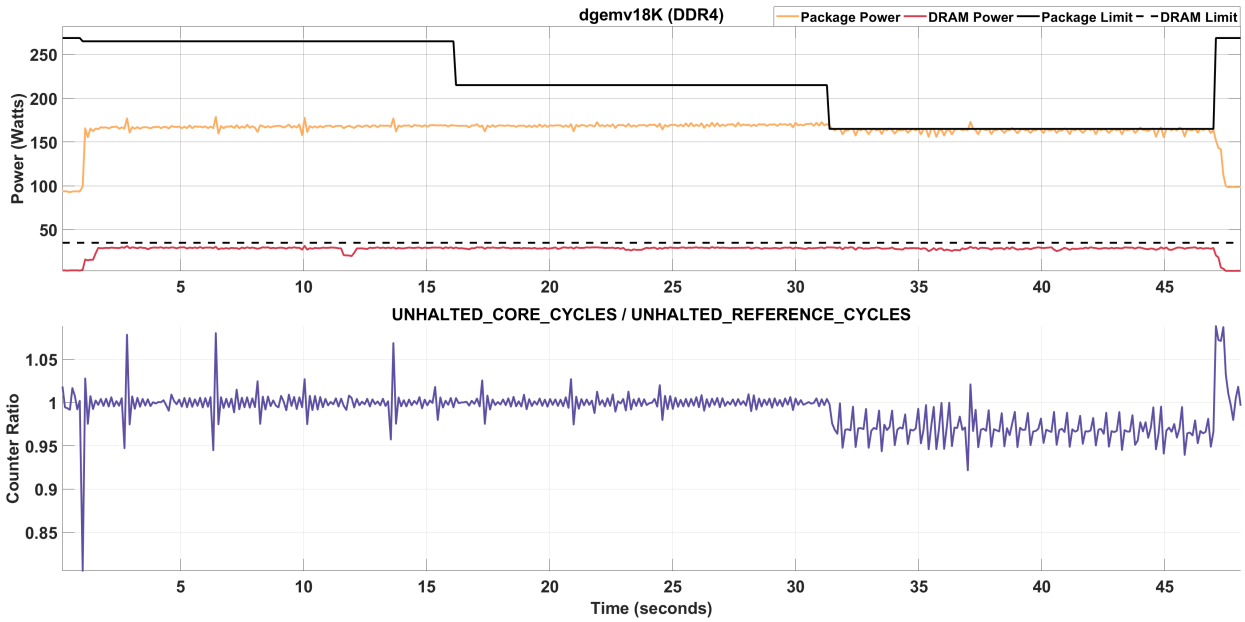

Figure 5.3: Power profile of an 18K DGEMV executing on DDR4 under multiple powercap limits alongside hardware counters.

Jacobi is another memory-bound application and a profile of a Jacobi execution is depicted in Figure 5.4. This run of Jacobi on MCDRAM shows similar behavior to that of the DGEMV execution. In fact, the package power limit of 215 Watts affects Jacobi power consumption, but had no effect on DGEMV power consumption at the same limit. At 215 Watts in the Jacobi profile, the ratio of counters drops, but it still remains above one. An interesting side effect from this particular limit is that it actually increases the performance of Jacobi. At the limit of 245 Watts the ratio was slightly above one, and bringing it right at one seems to have created an optimal performance setting. This leaves the door open for actually improving performance by power limiting which is a very interesting notion. Once again, the theme of applications of a specific computational category behaving in the same way under power limits is apparent in this result. Memory-bound applications can benefit from lowering the counter ratio to one via power limiting, while compute-bound applications aim to attain a ratio of one from below it.

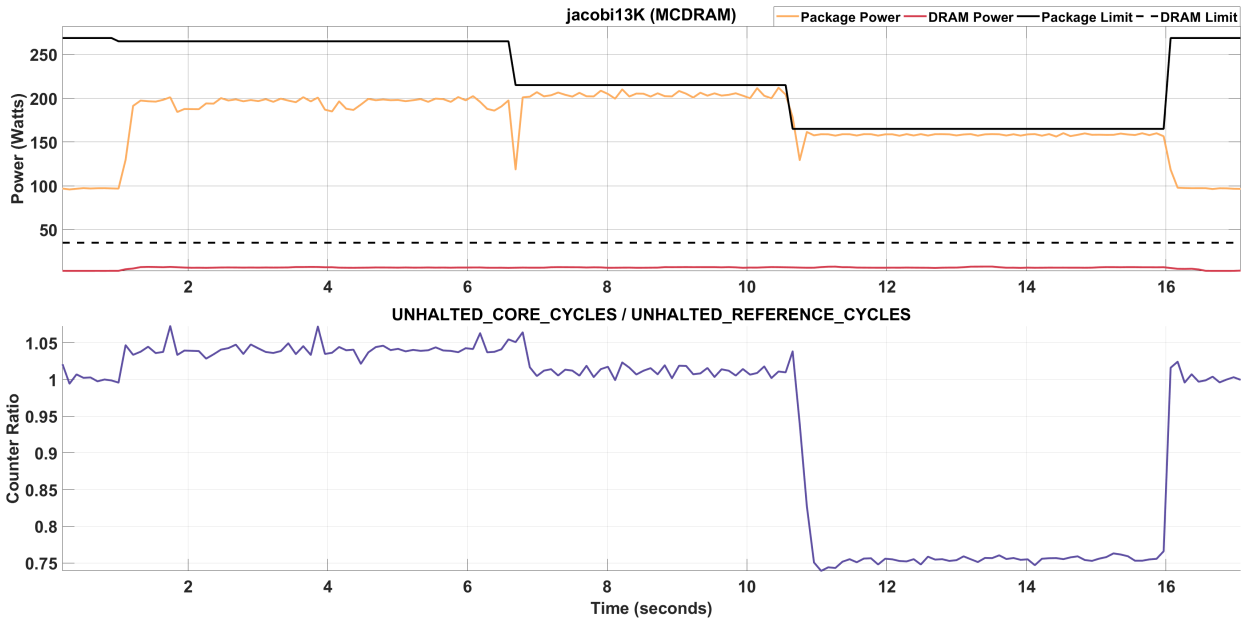

Figure 5.4: Power profile of a Jacobi executing on MCDRAM under multiple powercap limits alongside hardware counters.

5.2.3 Interleaved Implications

To further the point, a profile of the interleaved application DGESDD is displayed in Figure 5.5 run using MCDRAM. DGESDD has segments of both types of computational category and those segments exhibit the same characteristics as applications that are purely from a single category. Just as with Jacobi, the memory-bound section of DGESDD has a counter ratio that is at, or above, one at both package power limits of 245 Watts and 215 Watts. The drop below one only occurs when the limit is set to 155 Watts, and there is a subsequent degradation in performance that can be observed in the data.

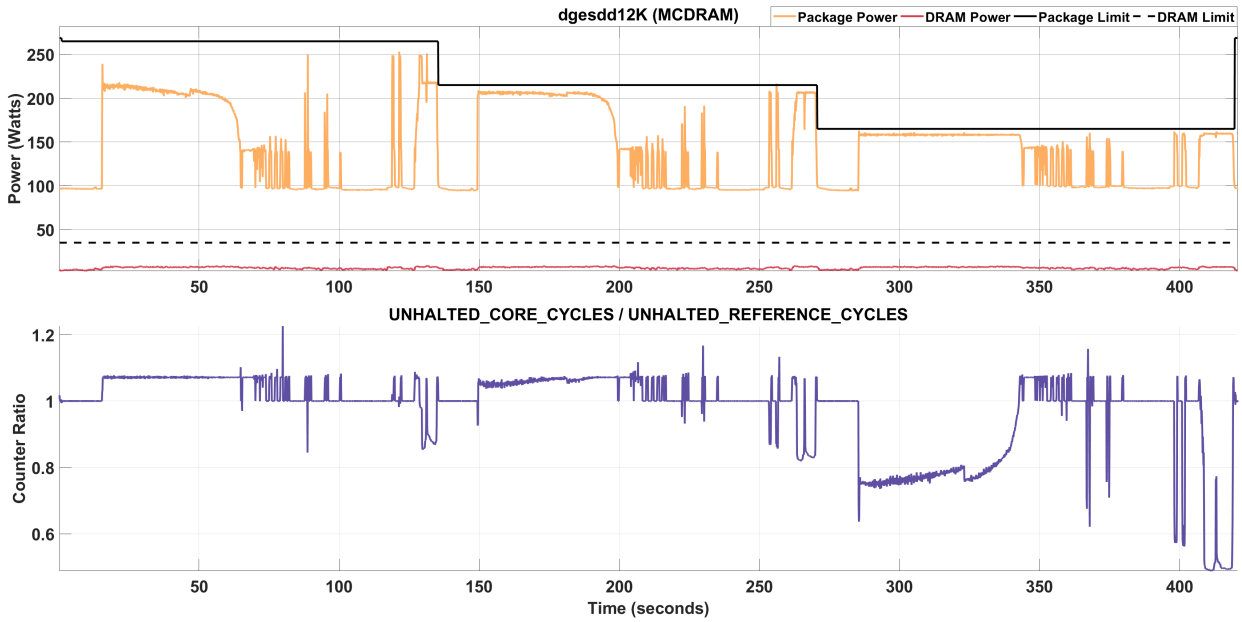

Figure 5.5: Power profile of an 12K DGESDD executing on MCDRAM under multiple powercap limits alongside hardware counters.

Figure 5.5 is another example of a profile of an interleaved application run on MCDRAM. DGESVD has a similar execution pattern to DGESDD because they both share the initial memory-bound segment followed by many compute-bound segments. Comparing those two can confirm if the trend is common among interleaved codes just as it is with the other two categories. Again the same behavior can be seen in that the counter ratio starts above one and only is brought below one at the package power limit of 155 Watts. Note that there is energy savings at the 215 Watts limit, because the power consumption is affected by the power limit but the ratio remains at or above one during the memory-bound segment. The result confirms the possibility that applying these power limits to the memory-bound segments can save energy without losing performance.

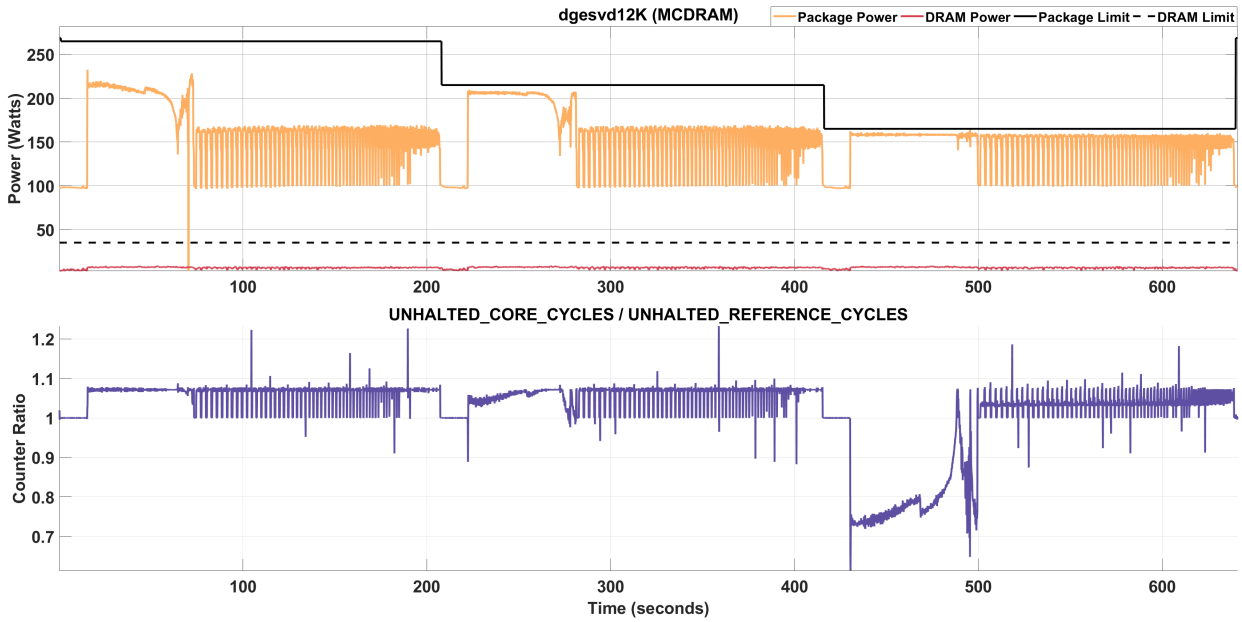

Figure 5.6: Power profile of an 12K DGESDD executing on MCDRAM under multiple powercap limits alongside hardware counters.

5.3 Moving Forward

It is now clear that applying limits once at the beginning of complex applications like this is not optimal because the way memory-bound segments experience power limits differs from that of compute-bound segments. Thus, an ideal situation would be to analyze this hardware counter ratio in real-time, and make informed decisions about an appropriate power limit. This data has shown that if such a framework could be assembled, it may be possible to analyze hardware metrics and enforce power limits on the fly in order to save energy. Future work in this area included fine-tuning parameter for a power modeling based on hardware counters. This work serves as a foundation for filtering out irrelevant counters and focusing on the ones who truly reflect power consumption.

Bibliography

[1] Haidar, A., Jagode, H., YarKhan, A., Vaccaro, P., Tomov, S., and Dongarra, J. (2017). Power-aware computing: Measurement, control, and performance analysis for intel xeon phi. In *2017 IEEE High Performance Extreme Computing Conference (HPEC'17), Best Paper Finalist*. IEEE, IEEE. 4

[2] Intel (2016a). *Intel 64 and IA-32 Architectures Software Developers Manual*, volume 3B of *System Programming Guide, Part 2*. Intel. 1

[3] Intel (2016b). An intro to mcdram (high bandwidth memory) on knights landing. https://software.intel.com/en-us/blogs/2016/01/20/an-intro-to-mcdram-high-bandwidth-memory-on-knights-landing. 10

[4] Intel (2017). Intel math kernel library (mkl). https://software.intel.com/en-us/mkl. 8

[5] Lee, J., Nam, B.-G., and Yoo, H.-J. (2007). Dynamic voltage and frequency scaling (dvfs) scheme for multi-domains power management. In *2007 IEEE Asian Solid-State Circuits Conference*, pages 360–363. 2

[6] PAPI (2017). The performance api (papi). http://icl.utk.edu/papi/. 4

Vita

Philip Vaccaro was born in Nashville, Tennessee, in the year 1991. He graduated with a Bachelors of Science in Computer Science from the University of Tennessee in 2014. During his undergraduate career he completed a Software Engineer Internship spanning over a year and a half. Upon completed of his undergraduate academic career, he spent a year in Princeton, New Jersey, working as a Software Engineer for a publicly traded company. In 2015 Philip returned to the University of Tennessee to pursue his graduate studies at the Innovative Computing Lab.