Performance-aware Energy Management for Next Generation Processor and Cloud Computing Systems

Thesis

Submitted in partial fulfillment
of the requirements for the degree of
DOCTOR OF PHILOSOPHY

by

Diyanesh Babu C. V.
ID No. **2013PHXF0507H**

Under the supervision of

Prof. M.B. Srinivas
&
Under the Co-supervision of

Prof. Subhendu Kumar Sahoo

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI
2024

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI

CERTIFICATE

This is to certify that the thesis entitled, **"Performance Aware Energy Management for Next Generation Processor and Cloud Computing Systems"** and submitted by **DIYANESH BABU C. V.** ID No. **2013PHXF0507H** in partial fulfillment of the requirements of Ph.D. of the Institute embodies original work done by him under my supervision.

[illegible]

Signature of the Supervisor:
Name in capital letters: M. B. SRINIVAS
Designation: **PROFESSOR**

Date: May 10, 2024

Subhendu kumar Sahoo

Signature of the Co-supervisor:
Name in capital letters: SUBHENDU KUMAR SAHOO
Designation: **PROFESSOR**

Date: May 10, 2024

"The difference between impossible and possible lies in a person's determination."

Swami Vivekananda

"The mind that opens up to a new idea never returns to its original size."

Albert Einstein

"Innovation distinguishes between a leader and a follower."

Steve Jobs

Acknowledgements

First and foremost, I would like to express sincere gratitude to my supervisor, Prof. Dr. M.B. Srinivas for his help and valuable guidance, without which this work would not have been accomplished. Whenever I have required support or have hit a barrier in my research, his doors have always been open for advice. His unwavering support throughout the journey has been a constant source of encouragement. I would also like to thank my Co-supervisor, Prof. Dr. Subhendu Kumar Sahoo, Doctoral Advisory Committee (DAC), Prof. Dr. Subhendu Kumar Sahoo and Prof. Dr. Tathagata Ray, for their timely feedback and insightful comments. I also take this opportunity to thank Head of the Department of Electrical Engineering, Associate Prof. Dr. Alivelu Manga Parimi for the support and help extended to me.

I would like to express sincere thanks to co-authors Dr. Karthick Rajamani from IBM Cloud Infrastructure Operations group and Dr. Venkata Kalyan Tavva from IIT Ropar for their perseverance to work with me from the start and spending personal time. Life outside the campus is an important ingredient of PhD life and in this respect, I express my gratitude to my friend, colleague and PhD mate, Saravanan Sethuraman for his great support right from day one and during tough times.

Finally, my family has been my pillar of strength during these years. My deepest gratitude goes out to my father, mother, wife, my daughter and son, who have stood by me through thick and thin. I would like to express special thanks to my lovely wife Aswini, for her care, love and continued support has made this journey memorable.

Abstract

Cloud computing has become an indispensable component of modern computing infrastructure, offering flexible and scalable services to users. However, the rapid growth of cloud computing has also led to concerns regarding energy consumption and carbon dioxide (CO2) emissions. The energy consumption of cloud data centers has a significant impact on CO2 emissions, as majority of electricity generation still relies on fossil fuels. According to a report by the International Energy Agency, data centers consumed around 200 TWh of electricity in 2018, accounting for around 1% of global electricity consumption [39]. The report also notes that data center energy consumption is expected to double by 2030 if no action is taken to improve energy efficiency. According to a study by A. Vakilinia and R. Buyya, cloud data centers are responsible for a significant portion of global CO2 emissions, with emissions expected to reach 3.2 gigatons by 2025 [40]. These trends highlight the need for energy-efficient cloud data center design and operation to mitigate the impact on the environment. In this thesis, we consider methodologies to address CPU and memory energy management challenges in cloud computing environments.

In first part of the thesis related to processor energy management, we present a highly accurate performance estimation methodology that accounts for architecture slack in workloads. Our work leverages the advanced instrumentation available in POWER8 processor that monitors core pipeline activity in relation to off-core memory accesses to build metrics for architecture slack characterization for workloads. Using these metrics, we construct a workload classifier that classifies workloads as core-bound and memory-bound and propose a performance prediction model for change in processor frequency for each class of workload - cPerf and mPerf, respectively. We evaluate these models with SPECCPU and PARSEC benchmark suites on a POWER8 based OpenPOWER system. We observe that the predicted performance with our models have high accuracy (97%) for both CPU and memory intensive benchmarks. We validate that the classifier is suitable to accurately classify phase of workloads during execution intervals. We propose an algorithm that uses classifier for phase classification and prediction

models for performance estimation at runtime. We leverage this algorithm and evaluate the execution time impacts of CPU and memory classified benchmarks. Overall, our methods based on architecture slack as key metric can be adopted by newer DVFS algorithms for phase classification and performance estimation at runtime, with a very high accuracy.

Memory subsystems in cloud computing are also a significant contributor to energy consumption. A key research gap is the need for more efficient use of DRAM. DRAM is a key component of cloud computing systems, but it also consumes a significant amount of energy.

In second part of the thesis related to memory energy management, we propose a new power mode as Voltage Reduced Self-Refresh (VRSR), which is basically reduced DRAM voltage operation in self-refresh. Our simulation results show that there is a maximum of ~12.4% and an average of ~4% workload energy savings, with less than 0.7% performance loss across all benchmarks, for an aggressive voltage reduction of 150 mV. We perform a detailed study of reducing self-refresh energy by reducing the supply voltage. PARSEC benchmarks in Gem5 full-system mode are used to quantify the merit of self-refresh energy savings at reduced voltages for normal, reduced, and extended temperature ranges. The latency impacts of basic operations involved in self-refresh operation are evaluated using the 16 nm SPICE model. Possible limitations in extending the work to real hardware are also discussed. As a potential opportunity to motivate for future implementation, DRAM architectural changes, additional low power states and entry/exit flow to exercise reduced voltage operation in self-refresh mode are proposed.

Table of Contents

Table of Contents

List of Tables

List of Figures

Abbreviations

AVATAR - A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems

BL - Bit Line

CLARA - Circular Linked-List Auto and Self Refresh Architecture

CPU – Central Processing Unit

DDR3L - Double Data Rate 3 Low Voltage

DDR4 - Double Data Rate 4

DIMM - Dual In line Memory Module

DPD - Data Pattern Dependencies

DPD - Deep Power-Down

DRAM - Dynamic Random-Access Memory

DSR - Deep Self-Refresh

DVFS - Dynamic Voltage and Frequency Scaling

eDRAM - Embedded Dynamic Random Access Memory

JEDEC - Joint Electron Device Engineering Council

LPASR - Low Power Auto Self Refresh

LPDDR - Low Power Double Data Rate

LSR - Long latency Self-Refresh

MIPS - Millions of Instructions Per Second

MRPI - Memory Reads Per Instruction

MRS - Mode Register Set

PDNA - Active Power-Down

PDNP - Precharge Power-Down

PMU - Performance Monitoring Unit

PTM - Predictive Technology Model

RAIDR - Retention-Aware Intelligent DRAM Refresh

RAPID - Retention-aware intelligent DRAM refresh

RDIMM - Registered Dual In-Line Memory Module

SRAM - Static Random-Access Memory

SREF - Self-Refresh

TCSR - Temperature Compensated Self Refresh

tRAS – Active to Precharge Delay

tRFC - Refresh cycle time

VID - Voltage ID

VRT - Variable Retention Time

WL - Word Line

Dedicate this to my parents, wife, daughter,
and son

Chapter 1

Introduction

In this chapter, we provide a brief introduction about processor and memory energy management in the context of cloud computing domain. We discuss their significance in reducing energy consumption and optimizing performance of systems.

Processor energy management is a critical aspect of cloud computing. In a cloud computing environment, multiple virtual machines run on a single physical server, and managing the energy consumption of each processor is crucial to maintain the overall system's efficiency.

Processor energy management techniques can be classified into two categories: dynamic voltage and frequency scaling (DVFS) and workload consolidation. DVFS adjusts the voltage and frequency of the processor to match the workload's requirements, while workload consolidation consolidates multiple workloads onto a single processor. DVFS can significantly reduce energy consumption by lowering the processor's voltage and frequency when the workload is low and increasing them when the workload is high. However, it can also affect the system's performance if the processor's frequency is reduced too much. Workload consolidation, on the other hand, can reduce energy consumption by consolidating multiple workloads onto a single processor, but it can also increase the system's response time.

First part of work focus on how architecture slack of workloads can be exploited for accurate performance management using DVFS technique.

Memory energy management is a critical aspect of cloud computing systems, as memory accounts for a significant portion of a system's energy consumption. Dynamic random-access memory (DRAM) is the primary type of memory used in modern cloud computing systems, and DRAM self-refresh will be effective to maintain data integrity when the DRAM memory cells were not accessed. However, DRAM self-refresh also consumes a significant amount of energy and managing its energy consumption is crucial for reducing overall system energy consumption.

There are several primary techniques are commonly used to reduce DRAM self-refresh energy consumption in cloud computing systems, including:

Power Gating: Power gating is a technique that shuts off power to unused portions of the DRAM chip. This technique can significantly reduce DRAM self-refresh energy consumption by reducing power consumption in idle memory areas.

Temperature-aware Self-refresh: Temperature-aware self-refresh is a technique that adjusts the self-refresh rate of the DRAM based on the memory temperature. This technique can reduce DRAM self-refresh energy consumption by reducing the number of self-refresh operations needed at lower temperatures.

Adaptive Self-refresh: Adaptive self-refresh is a technique that adjusts the self-refresh rate of the DRAM based on the memory's usage patterns. This technique can reduce DRAM self-refresh energy consumption by reducing the number of unnecessary self-refresh operations.

Data Compression: Data compression is a technique that reduces the amount of data stored in memory by compressing it. This technique can reduce DRAM self-refresh energy consumption by reducing the amount of data that needs to be refreshed.

Second part of work focus on achieving self-refresh energy savings at reduced DRAM voltages.

1.1 Processor Energy Management

In recent years, power and thermal issues have become primary design constraints for high-performance system designs. Restrictions arise from increased transistor densities and clock speeds, hindering the potential performance improvements and escalating expenses in development, acquisition, and operations. Consequently, there is a growing emphasis on effectively managing energy consumption and heat dissipation in high-performance computer systems [9,20,28].

Energy and thermal management are important for many reasons. First, to avoid employing performance crippling conservative designs to ensure that power and thermal caps are never exceeded, high-end systems employ active power and thermal management. Increasingly the cost of providing the power and cooling infrastructure for a datacenter or supercomputer approaches or exceeds the cost of the machines themselves. In many locations, electric utilities are unwilling to provide the additional power needed to add new machines to datacenters. Organizations also face external pressures to become “greener”. In the United States, both the Environmental Protection Agency and Department of Energy are issuing guidelines for energy-efficient systems and data centers [35,36]. An increasing number of customers insists that computers be energy-efficient [38], while still expecting performance improvements.

As the goal is to balance high performance with energy efficiency, power management becomes a strategic approach that involves leveraging opportunities to operate system components at lower power states without compromising performance. "Slack" refers to any characteristic that enables a system to operate a certain portion of itself at a lower power state while still meeting predefined performance objectives.

There are three forms of slack present in server computing environments: workload slack, user-demand slack, and architecture slack.

- **Workload slack** occurs when the workload is waiting on I/O or CPU does not have instructions to execute. It can be readily measured by monitoring the number of cycles in a given interval that the core is not yielded by the workload or operating system.
- **User-demand slack** occurs when systems can be run at less than full speed, but still achieve user performance requirements. It needs to be explicitly communicated

to the system (low-power mode, favor energy savings etc).

- **Architecture slack** arises when a workload's performance is bounded at least partially by a resource other than the processor clock frequency, e.g., by limited memory bandwidth. In this work, we focus on architecture slack present in server computing environments.

In a high-performance server system, it is common to find multiple processors that facilitate the simultaneous execution of multiple workloads. In both commercial and technical computing environments, one key objective is to maximize processor usage to speed up the execution of workloads for effective utilization of hardware resources. Another equally important key objective is to minimize energy usage, while maintaining high performance of workloads under execution.

Typically, a server class processor has many cores, and each core can support multiple hardware threads (multi-threading) for concurrent execution of workloads. Each process running on a hardware thread can have following different performance characteristics related to its current operating frequency: compute, memory, or moderate bound. For compute-bound processes, performance tracks linearly with processor core frequency and therefore limited by CPU speed. In the case of memory-bound processes, performance is unaffected by the frequency of the processor core. Instead, it is constrained by factors such as memory bandwidth and latency, which are not directly related to the speed of the CPU.

When a process runs on a hardware thread, it can execute core-bound or memory-intensive operations within a specific timeframe, referred to as a "phase." Core-bound operations can be completed within the core itself, without needing external resources. On the other hand, non-core-bound operations rely on external resources for completion. Typically, core-bound operations involve high-latency instructions that have a higher chance of causing processor pipeline delays. For example, retrieving data from on-chip L2 and L3 caches can have moderately long latencies of 8-60 cycles, while accessing DRAM can result in even longer latencies (e.g., 200+ cycles). To enhance performance during such scenarios, instruction-level parallelism (ILP) allows overlapping execution of multiple instructions using the same circuitry. However, there are cases where an instruction in the pipeline depends on the completion of a previous instruction, creating dependencies. Due to this, the core must wait until producer instruction is completed and this leads to idle cycles or stalls. There can be many such idle scenarios in modern pipelines that support simultaneous multi-threading (SMT). SMT refers to the capability of a single physical processor core to execute instructions from multiple hardware thread contexts concurrently. This allows the processor core to read and execute instructions in parallel, enabling scheduling of multiple applications simultaneously on the same core. In one situation, if a core is waiting for the outcome of memory access requests from outside the core and is not executing any operations from other threads, it is considered idle. In another scenario, if a core is waiting for the result of memory access requests from outside the core and not completing any execution threads during a cycle, the processor core is deemed idle and non-operational. When a core is both idle and waiting for off-core memory access, it indicates that there is available capacity or slack within the computer system. The presence of slack results in idle cycles, and if the processor is already operating at its maximum frequency, these cycles are essentially lost. These unused cycles are known as the "architectural slack" of the processor. To optimize performance, frequency reductions can be applied during these cycles without impacting overall performance.

1.2 Memory Energy Management

Data centers rank among the largest consumers of electrical power, accounting for around 200 terawatt-hours (TWh) of electricity consumption, which is nearly 1% of the global electricity demand. This substantial energy usage contributes to approximately 0.3% of global CO2 emissions [85]. Be it on-premise or in the cloud, demand for servers has been skyrocketing due to ever growing computing needs and big data explosion. However, around 30 percent of servers are either underutilized or completely idle, as per previous research performed by the uptime institute [86].

Currently, the CPU stands as the most power-intensive element within a server, while memory holds the position as the server's second most significant consumer of power [87].

In a server, main memory capacity and bandwidth requirements continue to grow, year over year. Modern data intensive applications from emerging areas like cloud computing, artificial intelligence, machine learning, augmented reality, geonomics and accelerated computing have clearly necessitated high capacity and low latency memory for superior performance. Due to its distinct benefits of low latency, high density, and well-established fabrication process, DRAM technology remains the favored option for main memory. DRAM energy usage makes up approximately 46% of the total energy consumption in a system [57, 88]. Specifically, in a multicore processor executing a collection of parallel applications with high memory demands, the DRAM core alone consumes roughly 20% of the overall system energy [50].

A DRAM cell consists of an access transistor and a capacitor, where the capacitor serves as a storage unit for data in the form of electrical charge. However, it is important to note that the capacitor gradually loses charge over time due to leakage. Refresh is an essential operation to ensure data retention in DRAM memories. However, it adversely impacts power dissipation and performance. With growing DRAM chip density, refresh power consumption has become significant portion of the total device energy [41].

Industry standard DRAM devices support auto-refresh and self-refresh modes to perform the refresh operation, during active and idle scenarios respectively. Memory controller must issue refresh commands periodically interleaved with the core's read & write data to the DRAM device and it is referred to as auto-refresh.

Once the read and write queues of the controller remain empty for a specific duration, it instructs the DRAM to transition into a low power mode. Self-refresh is an energy-saving mode that can be achieved without compromising the integrity of the data, making it the most power-efficient state. In this mode, the DRAM device performs refresh operation using an inbuilt timer, whereas DLL, clocks and IO pins are all turned off to save the background power. This mode can be exercised to achieve good energy savings during moderate or long idle phases of workloads.

With the growing size of DRAM devices, the power impact of self-refresh becomes increasingly significant. A review of literature reveals that in DDR4x devices (as depicted in Figure 1.1), when memory size doubles, such as from 4 Gb to 8 Gb, 8 Gb to 16 Gb, and 16 Gb to 32 Gb, there is a corresponding increase in self-refresh current (IDD6) of 50%, 89%, and 118% respectively [42], [48-50].

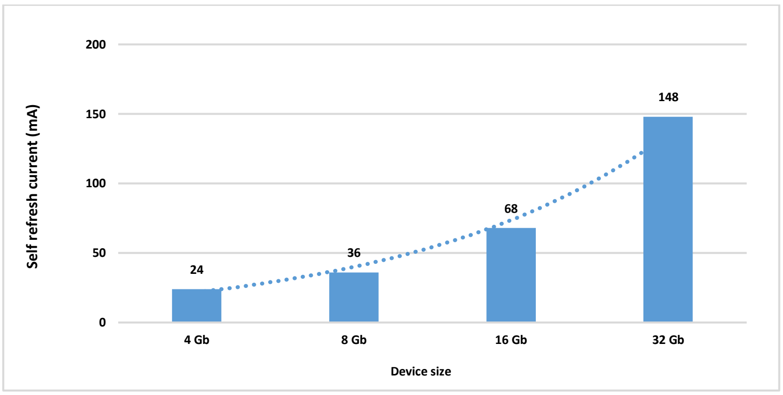


Figure 1.1: Self-refresh current (IDD6) increase trend in DDR4x devices

Extensive research has been conducted to minimize power consumption during refresh operations. Chang et al. [43] conducted a thorough investigation focusing on DDR3L DRAMs. Their study primarily involved characterizing the behavior of the DRAM chip when operated at reduced voltages below the nominal value. They examined the impact of low DRAM supply voltages of 1.2 V and 1.15 V, compared to the nominal value of 1.35 V, across various retention times (64 ms, 128 ms, 256 ms, 512 ms, 1024 ms, 1536 ms, and 2048 ms) on DDR3L DIMMs at temperatures of 20°C and 70°C. The study revealed that no weak cells were observed when the supply voltage was reduced up to a retention time of 512 ms, which is eight times the standard refresh interval of 64 ms. Consequently, the researchers concluded that at these temperatures, a reduction in supply voltage does not necessitate any modifications to the standard refresh interval. Furthermore, their work demonstrated that bit errors resulting from reduced voltage operation could be mitigated by increasing the latency of row activation (tRAS), restoration (tRCD), and precharge (tRP) operations.

Pardeik et al. have proposed to minimize the refresh power consumption by increasing the refresh rate and reducing the DRAM supply voltage during long idle scenarios [44]. Their work involved power evaluation of 16 GB & 32 GB DDR4 RDIMM modules at different refresh rates 2x, 4x, 8x and multiple voltages 1.25 V, 1.2 V, 1.15 V, 1.1 V, 1.05 V, 1 V. Their characterization results revealed a key finding that, on both 16 GB and 32 GB DIMMs, the voltage reduction of 200 mV, 1.25 V nominal to 1.05 V yielded significant power savings, despite of increased refresh rates. They observed 30%, 28%, 26%, 24% power savings for different refresh rates 7.8(1x), 3.9(2x), 1.94(3x), 0.96(4x) respectively, without any data integrity errors.

Byoungchan et al. [60] conducted a study where they observed that DRAM cells in the self-refresh mode operate in two distinct modes: static (idle) and dynamic (refreshing), and the transition between these modes follows a predictable pattern. They proposed a novel approach to optimize the leakage current of DRAM cells by aligning the word-line and body voltage levels with the cell's state. Their objective was to enhance the power efficiency of DRAM, leading to the introduction of two new self-refresh modes: Enhanced Self-Refresh (ESR) and Long Latency Self-Refresh (LSR). Through simulations, they demonstrated that the retention time of DRAM cells improved by 2.42 times in the ESR mode and 3.58 times in the LSR mode. Leveraging the extended retention time, their approach involved applying a reduced refresh rate while maintaining improved leakage current. The ESR mode could directly replace the original self-refresh mode without requiring modifications to the memory controller. It employed a selective word-line bias technique, which necessitated two transistors per sub-array to independently control the voltage level of individual sub-arrays. On the other hand, the LSR mode represented a new power-saving mode with even higher efficiency than the ESR mode. However, due to its distinct exit latency compared to self-refresh, it required adjustments to the memory controller. In addition to selective word-line bias, the LSR mode employed selective body biasing to achieve further power savings.

Existing literature [43, 44] motivates us to investigate energy savings in self-refresh mode at lower DRAM voltages. We observe that the basic idea in [43] explores energy saving opportunities with auto-refresh during memory mainline read & write operation, while our work is orthogonal that focus on self-refresh mode exercised during long idle times. In contrast to [44], we further extend our studies to lower voltage operation at reduced and increased self-refresh rates associated with wider temperature ranges, as supported by DDR standards. For lowering DRAM voltage during self-refresh, we consider the implementation of varying array voltage (Varray) in step sizes without modifying standard refresh rates, which is orthogonal to work [60] that uses selective body bios and selective word-line bios controls to attain the increased retention time thereby to lower the refresh rates for energy savings. Reducing voltage during self-refresh needs additional latency cycles of tRAS and tRP parameters for error free operation [43]. We quantify the performance impact seen by workloads with such increased latency cycles.

In our research, we investigate the necessary modifications to both the DRAM and controller to implement the proposed energy-saving feature. We provide a comprehensive description of the interaction flow between the controller and DRAM, focusing on the newly suggested architectural changes. To maximize the effectiveness of our approach, it can be combined with the findings of previous studies, such as [43] and [60]. As a result, we introduce a new low power mode for DRAM called "voltage reduced self-refresh operation (VRSR)."

1.3 Background

1.3.1 Processor – Architecture slack exploitation for performance management

While previous research [5,10,18,32,33] extensively explores the utilization of slack and its applications, we have identified a gap in leveraging architecture slack for performance management. Fields et al. [10] introduce the concept of slack and its exploitation in the design of a processor with heterogeneous pipeline implementations. Our work complements theirs by developing methods for highly accurate performance prediction. Liang et al. [18] propose a cache-miss based prediction model for energy and performance degradation, while Spiliopoulos et al. [32] present a slack time-based model. However, these approaches do not consider architecture slack. In contrast, Hari et al. [5] propose a method to exploit timing slack in embedded applications. Their approach relies on the execution of an application that avoids the processor's static critical paths, enabling energy savings by scaling down the processor's voltage while maintaining the same frequency until the longest active paths meet the timing constraints. However, this approach requires profiling applications prior to execution, resulting in significant overhead. Similarly, Sharanyan et al. [33] propose a multicore CPU scheduler that combines traffic sources, latency tolerance, and computational resource requirements. They utilize "CPU stall cycles on cache misses" as a key metric to co-locate threads on the same socket or physical core for improved parallel efficiency. Our work aims to bridge the gap by focusing on exploiting architectural slack for performance management, providing a novel approach that complements existing research in this area.

We refer “workload classification” as detecting whether the workload is compute-bound or memory-bound. In our current work, we propose a fixed interval-based online “phase classification” scheme that uses microarchitecture-dependent metrics (instruction throughput and not-busy cycles) obtained from hardware counters. “Performance estimation” is a measure of frequency-performance relationship at target frequency with respect to its current frequency-performance setting, for a given workload under execution.

1.3.2 Memory – Voltage Reduced Self-Refresh operation

JEDEC (Joint Electron Device Engineering Council) supports many features to optimize self-refresh power (Table 1.1). Low-Power Auto Self Refresh (LPASR) and Temperature Compensated Self Refresh (TCSR) are features that adjusts refresh rate depending on the ambient temperature in DDR4x and mobile LPDDRx devices respectively. The Partial Array Self Refresh (PASR) feature allows the controller to choose the specific portion of memory arrays that need to be refreshed during self-refresh in LPDDRx. By combining the TCSR and PASR features, even more significant power savings can be achieved [41].

Counter	Description	Technique
DDR4x	Low-Power Auto Self Refresh (LPASR) [42]	Refresh rate is adjusted based on temperature
Low Power DDRx (LPDDRx)	Temperature Compensated Self Refresh (TCSR) [47]	Refresh rate is adjusted based on temperature
LPDDRx	Partial Array Self Refresh (PASR) [47]	Refresh operation is limited to portion of the memory’s array where data is stored

Table 1.1: Power saving features in self-refresh mode

The JEDEC standard defines the Deep Power-Down (DPD) mode as the most extreme power-saving state, where the entire memory array in the device is shut down. In DPD mode, all internal voltage generators are halted, resulting in the loss of all data stored in the memory. This mode proves beneficial in mobile applications where continuous data retention in DRAM is not necessary for most of the time. Additionally, a technique called Deep Self-Refresh (DSR) exists, which combines the power-saving advantages of Partial Array Self Refresh (PASR) and DPD [51]. While PASR keeps the internal voltage generators of the DRAM in a low-power mode, DSR allows the unused banks of memory cells to enter the DPD state. It is important to note that although DPD offers even lower energy consumption in self-refresh mode compared to PASR, it is not included in the JEDEC standard. In our work, we focus on reducing DRAM energy consumption by lowering the DRAM supply voltage in the normal self-refresh mode, without the need for additional power-saving modes like DPD or DSR. Our work reported has the following key take aways:

(a) A survey of current techniques to optimize power in self-refresh mode is initially presented. The main cause for power consumption in self-refresh is also described. Further, voltage is proposed as a potential knob to minimize the power consumption. Some of the recent work [43, 44] on comprehensive study & characterization of DRAM data retention ability at lower voltages, wide temperature ranges and refresh rates are presented and discussed. The self-refresh energy scaling trends in modern devices [42] with respect to wider temperature ranges and currently supported adaptive refresh rate techniques are studied in detail. Two key aspects are considered in our study of self-refresh energy savings at lower voltages and different refresh rates. They are (a) Increasing DRAM row activation (tRAS) and precharge (tRP) latencies at reduced voltages to ensure data retention (b) Latency increase in tRAS and tRP parameters are quantified using open-source model, after adapting it for 16nm DDR4 technology and these new latency values are used in simulation for energy and performance study.

(b) To assess the energy-saving effects of self-refresh at reduced voltages, we employ a set of eight PARSEC benchmarks for evaluation purposes. The simulation and measurement setup to quantify the DRAM energy consumption are described in detail. Based on full-system simulations, a detailed study of self-refresh energy savings for a set of 6 reduced voltage points, from 25 mV through 150 mV with a step size of 25 mV, at normal (1x standard refresh), extended (2x refresh) and reduced (0.5x refresh) temperature ranges are performed.

(c) Increase of row refresh cycle time (tRFC) due to prolonged tRAS and tRP latencies at reduced voltage levels is discussed. This increased latency presents an overhead in exit latency of VRSR scheme and its impact on workload performance is analyzed.

(d) Subsequently, simulation results are presented and discussed to evaluate the potential energy savings across six reduced voltage points and the corresponding performance impact due to exit latency overhead. These discussions encompass all eight PARSEC benchmarks, providing insights into the effects of an aggressive voltage reduction of 150 mV.

(e) Some of the challenges and limitations in expanding our work to get detailed hardware measurements are discussed. DRAM architectural changes and additional power modes to exercise reduced voltage operation in self-refresh mode are proposed. Based on this proposal, the key open areas are presented to motivate researchers for future exploration with a goal to realize a full-scale solution.

1.4 Motivation

1.4.1 Processor – Performance Aware Energy Management

Dynamic Voltage and Frequency Scaling (DVFS) is a widely recognized and commonly utilized technique that enhances the energy efficiency of computing systems by dynamically adjusting the voltage and frequency based on the current utilization of processor cores. The primary objective of DVFS methodology is to develop a scheduling scheme for the usage of processor clock frequency-voltage settings over time, aiming to minimize processor power consumption while minimizing any performance degradation. To achieve this, a DVFS scheduling algorithm must determine the appropriate timing for adjusting the current frequency-voltage setting (scaling point) and identify the optimal new frequency-voltage setting (scaling factor).

In traditional approaches, metrics such as instruction retired and cache miss have been utilized for performance management [16, 21, 27]. Many existing DVFS algorithms rely on metrics like instructions retired or executed, assuming that an application's performance scales proportionally with the processor clock frequency. According to this assumption, halving the processor clock frequency would halve the computing system's performance. However, in reality, the execution time might only double in worst-case scenarios when the processor clock frequency is halved. Consequently, a DVFS scheduling algorithm based on this model might prioritize scheduling tasks at a faster processor clock frequency, potentially completing them well ahead of their deadlines (leading to a "race to halt" situation). Conversely, a slower processor clock frequency could be scheduled, meeting the performance deadline while consuming less power. Nevertheless, making decisions about "scaling factor" requires accurate runtime prediction of the operational relationship between frequency and performance. While cache miss is another common metric used to measure memory boundedness, it alone cannot guide frequency scaling decisions while accounting for the impact on performance. Hence, there is a clear need for a metric in DVFS that can evaluate the actual performance impact of frequency changes based on the runtime conditions.

To address this requirement, we conduct an evaluation of the instrumentation features available in various processor families to identify the performance monitoring unit (PMU) events that can effectively track the relationship between frequency and performance during runtime. Through our analysis, we have discovered that the POWER8 core [30] possesses a specific counter capable of monitoring the pipelined activity during off-core memory accesses, serving as a reliable indicator of architecture slack. In our work, we develop methods to exploit architecture slack-based counter in the POWER8 core, as this being the first core that offers such a capability. However, next generation POWER9 core also has this counter [3]. Therefore, for any related or future exploration, same methods and approaches can be extended to POWER9 and any other pipelined architectures (Intel, AMD processors) that offer such a capability for measurement of the architecture slack.

1.4.2 Memory – Performance Aware Energy Management

Figure 1.2a shows a DRAM cell arrangement and each cell, has a storage capacitor Cs which stores electrical charge. Data in a memory cell is stored as either the presence or absence of charge. To access the data, a transistor, represented as T, is used. The word line (WL) connects to the transistor's gate and is responsible for accessing the data. On the other hand, the bit line (BL) transfers data to or from the storage capacitor (Cs) and is connected to the transistor's drain. One plate of Cs is linked to the transistor's source, while the other electrode is biased at the cell plate voltage, denoted as Vp. The bit line is typically connected to multiple cells organized in a column, while the word line is connected to multiple cells arranged in a row. However, bit lines have relatively large parasitic capacitance Cbl since they connect all of the transistors in a column.

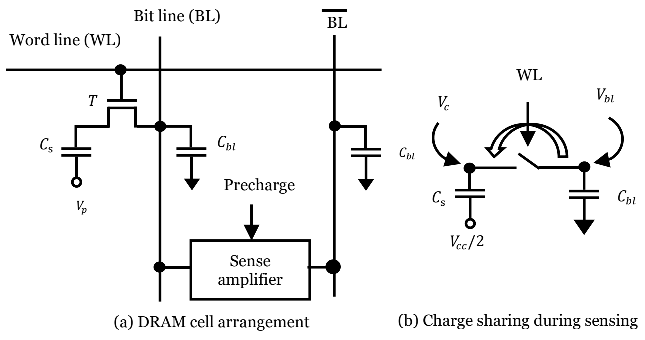


Figure 1.2: DRAM cell and charge access scheme

As illustrated in Figure 1.2b, when the word line is driven high, the charge redistribution occurs, and this creates a small difference between bit line voltages. A sense amplifier as shown in Figure 1.8a, which is associated with each bit line pair is used to detect the small voltage differences that occur during charge sharing. These bit lines must be pre charged with Vcc/2 before any read operation.

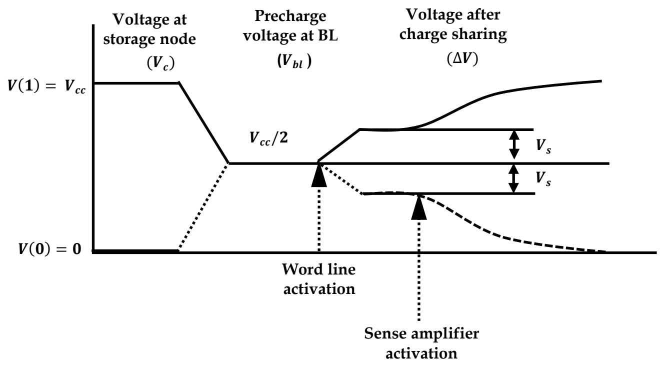


Figure 1.3: Storage node, bit line and charge sharing voltages

As illustrated in Figure 1.3, voltage at storage node (VC) can be either 1 or 0. During word line activation (access operation), when a DRAM cell is connected to bit line (BL), it loses majority of its charge from Vcc to (Vcc/2 + Vs) when storing ‘1’ or it is charged up from 0 to (Vcc/2 - Vs) when storing ‘0’, where Vs is given by equation (1.1). This is due to Cs charge sharing with large bit line capacitor Cbl. This generates a small “readout” ΔV which drives the input of the amplifier latch to one of the stable points 1 or 0, depending on the sign of ΔV (sense operation). Both access and sense operations are combinedly referred as row activation.

$$V_S = (V_{CC}/2) \,/\, (1 + \frac{C_{bl}}{C_S}) \qquad (1.1)$$

As the word line remains active, write back (restore operation) occurs, Cs is recharged back to Vcc level from (Vcc/2 + Vs), when storing '1'. Similarly, $\overline{(BL)}$ will also charge up from (Vcc/2 - Vs) to Vcc and Cs will be discharged from (Vcc/2 - Vs) to 0, when storing '0'. As a result, refresh power is mostly accounted for cells storing "1" due to recharging of cells from (Vcc/2 + Vs) to Vcc back and charging $\overline{(BL)}$ capacitance Cbl from (Vcc/2 - Vs) to Vcc level. For stored data '1' and '0', refresh power consumed P1 and P2, can be given as per equations (1.2) and (1.3) respectively.

$$P1 = 0.5 * (C_S) * \left(\frac{V_{cc}}{2} - V_S\right)^2 * f \tag{1.2}$$

$$P2 = 0.5 * (C_{bl}) * \left(\frac{V_{cc}}{2} + V_S\right)^2 * f \tag{1.3}$$

where f = refresh rate and total refresh power, P = P1 + P2. Equations (1.2) & (1.3) clearly show that voltage is a potential knob to exercise power reduction during refresh. Self-refresh power also increases proportionately with respect to the number of cells to be refreshed. Therefore, in higher density devices, it contributes to a significant fraction of the total energy consumption.

1.5 Organization of the Thesis

This thesis is organized as follows.

Chapter 1 gave a brief introduction about the importance of processor and memory energy management to reduce energy consumption:

- Specific to processor sub-system, given the dynamic workload nature in cloud computing environments, how workload architecture slack can be exploited to develop accurate models for performance management.
- Specific to memory sub-system, given the increasing DRAM energy consumption, the motivation towards reducing energy consumption by reducing voltage during self-refresh mode.

Chapter 2 summarizes literature survey on the following topics:

- Architecture slack exploitation for workload classification, phase classification and performance estimation.
- Studies to reduce DRAM refresh power consumption.

Chapter 3 covers the following:

- Experimental setup comprising hardware platform, SPECCPU 2006 and PARSEC benchmarks, tools to measure architecture slack and performance parameters of workloads for architecture slack exploitation
- Experimental setup including GEM5 full system simulator and PARSEC benchmarks for measuring DRAM energy consumption at reduced voltages

Chapter 4 covers the following:

- Methodology and evaluation of using architecture slack for workload classification of benchmarks, development of compute and memory regression models, phase classification at runtime, performance evaluation of benchmarks using a custom algorithm

Chapter 5 covers the following:

- Methodology of latency evaluation of timing parameters at reduced voltage DRAM operation using SPICE model, controller & DRAM architectural changes for practical implementation, Quantitative evaluation of performance overhead due to voltage reduced self-refresh

Chapter 6 presents the conclusions and the future work.

Chapter 2

Literature Survey

2.1 Workload Classification, Phase Classification and Performance Estimation

The exploration of architecture slack in server computing environments is a burgeoning field with significant implications for performance management and optimization. Our work is positioned at the forefront of this research, aiming to develop methods that leverage architecture slack for workload classification, phase classification, and performance estimation. This endeavor is underscored by the introduction of a novel metric that measures architecture slack through core instruction throughput and not-busy cycles. The utility of this metric is demonstrated through its application in CPU and memory microbenchmarks, SPECCPU 2006, and PARSEC benchmarks on a POWER8 server processor.

Prior work in the realm of workload classification has predominantly focused on distinguishing between compute-bound and memory-bound workloads. Basireddy et al. [2] utilized the Memory Reads Per Instruction (MRPI) metric on a heterogeneous multi-core platform to classify workloads and optimize energy consumption through voltage-frequency settings. Similarly, Robert et al. [27] employed hardware performance counters to gauge CPU load influenced by main memory accesses, guiding frequency scaling decisions. Chung-Hsing et al. [12] introduced a Dynamic Voltage Scaling (DVS) algorithm, "β-Adaptation," which adjusts core frequency based on the MIPS rate, highlighting the importance of instruction metrics over CPU cycles for workload requirement determination. These studies [2,12,27] underscore the significance of hardware metrics in workload classification but do not address architecture slack directly.

Phase classification [14,15,29,37] and its applications have been well studied in many prior works [6,25,34]. In the domain of phase classification, extensive research has been conducted to identify program phases and optimize performance. Srinivasan et al. [34] introduced an online program phase classification scheme utilizing a bottleneck type vector (BTV) to enhance performance per watt. Chesta et al. [6] proposed a phase detection approach using execution vectors (EVs) derived from hardware counters. Rodrigues et al. [25] demonstrated the efficacy of combining online phase classification with dynamic core morphing in asymmetric multicore processors. These studies [6,25,34] highlight the potential of microarchitecture-dependent metrics in phase classification but do not specifically explore the role of architecture slack.

Performance estimation, a critical aspect of performance management, has been well studied [1,7,17,24]. Shoaib et al. [1] developed a performance prediction model, DEP+BURST, for multithreaded managed applications, significantly reducing performance estimation error. Rajamani et al. [24] and Contreras and Martonosi [7] utilized hardware counters for power management, while Sang-Jeong et al. [17] employed regression analysis based on CPI and memory accesses for runtime performance projection. These studies [1,7,17,24] emphasize the importance of hardware counters in performance prediction but do not leverage architecture slack for this purpose.

Our work distinguishes itself by focusing on architecture slack as a novel metric for workload classification, phase classification, and performance estimation. By leveraging instruction throughput and not-busy cycles, we aim to provide a more nuanced understanding of server computing environments. This approach not only builds upon the foundational work of prior studies [2,12,27,34,6,25,1,7,17,24] but also introduces a new dimension to performance management strategies. Through the characterization of our metric and the development of performance estimation models, we contribute to the ongoing discourse on optimizing server performance, marking a significant advancement in the field.

2.2 Reducing DRAM Refresh Power Consumption

The quest to reduce DRAM refresh power consumption has been a focal point of numerous studies, each contributing unique insights and methodologies to address this challenge. The journey begins with the work of Byoungchan et al. [60], who introduced Enhanced Self-

Refresh (ESR) and Long Latency Self-Refresh (LSR) modes. These modes innovatively applied selective voltage levels to DRAM cell transistors based on their activity state, optimizing leakage current and significantly improving power efficiency. The ESR mode, requiring minimal modifications, and the LSR mode, which achieved greater power reduction at the cost of increased latency, laid the groundwork for subsequent research in DRAM power efficiency.

Building on these foundational concepts, RAIDR [61] exploited the variability in DRAM retention times [64] by grouping rows into bins with specific refresh rates, thereby enhancing system performance and reducing memory energy consumption. In a different study [62], a profiling mechanism is used to detect retention failures when the memory module enters self-refresh mode. This approach, however, faced challenges due to Variable Retention Time (VRT) failures, a problem that Qureshi et al. [65] aimed to address with AVATAR. AVATAR's dynamic adjustment of refresh periods sought to mitigate bit errors caused by VRT, though it could not guarantee the detection of all failing cells due to data pattern dependence. This limitation prompted the development of REAPER [66], which employed testing with multiple data patterns to identify failing cells more effectively.

Das et al. [67] proposed a mechanism that only fully refreshes DRAM cells when necessary, using low-latency partial refresh operations to maintain data integrity. This approach, validated through real workload memory traces, demonstrated a significant reduction in refresh performance overhead, marking a step forward in efficient DRAM management.

Further innovations came from Liu et al. [68] with the Flikker technique, which differentiated between critical and non-critical data for refresh rate adjustments, and the DIMMer approach [69], which powered off unused memory capacity to save energy. RAPID [70] and CLARA [71] introduced software and hardware solutions, respectively, to optimize refresh operations based on retention time variations, highlighting the complexity of addressing DRAM refresh power consumption.

Jung et al. [72] explored power-down mode policies in 3D-DRAMs, demonstrating significant energy savings through adaptive refresh periods based on temperature data. This study underscored the potential of temperature-aware refresh strategies in reducing DRAM power consumption.

The literature reveals a consistent theme: the challenge of profiling DRAM cells for retention times due to VRT and Data Pattern Dependencies (DPD), as highlighted by Liu et al. [63]. These phenomena, which cause cells to exhibit unpredictable retention states, complicate the development of efficient refresh strategies.

In response to these challenges, recent work [73-78] has introduced algorithmic changes, new DRAM refresh commands, and device-level refresh mechanisms. These innovations aim to improve performance and energy consumption by dynamically adjusting refresh operations to the retention characteristics of DRAM cells. For instance, the proposal to replace the NMOS transistor in 3T eDRAM with a relay [77] and the introduction of a retention-aware refresh technique called elaborate-refresh [78] represent significant advancements in DRAM technology.

In summary, the body of literature on reducing DRAM refresh power consumption highlights a progression from initial strategies focusing on selective voltage application and grouping based on retention times to more sophisticated hardware and software approaches that address the challenges posed by VRT and DPD. These studies collectively underscore the critical need for efficient DRAM refresh mechanisms that can adapt to the varying retention characteristics of cells, thereby reducing power consumption without compromising system performance or data reliability. This evolving landscape of DRAM refresh strategies forms the basis of the current problem statement, which seeks to develop more effective refresh techniques that dynamically adjust to retention time variability and dependencies.

Chapter 3

Experimental Setup

3.1 Experimental Setup for Architecture Slack Exploitation

We have used Amester tool [22,26] to collect run time traces of a workload under measurement, on a 1-socket POWER8 hardware platform. Figure 3.1 depicts the measurement setup. POWER8 platform used for this work comprises an 8-core processor and has 128 GB of main memory. A measurement system runs Amester (Automated Measurement of Systems for Energy and Temperature Reporting) tool and connects to host system for data collection. Amester is an out-of-band tool which can collect parameters of interest, without impacting the workload performance that runs on host system.

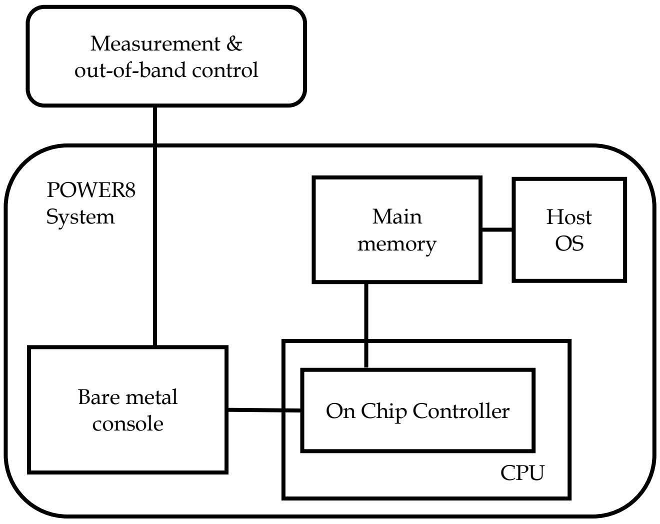


Figure 3.1: Measurement setup

POWER8 processor has a piece of hardware & associated firmware called the On-Chip Controller (OCC) [13]. The OCC (On-Chip Controller) is a separate processor integrated on the chip alongside the main POWER processor cores. It has its own dedicated 512K SRAM and can access main memory. The OCC firmware operates in a continuous loop with a duration of 250 microseconds, constantly gathering system data. It can collect detailed information on temperature, performance, power, and utilization, and has the ability to control processor frequency and memory bandwidth. We have leveraged the OCC's measurement capabilities to collect performance and not-busy stall traces for workload analysis, and its frequency control capability to adjust frequencies using a max-min performance threshold algorithm.

To determine the workload boundedness and leverage the Wb to mipsr correlation, we employed SPEC CPU2006 [31] and PARSEC [23] benchmarks in our study. The POWER8 architecture comprises 12 cores, and each core has the capability to handle eight hardware threads concurrently (SMT8).

Our experimental setting was a single core configured in SMT8 mode. Our goal was to study architecture slack with diverse set of both serial and parallel benchmarks. We identified SPECCPU 2006 as one workload set, as it offered a diverse set of serial benchmarks. We found PARSEC as another workload set for multi-threaded workloads, as their programs have been parallelized to take advantage of multiprocessor computers with shared memory. Together, these benchmarks exhibit a variety of performance sensitivities to changes in CPU frequency, which was an essential aspect for our investigation. We acknowledge that SPECCPU underwent revisions in 2017, introducing enhanced features, improved applications, multi-threading options for select applications, and an optional power consumption measurement metric [19]. However, since our study and results are not significantly impacted by these features, we opted to continue using SPECCPU 2006.

3.2 Experimental Setup for Study of Self-Refresh Energy Savings

We utilized a full system model based on GEM5 [52] to simulate a dual-core X86 ISA processor model, as presented in Table 3.1. The simulated DDR4 subsystem is connected to the controller via a single DDR4-2400 64-bit channel. This subsystem comprises 16 DRAMs, with 8 per rank, and each DRAM has an 8-bit interface (×8). Timings and key current values are based on the Micron DDR4-2400 8 Gbit datasheet (Micron MT40A2G4) as presented in

Table 3.2. To calculate the energy components, we employed the timings and key current values based on the Micron DDR4-2400 8 Gbit datasheet (Micron MT40A2G4) [42], which were utilized by the DRAMPower tool [53, 54] integrated within gem5. During simulation, the commands and timestamps are provided to DRAMPower at runtime. The DRAM subsystem consists of 16 banks, and the buffer that holds incoming requests for all banks is divided into separate read and write queues. The controller reorders the requests, and its scheduling policy follows the principle of First Ready - First Come First Served (FR-FCFS). The page policy employed is open adaptive, meaning that a specific bank's page is closed if there are no row hits (but row misses occur), and it remains open if there are no requests directed to that bank.

Processor	X86 ISA, Dual Core, 4GHz, TimingSimpleCPU
L1 I-Cache	32KB private, 4-way, 64B line, 2 cycle access time
L1 D-Cache	32KB private, 4-way, 64B line, 2 cycle access time
L2 D-Cache	128KB shared, 8-way, 64B line, 20 cycle access time
Memory controller	FR-FCFS, open-adaptive, address mapping RoRaBaCoCh, 128B write buffer, 64B read buffer, 64B cache line
Main memory	DDR4 1Gbx8 device, 16 banks, 1 Channel, 2 Ranks per channel, 1200 MHz, BL8, Page size 1 KB
PARSEC	blackscholes, bodytrack, dedup, fluidanimate, freqmine, streamcluster, swaptions, x264

Table 3.1: Simulation setup

Current	***Values (mA)***
IDD0	48
IDD1	60
IDD4R	135
IDD4W	123
IDD5R	53
IDD3N	43
IDD2N	34
IDD3P	37
IDD2P	25
IDD6N	30
IDD6E	35
IDD6R	20

Timing	***Values (ns)***
tCK	0.833
tRAS	32
tRCD	14.16
tREFI	7800
tRP	14.16

Voltage	***Values (V)***
VDD	1.2
VPP	2.5

Table 3.2: Power and timing parameters based on DDR4-2400 8 Gbit device [42]

Our intent is to study self-refresh energy savings with realistic workloads. The PARSEC benchmarks [23] are selected for this purpose, that generate parse traffic to memory with long idle times, thereby exercises the memory device to be in self-refresh mode for most of the times. We utilized the staggered power-down strategy implemented in DRAMPower [54]. This approach enables the memory controller to transition from active power-down mode to precharge power-down mode and subsequently enter self-refresh mode in a staggered manner, reducing the energy consumption of the DRAM. The staggered power-down strategy, developed by Jung et al. [55], aims to achieve additional energy savings by eliminating unnecessary self-refresh entries [56].

In our study, all the active, idle & power-down energy components of the DRAM device are measured. All energy components are categorized into 4 groups, as shown in Table 3.3. G4 is the self-refresh energy, which is of primary interest for us to investigate energy savings with respect to reduced DRAM array voltage at normal (0- 85°C), extended (0- 95°C) and reduced (0- 45°C) temperatures in LPASR mode. Array voltage is reduced up to 150 mV at 25 mV granularity from 1.20 V nominal (i.e. 1.175 V, 1.150 V, 1.125 V, 1.100 V, 1.075 V, 1.050 V) and quantify self-refresh energy savings at three temperature ranges (IDD6N/IDD6E/IDD6N).

Group	**DRAM energy breakdown**	
G1	act/pre	IDD0: One bank ACTIVATE-to-PRECHARGE current IDD1: One bank ACTIVATE-to-READ-to-PRECHARGE current
	Read	IDD4R: Burst read current
	Write	IDD4W: Burst write current
	refresh	IDD5R: Distributed refresh current (1X REF)
G2	actBack	IDD3N: Active standby current
	preBack	IDD2N: Precharge standby current
G3	actPowerDown	IDD3P: Active power-down current
	prePowerDown	IDD2P: Precharge power-down current
G4	selfRefresh	IDD6N: Self refresh current

Table 3.3: DRAM energy components

Chapter 4

Architecture Slack Exploitation for Phase Classification and Performance Estimation

4.1 Workload Classification of Benchmarks

Exploiting slack, when the processor core is busy executing programs, requires a very close activity monitoring of execution units. Our goal is to exploit such slack cycles, combined with other metrics for workload classification and performance estimation.

Counter	Description
Instruction throughput (*cIPS*)	Completed instructions per second throughput of a core
Stall cycles (*cIdle*)	Execution pipeline stall cycles (Not Busy) with outstanding L3 miss

Table 4.1: Events tracked for Performance Management

Table 4.1 lists the counters available in POWER8 that are accessed by our implementations. While the Instruction throughput counters are widely available across different processor families, the *cIdle* counter is a novel counter introduced in POWER8 especially for facilitating the tracking of idleness in the core when waiting on data from memory. *cIPS* measures the rate of completed 'millions of instructions per second (MIPS)' i.e., instruction throughput in a core pipeline that is a direct measure of the performance. *cIdle* measures 'not busy' cycles of a core, when execution units are not busy, while there is at least one L3 cache miss pending, which is a direct measure of the execution pipeline stall cycles.

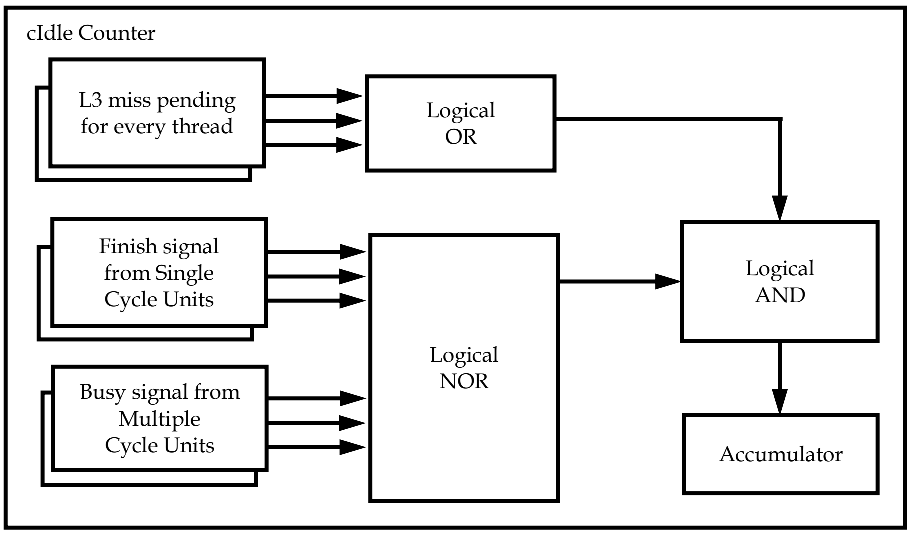

cIdle Counter
L3 miss pending for every thread
Logical OR
Finish signal from Single Cycle Units
Busy signal from Multiple Cycle Units
Logical NOR
Logical AND
Accumulator

Figure 4.1: Hardware implementation of *cIdle* counter

The POWER8 core pipeline has both single-cycle and multiple-cycle execution units. Figure 4.1 illustrates the hardware implementation of the cIdle counter [11]. This counter increases during each execution cycle when at least one thread is waiting for off-core memory access and no threads are actively working, indicating that at least one processor core is not in use. The cIdle counter performs a logical AND operation between the finish signals of single-cycle units, the delayed busy signals of multi-cycle units, and the logical OR operation of L3 miss signals from all threads. The output signal is activated when all pipelines are not busy, meaning there are no finishes from single-cycle units or no delayed busy signals from multi-cycle units. Accumulating this metric over millions of cycles reveals insights into pipeline activity trends during periods of off-core memory access.

We identify that the instruction throughput (MIPS) and not-busy cycles (stalls) as providing measures, which are complementary in nature to understand the fast or slow execution behavior of pipeline. With this intuition, we propose to combine these metrics to characterize the architecture slack and leverage for workload classification, phase classification and performance estimation.

Workload classification can infer how the performance of a workload is bounded to frequency changes. We propose a compounded metric Wb (workload bound) for this classification purpose, which basically is "MIPS to Stalls" ratio, as shown in Equation 4.1. It is measured in millions of instructions per second to million cycles of idle stalls. Wb metric is sensitive to frequency changes, as measures of both instruction throughput and stall cycles are impacted directly by frequency. It is a measure of architecture slack of workloads.

Performance estimation by a DVFS algorithm require accurate prediction of operational relationship between frequency and performance at current and target frequency points (frequency sensitivity), at run-time. It is essentially a measure of "architecture slack" present in the workloads to indicate how many wasted cycles can be eliminated, by lowering the frequency with minimal impact on performance. mipsr (ratio of MIPS between target and current frequency) is a direct measure of frequency sensitivity accounting for architecture slack, between two frequency points. However, measuring mipsr at runtime, has the following drawbacks (a) DVFS algorithms need performance measurement at current and target frequency to determine the sensitivity. This necessitates DVFS controller to slew frequency from current frequency to target frequency point and (b) It must be calibrated periodically to account for phase changes that occurs during runtime. Both (a) and (b) are huge overhead for a performance control algorithm. One important motivation of our work is to evaluate how Wb can be leveraged as proxy to mipsr, by exploiting the relationship between these parameters. This is mainly to address the above-mentioned drawbacks. The intuition behind this approach is that both parameters provide similar measure of frequency sensitivity accounted for architecture slack of workloads.

$$W_b = \left(\frac{cIPS}{cIdle}\right) mips/mcyc \tag{4.1}$$

Equation 4.1 is based on the following simple intuition. For a compute intensive workload, instruction throughput measured by *cIPS* will be high and not-busy cycles measured by *cIdle* will be low. Such trend will be exactly opposite for a memory-bound workload. Therefore, higher bound Wb(high) indicates, more performance impact (more throughput and less stalls) which is tightly bounded to frequency change implying a core-bound workload. Lower bound Wb(low) indicates, less performance impact (less throughput and more stalls) which is loosely bounded to frequency change, implying a memory-bound workload. We have validated "Wb" behavior for a 1.5 GHz broader frequency range, from 2 to 3.5 GHz, at 100

MHz granularity using four custom micro benchmarks: sqroot, fma, mcopy and mlr. sqroot & fma are CPU intensive benchmarks, whereas mcopy & mlr are memory intensive benchmarks.

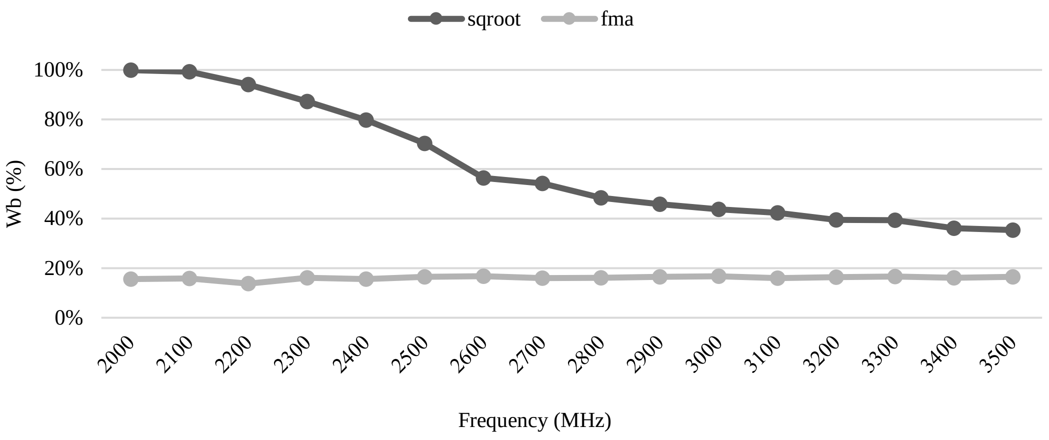


Figure 4 .2: Wb (MIPS to stalls ratio) for custom core-bound benchmarks

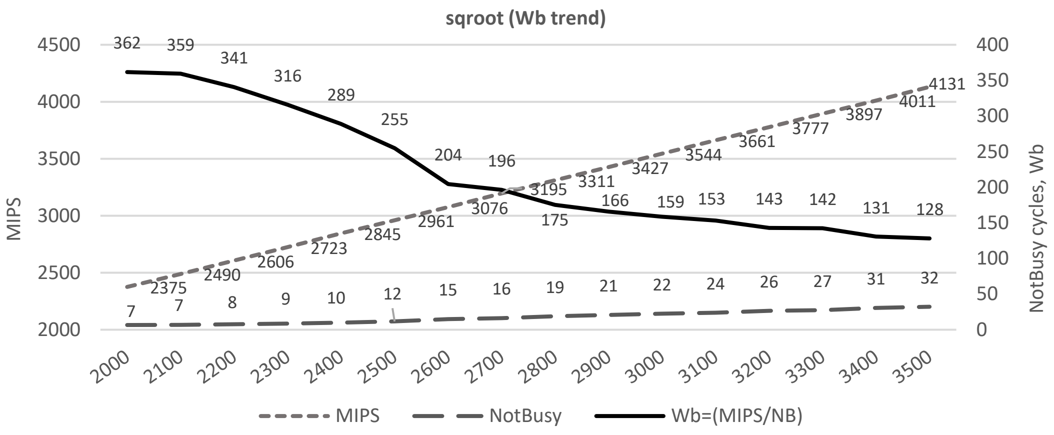


Figure 4 .3: Detailed Wb trend for sqroot core-bound benchmark

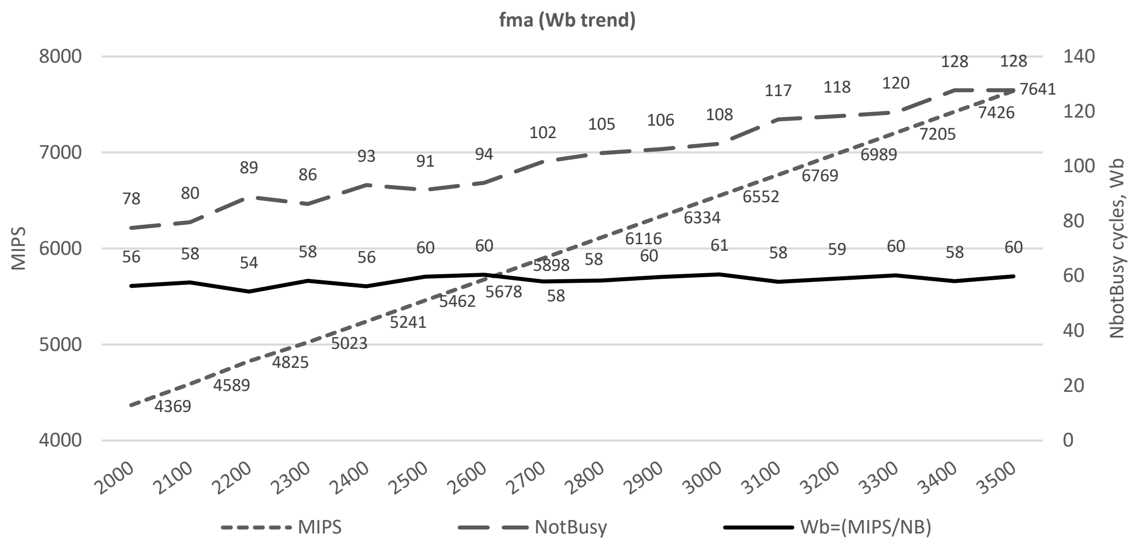


Figure 4.4: Detailed Wb trend for fma core-bound benchmark

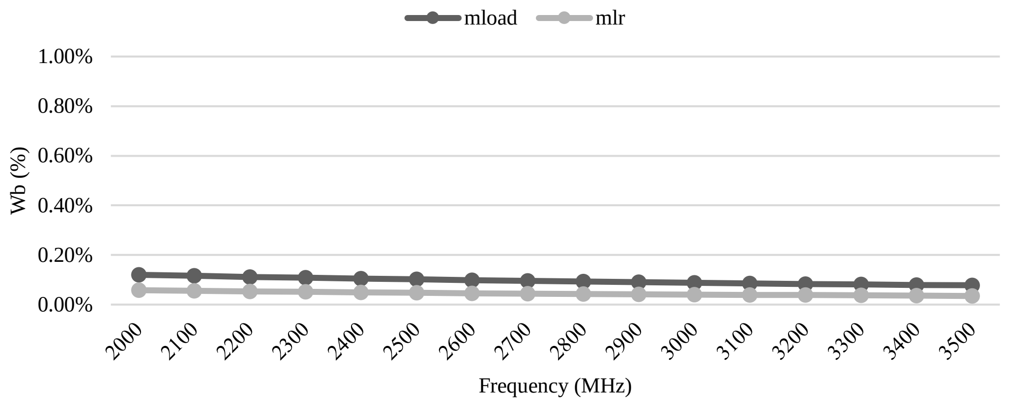


Figure 4.5: Wb (MIPS to stalls ratio) for custom memory-bound benchmarks

The behavior of Wb for these four benchmarks, from 2 GHz to 3.5 GHz, is shown in Figures 4.2 & 4.5. Wb-values shown have been normalized to the peak value which is the value for sqroot at 3.5 GHz. Figures 4.3 and 4.4 illustrate the detailed Wb trend along with MIPS and Not busy cycles, across the 2 to 3.5 GHz full frequency range for sqroot & fma respectively. It is observed that both MIPS & Not busy cycles are impacted at each measurement point of 100 MHz granularity step size, resulted in overall trend of Wb metric at each frequency point. This shows that measure of these two metrics (MIPS, not-busy cycles) is impacted directly by frequency and hence Wb inherently accounts for the same.

In sqroot and fma benchmarks, Wb(high) consistently exceeds 10% across all frequencies, indicating a significantly higher instruction throughput than stalls. On the other hand, in mload and mlr benchmarks, Wb(low) remains below 0.2%, suggesting a considerably higher occurrence of stalls compared to instruction throughput. This substantial disparity in the normalized Wb metric value demonstrates its effectiveness in distinguishing between core-bound and memory-bound workloads.

For a given workload set, either one of the bound Wb(high) or Wb(low) can be used as a threshold for demarcation. In one scenario, a classification algorithm can classify workloads that have Wb above Wb(high) as core-bound workloads and workloads that have Wb below Wb(high) as memory-bound workloads. In another scenario, a classification algorithm can classify workloads that have Wb below Wb(low) as memory-bound workloads and workloads that have Wb above Wb(low) as core-bound workloads. For a given workload or an application set, the Wb(high) and Wb(low) bounds can be learnt over time, based on which control algorithm can decide, what Wb bound to use for demarcation given the optimization targets. In one optimization, Wb(high) can be used a threshold to prioritize core-workloads to guide frequency scaling decisions towards faster completion (favor performance). In another optimization, Wb(low) can be used as a threshold to prioritize memory-workloads to guide frequency scaling decisions towards frequency reduction, with minimal impact in performance (favor energy).

In order to classify workload boundedness and exploit Wb to mipsr correlation, we used SPEC CPU2006 [31] and PARSEC [23] benchmarks in this work. There are total of 38 benchmarks (29 SPECCPU 2006 and 9 PARSEC). We collected traces of MIPS and stalls for all these benchmarks for the entire runtime duration, at 2 GHz and 3.5 GHz frequencies. We computed MIPS ratio (mipsr) and Wb at these two frequency points, as shown in Table 4.2. mipsr(norm) is the mipsr normalized to frequency ratio. Wb(norm) are measured Wb values normalized with current its GHz frequency for 3.5 GHz and 2 GHz frequency bounds.

For workload classification, we have used lower bound of Wb which is Wb(low) as threshold to set memory boundedness as classification criteria. Based on Wb characterization learning with custom benchmarks in section 3, we have leveraged Wb(low) as 0.2% of maximum boundedness. Considering max Wb at 3.5 GHz for these benchmarks, this threshold translates to ~5 (0.2% of 2615). This means, benchmarks with Wb above 5 are determined as core-bound and benchmarks with Wb below 5 are determined as memory-bound. Based on this Wb(low) threshold, there are 23 SPEC CPU2006 and 9 PARSEC benchmarks identified as core-bound workloads and there are 6 SPEC CPU2006 identified as memory-bound workloads, as shown in Table 4.2.

Benchmarks	mipsr (norm)	Wb(norm), 3.5 GHz	Wb(norm), 2 GHz
Core bound (Wb > Wb(low))			
Vips	99.4%	2615	1288
x264	99.6%	1588	921
456.hmmer	98.5%	1490	840
464.h264ref	99.4%	1418	967
freqmine	99.3%	970	465
444.namd	99.3%	948	632
Ferret	98.3%	721	461
416.gamess	98.8%	703	448
465.tonto	99.4%	584	353
400.perlbench	98.5%	464	374
401.bzip2	98.4%	291	258
454.calculix	99.6%	282	194
435.gromacs	99.4%	245	165
445.gobmk	99.5%	218	152
bodytrack	98.3%	178	158
470.lbm	97.2%	158	145
blackscholes	99.2%	157	98
458.sjeng	98.6%	153	125
fluidanimate	98.8%	123	117
447.dealII	95.5%	114	88
453.povray	98.6%	100	59
473.astar	97.5%	48	41
facesim	97.0%	47	36
437.leslie3d	96.6%	43	33
483.xalancbmk	97.4%	38	29
streamcluster	98.1%	36	32
462.libquantum	97.4%	33	19
434.zeusmp	98.4%	30	25
481.wrf	97.0%	28	29
403.gcc	95.1%	18	16
482.sphinx3	96.5%	16	13
436.cactusADM	95.8%	10	11
Memory bound (Wb < Wb(low))			
410.bwaves	85.9%	4	3
471.omnetpp	83.5%	3	3
433.milc	86.3%	3	3
450.soplex	84.0%	2	2
459.GemsFDTD	82.5%	2	2
429.mcf	81.8%	2	1

Table 4.2: Parameters mipsr(normalized), Wb(normalized) values

4.2 Performance Estimation Models

For performance prediction, we have developed compute and memory regression models, which are shown in figures 4.1 and 4.2 respectively. These models exploit the relationship between Wb and mipsr across all benchmarks, at 3.5 GHz and 2 GHz frequency bounds. They provide mipsr performance estimate, at target frequency using Wb measured at current frequency. These models are developed using the following 19 benchmarks (out of 38 total) as training set: (429.mcf, 450.soplex, 433.milc, 410.bwaves, 436.cactusADM, 403.gcc, 434.zeusmp, 437.leslie3d, 453.povray, 447.dealII, 458.sjeng, 445.gobmk, 435.gromacs, 454.calculix, 401.bzip2, 400.perlbench, 416.gamess, 444.namd, 456.hmmer). The remaining 19 benchmarks are used to validate these models (as validation set): (459.GemsFDTD, 471.omnetpp, 482.sphinx3, 462.libquantum, 481.wrf, 483.xalancbmk, streamcluster, facesim, 473.astar, blackscholes, fluidanimate, 470.lbm, bodytrack, 465.tonto, ferret, freqmine, x264, 464.h264ref, vips).

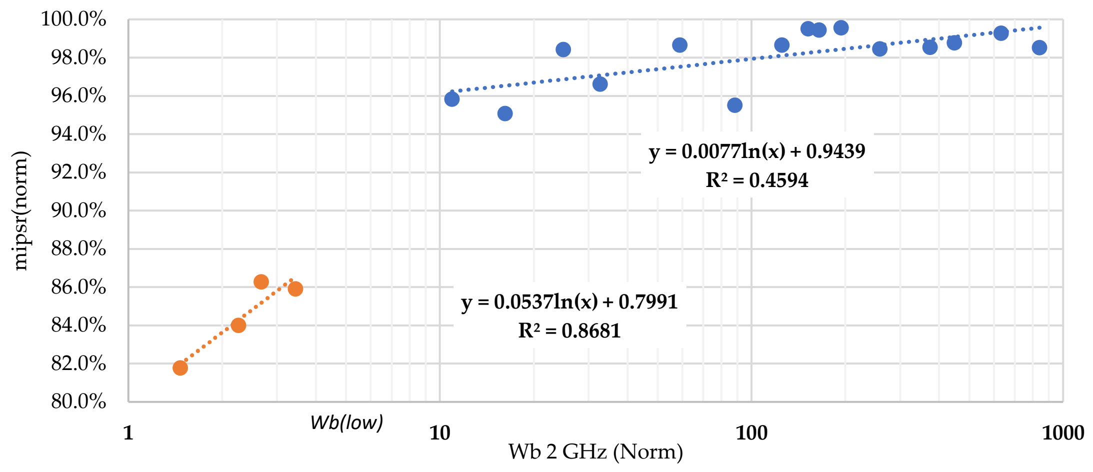


Figure 4.6: Compute and memory regression models for mipsr estimate at 3.5 GHz target frequency

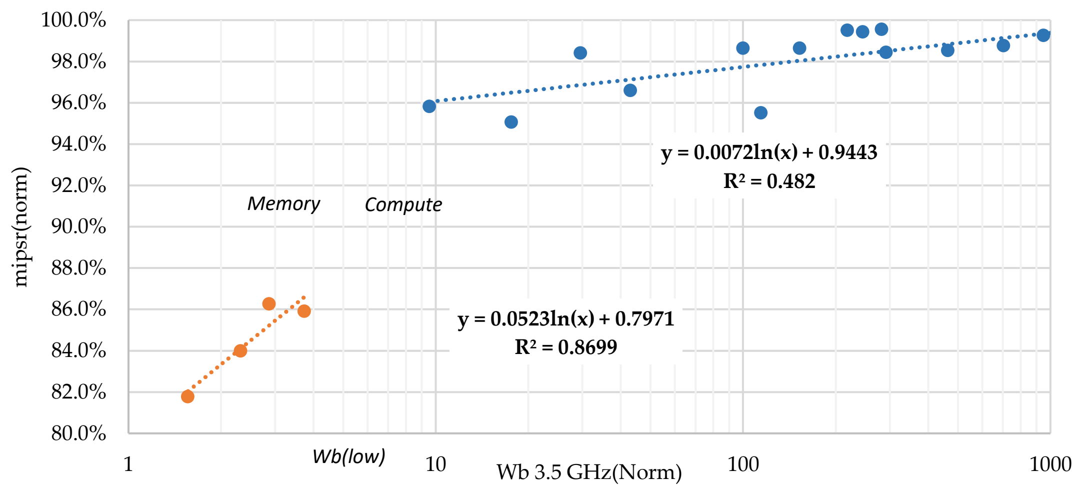


Figure 4.7: Compute and memory regression models for mipsr estimate at 2 GHz target frequency

Based on measurements of these 19 training benchmarks, 15 benchmarks whose Wb values are above Wb(low) are identified as core-bound workloads and 4 benchmarks whose Wb values are below Wb(low) are identified as memory-bound workloads. Such identified 15 core-bound and 4 memory-bound benchmarks, are used to realize compute and memory regression models. Figure 4.6 shows the mipsr and Wb relationship for core-bound and memory-bound benchmarks at 3.5 GHz target frequency, using 2 GHz measurements and their models are shown in Equations 4.2 & 4.3. Figure 4.7 shows the mipsr and Wb relationship for core-bound and memory-bound benchmarks at 2 GHz target frequency, using 3.5 GHz measurements and their models are shown in Equations 4.4 & 4.5.

$$cperf_{(3.5\,GHz)} \approx [0.0077\, ln(W_b@2GHz) + 0.9439] \tag{4.2}$$

$$mperf_{(3.5\,GHz)} \approx [0.0537\, ln(W_b@2GHz) + 0.7991] \tag{4.3}$$

$$cperf_{(2\,GHz)} \approx [0.0072\, ln(W_b@3.5GHz) + 0.9443] \tag{4.4}$$

$$mperf_{(2\,GHz)} \approx [0.0523\, ln(W_b\ @3.5GHz) + 0.7971] \tag{4.5}$$

All 38 benchmarks (including both validation and training sets) are used to validate compute and memory regression models for their performance prediction accuracy and detailed results are presented later in section 5.

4.3 Phase Classification at Runtime

Classifying a workload on an interval by interval during runtime, is important to detect its boundedness, based on which frequency scaling-factor can be determined for performance management. We define this as “phase”, which is essentially the estimate of Wb for a given interval of measurement. A phase transition or change can be determined based on Wb value transition from high to low (compute to memory) or from low to high (memory to compute). For phase classification, we have defined a fixed interval of 1 second. We accumulated Wb over this interval and evaluated phase of three benchmarks having distinctive phase characteristics: compute bound (400.perlbench), memory bound (471.omnetpp) and mixed bound (433.milc). Here mixed bound implies, the compute phases (Wb above Wb(low)) and memory phases (Wb below Wb(low)) alternate during runtime. We characterized phase behavior for entire duration of the runtime at four different frequency points (3.5/3/2.5/2 GHz) to verify its consistency across broader operating frequency ranges, as shown in Figure 4.5.

4.4 Performance Evaluation of Benchmarks

Performance estimation is done by DVFS algorithms to support frequency scaling. Essentially, it is a measure of frequency-performance relationship at target frequency with respect to its current frequency-performance setting, as shown in Equation 4.6.

$$mips_{(target)} = \left(\frac{freq_{(target)}}{freq_{(current)}}\right) * mips_{(curent)} * (1 - \beta) \qquad (4.6)$$

In Equation 4.6, β refers to the memory boundedness of a workload. For an ideal compute bound workload, (β=0), performance will track linearly with frequency. However, for any practical workload, performance will track non-linearly, as there will be certain amount of memory transactions involved, as shown by mipsr in Equation 4.7.

$$mipsr \ \alpha \ \frac{mips_{(target)}}{mips_{(curent)}*(1- \beta)} \qquad (4.7)$$

For establishing the relationship between Wb and mipsr, we have developed regression models at two extreme frequency points, 2 GHz and 3.5 GHz, using SPECCPU 2006 and PARSEC benchmarks suite. It has mix of CPU and memory intensive benchmarks, therefore there are two models realized, cPerf and mPerf to estimate mipsr for compute and memory bound workloads. These models can give an estimate of mipsr at target frequency, using Wb at current frequency.

We summarize all above stated methods and propose the overall performance management scheme as shown in Figure 4.8. It comprises three major steps (a) Workload bound detection (b) Workload or phase classification and (c) Performance estimation.

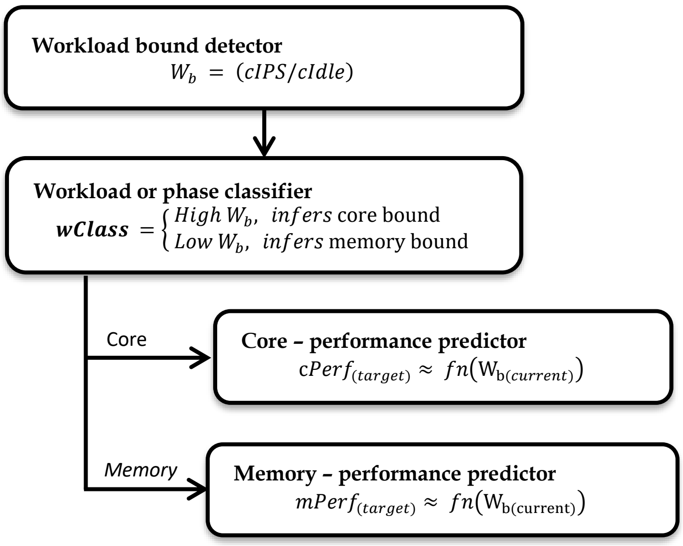

Workload bound detector
$W_b = (cIPS/cIdle)$
Workload or phase classifier
$wClass = \begin{cases} High\ W_b,\ infers \text{ core bound} \\ Low\ W_b,\ infers \text{ memory bound} \end{cases}$
Core
Core – performance predictor
$cPerf_{(target)} \approx fn(W_{b(current)})$
Memory
Memory – performance predictor
$mPerf_{(target)} \approx fn(W_{b(current)})$

Figure 4.8: Performance prediction scheme

We have used this scheme to build accurate performance predictors. In first part of our work, we have developed offline regression models to exploit correlation of Wb and mipsr parameters, using a subset of benchmark suite (training set). Here offline refers to measurement of these parameters for entire runtime of these benchmarks. To assess the accuracy of our prediction models, we conducted validation using the remaining benchmarks in our dataset. In the second part of our study, we devised an online interval-based approach for classifying phases of benchmarks based on their unique characteristics. These phases include compute bound, memory bound, and mixed bound workloads. We have verified runtime performance prediction accuracy of few benchmarks using cPerf and mPerf models with measured performance traces. We have also developed an algorithm to determine compute or memory phases at runtime to guide frequency scaling decisions and evaluated execution time impact of all benchmarks.

We implemented a max-min performance-threshold algorithm for performance management. We studied the performance impact by comparing execution times of all benchmarks, between max-min and "Dynamic Power Saver, Favor Performance (DPS-FP)" algorithm in POWER8 [4].

Max-min algorithm collects MIPS and not-busy cycle traces to compute Wb for the current phase interval of 1 second duration. Using computed Wb, it determines the compute or memory phase, based on whether Wb is above or below Wb(low) respectively. It then drives the frequency to freq(min) for memory phase and freq(max) for compute phase. We have set freq(min) threshold to 2 GHz for memory phase to guide towards minimum frequency and freq(max) threshold to 3.5 GHz for compute phase to guide towards maximum frequency.

ALGORITHM 1: MAX-MIN PERFORMANCE THRESHOLD

```
Input: All 38 benchmarks, Wb
Output: Slew frequency to max or min frequency depends on compute or memory phase respectively
1.  Initialization: $freq_{(min)}$= 2 GHz, $freq_{(max)}$= 3.5 GHz, $W_b = W_{b(low)}$(5), Phase duration = 1 sec, interval i = 0;
2.  While (1) do
3.  Collect MIPS and not-busy cycles for for 1 sec intervals (i)
4.  Compute $W_{b(i)}$ for every $i_{th}$ interval
5.  if ( $W_{b(i)} > W_{b(low)}$)
6.  Compute MIPS using ($W_{b(i)}$, cPerf)
7.  Increment i++                /*Next interval*/
8.  Slew to $freq_{(max)}$ for $W_{b(i)}$     /*Increase performance */
9.  else
10. if ( $W_{b(i)} <= W_{b(low)}$)
11. Compute MIPS using ($W_{b(i)}$, mPerf)
12. Increment i++                /*Next interval*/
13. Slew to $freq_{(min)}$ for $W_{b(i)}$     /* Eliminate wasteful stall cycles waiting on memory resources*/
14. end if
15. end while
```

DPS-FP is a DVFS algorithm specifically designed for POWER8 to optimize performance. It utilizes MIPS throughput to determine the level of core utilization and adjusts the frequency accordingly. When the core is moderately or heavily utilized, the frequency is increased to the maximum (3.5 GHz) to maximize performance. On the other hand, if the core is lightly utilized or idle, the frequency is lowered to the minimum (2 GHz) to conserve power.

Table 4.3 shows the comparison of benchmarks execution time between max-min and DPS-FP algorithms. Column A lists all the core and memory benchmarks. Columns B and C list the runtime (seconds) of all benchmarks measured with DPS-FP and max-min algorithms respectively. Column D shows the comparison of runtime differences (decrease or increase) across all benchmarks.

A	*B*	*C*	*D*
Benchmarks	**DPS-FP Algo (default)**	**(2-3.5) GHz Min-Max Algo**	**Runtime decrease/ increase (%)**
CPU Benchmarks			
Vips	406	405	-0.2%
x264	465	463	-0.4%
456.hmmer	487	485	-0.4%
464.h264ref	1360	1353	-0.5%
freqmine	969	968	-0.1%
444.namd	716	714	-0.3%
Ferret	485	483	-0.4%
416.gamess	2487	2482	-0.2%
465.tonto	746	743	-0.4%
400.perlbench	1025	1020	-0.5%
401.bzip2	1576	1572	-0.3%
454.calculix	1066	1063	-0.3%
435.gromacs	677	676	-0.1%
445.gobmk	1704	1700	-0.2%
bodytrack	460	458	-0.4%
470.lbm	206	205	-0.5%
blackscholes	331	330	-0.3%
458.sjeng	2441	2438	-0.1%
fluidanimate	855	851	-0.5%
447.dealII	1258	1253	-0.4%
453.povray	656	654	-0.3%
473.astar	1204	1199	-0.4%
Facesim	476	474	-0.4%
437.leslie3d	149	149	0.0%
483.xalancbmk	774	771	-0.4%
streamcluster	530	528	-0.4%
462.libquantum	188	188	0.0%
434.zeusmp	775	772	-0.4%
481.wrf	825	822	-0.4%
403.gcc	864	861	-0.3%
482.sphinx3	977	973	-0.4%
436.cactusADM	290	289	-0.3%
Memory Benchmarks			
410.bwaves	185	201	8.1%
471.omnetpp	814	877	7.2%
433.milc	434	466	6.9%
450.soplex	516	559	7.8%
459.GemsFDTD	506	546	7.4%
429.mcf	475	509	6.6%

Table 4.3: Benchmarks – Performance impact study

4.5 Results and Discussion

MIPS prediction accuracy is a critical metric in evaluating the performance of computer processors, particularly in the context of estimating the processor's capability to execute a given number of instructions per second at various frequencies. This accuracy is determined by comparing the MIPS estimated through computational models, such as cPerf and mPerf, against the MIPS actually measured on the hardware for a target frequency, as shown in Equation 4.8. The process of estimating MIPS involves specific steps outlined in an algorithm, where lines 6 and 11 play a pivotal role in generating the MIPS estimate. Essentially, this comparison between estimated and measured MIPS allows for the assessment of the precision of the models used in predicting processor performance under different operational conditions.

The accuracy of MIPS prediction was evaluated through empirical measurements conducted on hardware running at frequencies of 2 GHz and 3.5 GHz. These measurements were taken over the entire runtime duration of the processor while executing benchmark suites like SPEC CPU2006 and PARSEC, which are widely recognized for their ability to simulate real-world computing environments and workloads. During these tests, both the MIPS and the not-busy cycles - periods when the processor is not executing any instructions - were meticulously recorded. Furthermore, a metric known as Wb was calculated based on the data collected from these measurements. The Wb metric, derived from the measured MIPS and not-busy cycle traces at the two specified frequency points, serves as an additional parameter in evaluating the processor's efficiency and the accuracy of MIPS predictions, thereby providing a comprehensive view of the processor's performance characteristics.

$$MIPS\ (error) = (MIPS_{(predicted)} \sim MIPS_{(measured)}) \tag{4.8}$$

In the realm of computing performance evaluation, the concept of using Wb as a proxy for MIPS has been explored with promising results. Specifically, Wb values computed at a base frequency of 2 GHz were utilized to forecast MIPS at a higher target frequency of 3.5 GHz. This prediction was facilitated by employing specific prediction models, as detailed in Equations 4.1 and 4.2 found in section 4.2 of the referenced document. The accuracy of these predictions was then assessed by comparing the predicted MIPS values against the actual MIPS measurements obtained at the 3.5 GHz frequency. The comparison, illustrated in Figure 4.9, revealed that the discrepancy between the predicted and actual MIPS values was within a narrow margin of error, not exceeding 3% across various benchmarks. This outcome underscores the efficacy of using Wb as a reliable indicator for MIPS rates across different frequencies, thereby supporting its potential as a versatile tool in performance evaluation.

Further extending this methodology, the study also investigated the reverse scenario where Wb values computed at the higher frequency of 3.5 GHz were used to predict MIPS at the lower target frequency of 2 GHz. This prediction process was guided by another set of prediction models, encapsulated in Equations 4.3 and 4.4, as outlined in the same section 4.2. The fidelity of these predictions was similarly evaluated by comparing the predicted MIPS against the actual MIPS measurements at the 2 GHz frequency, with the findings depicted in Figure 4.10. Remarkably, the error margin between the predicted and measured MIPS values remained within the 3% threshold across all benchmarks in this scenario as well. This consistency in prediction accuracy across both directions of frequency change further validates the robustness of Wb as a proxy for MIPS, demonstrating its applicability over a broad spectrum of frequencies in computing performance assessments.

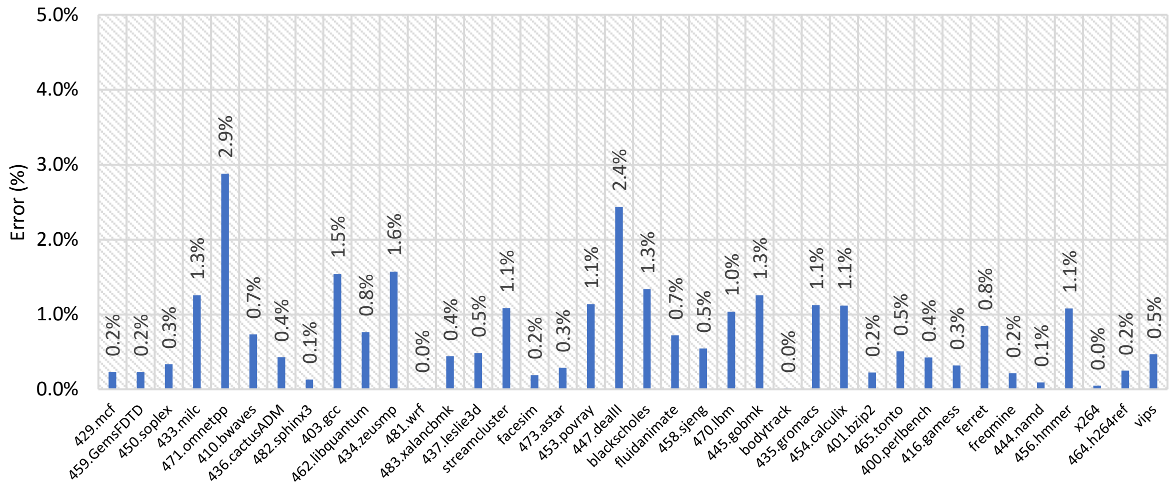


Figure 4.9: MIPS predicted (using models) vs measured error at 3.5 GHz

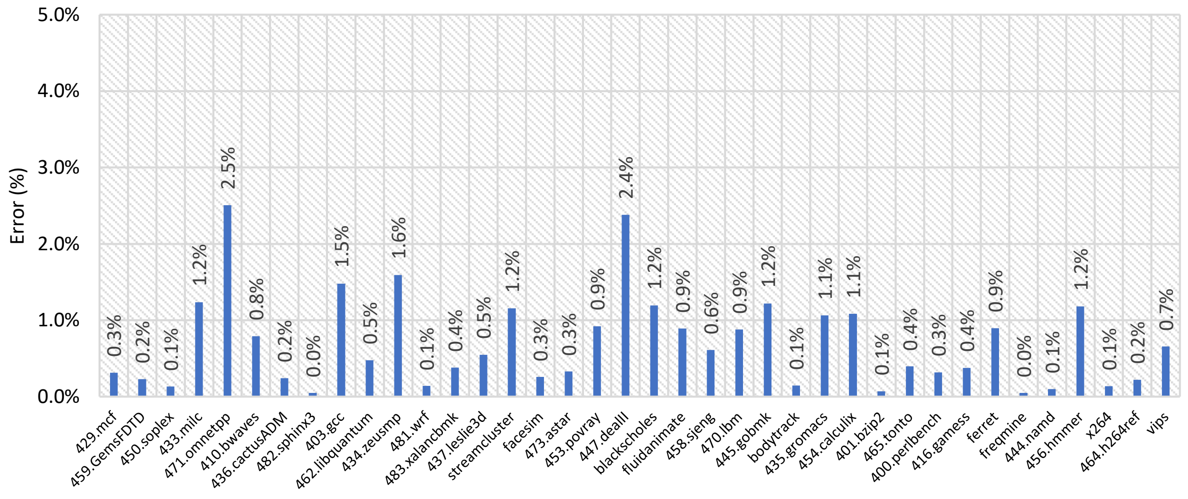


Figure 4.10: MIPS predicted (using models) vs measured error at 2 GHz

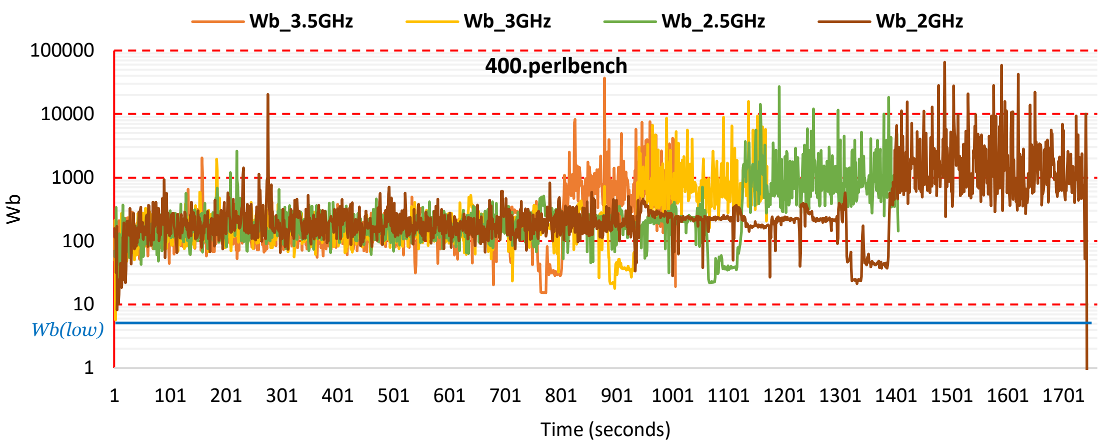
In the realm of online phase classification, the metric Wb over a one-second duration serves as a pivotal unit of measurement for analyzing the runtime phase characteristics of various benchmarks. Specifically, Figure 4.11 delves into the examination of three distinct benchmarks—400.perlbench, 471.omnetpp, and 433.milc—across four varying frequencies: 3.5 GHz, 3 GHz, 2.5 GHz, and 2 GHz. The analysis reveals that 400.perlbench consistently maintains a Wb value above the designated low threshold (Wb(low)), indicating its highly compute-bound nature throughout its runtime. Conversely, 471.omnetpp demonstrates a Wb value significantly below the Wb(low) threshold, categorizing it as highly memory-bound during its entire runtime. Meanwhile, 433.milc presents a more complex scenario, with its Wb value fluctuating above and below the Wb(low) threshold, thereby showcasing a transient behavior with mixed phases of compute and memory-bound characteristics.

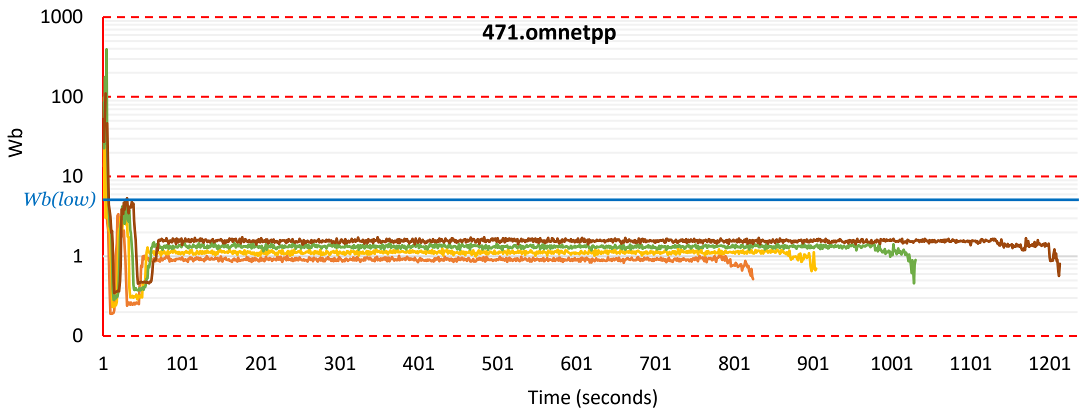
This intricate analysis underscores a crucial observation: the phase classification, as determined by the Wb value, remains consistent across different frequency settings, whether the value stays above or below the threshold. This consistency in behavior, irrespective of the frequency variations, suggests that the utilization of Wb for phase classification is not only effective but also versatile, capable of adapting to a broad spectrum of frequency ranges. Such a revelation is instrumental in understanding the inherent characteristics of benchmarks, enabling a more nuanced approach to optimizing performance across diverse computational environments. The ability to accurately classify phases based on Wb values, therefore, holds significant promise for enhancing the efficiency and adaptability of computing systems, paving the way for more sophisticated performance optimization strategies that can cater to the specific demands of various benchmarks.

The comparison between the max-min threshold algorithm and the conventional DPS-FP algorithm offers insightful revelations about their efficacy across different types of benchmarks. The max-min threshold algorithm dynamically adjusts the processor frequency to 3.5 GHz for compute-intensive phases and to 2 GHz for memory-intensive phases, contingent upon Wb surpassing or not surpassing a predefined low threshold (Wb(low)). Conversely, the DPS-FP algorithm maintains a frequency of 3.5 GHz for moderate to high core utilization scenarios and reduces it to 2 GHz under conditions of minimal core activity or idleness. This distinction in frequency management is particularly pronounced in core benchmarks, where the prevalence of compute phases over memory phases for the majority of the workload runtime is substantiated by high Wb values as documented in Table 4.2, and further illustrated by the high MIPS) and low Not Busy cycles of three core benchmarks depicted in Figure 4.12. Despite the high MIPS and Wb leading both algorithms to operate at 3.5 GHz for most of the workload runtime, the observed runtime difference between the two algorithms across all core benchmarks is marginal, at merely 0.5%.

In contrast, the behavior of these algorithms diverges more significantly in memory benchmarks, characterized predominantly by memory phases over compute phases throughout the workload runtime. This scenario is corroborated by the low Wb values presented in Table 4.2 and the corresponding low MIPS and high Not Busy cycles of three memory benchmarks, as demonstrated in Figure 4.13. The DPS-FP algorithm's propensity to elevate the frequency to 3.5 GHz even with moderate core activity results in expedited execution. However, the max-min threshold algorithm, guided by the Wb value, reduces the frequency to 2 GHz, effectively minimizing wasteful cycles that would otherwise be spent in stalls awaiting memory resources during memory phases. This frequency adjustment strategy by the max-min algorithm leads to a notable performance improvement, with an average increase of approximately 7.3% across all memory benchmarks, and a peak increase of about 8.1% for the 410.bwaves benchmark.

Further validation of these algorithm’s performance was conducted through the evaluation of predicted versus measured runtime performance for both compute-bound (400.perlbench) and memory-bound (471.omnetpp) benchmarks within a frequency range of 2-3.5 GHz. The comparison, as depicted in Figure 4.14, showcases the close alignment between the predicted performance, derived from cPerf and mPerf models, and the actual measured performance, with a deviation within a 3% error margin. This congruence underscores the reliability of the max-min threshold algorithm in optimizing performance by judiciously managing processor frequencies based on the nature of the workload, thereby affirming its potential as a superior alternative to the conventional DPS-FP algorithm for certain benchmark categories.





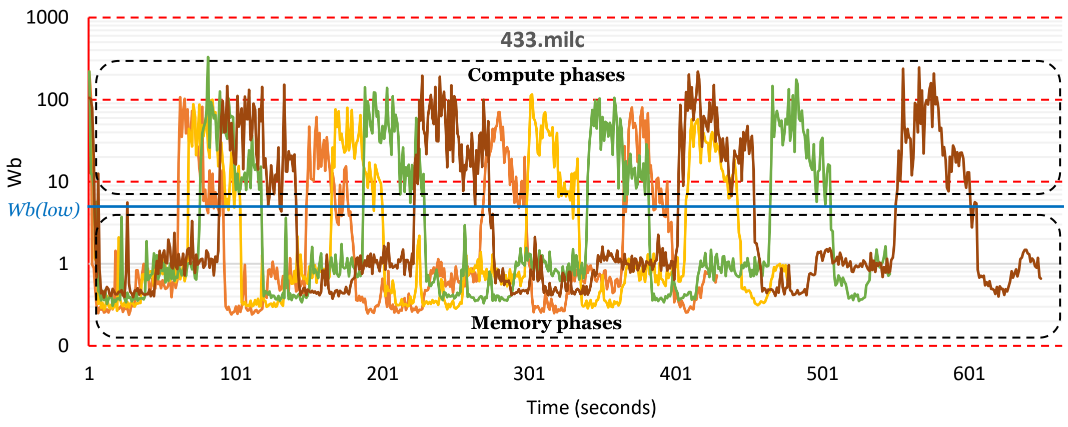
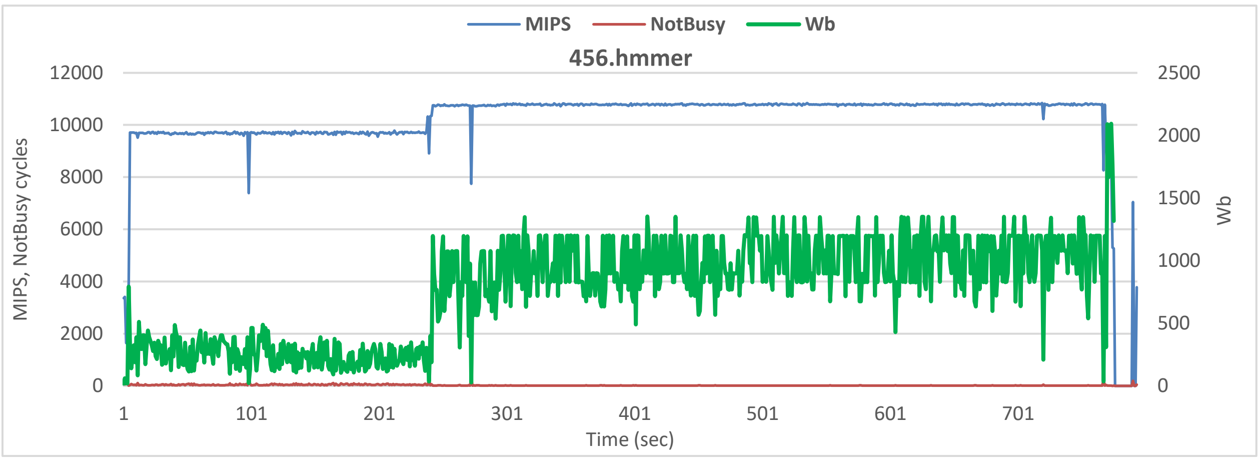
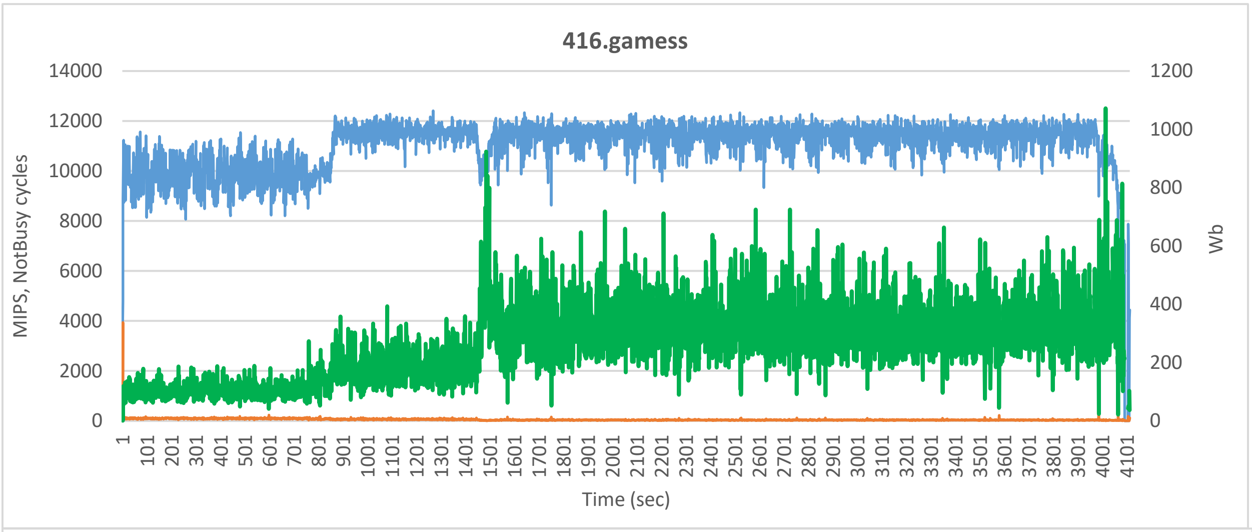


Figure 4.11: Runtime phase characteristics of 400.perlbench, 471.omnetpp and 433.milc benchmarks





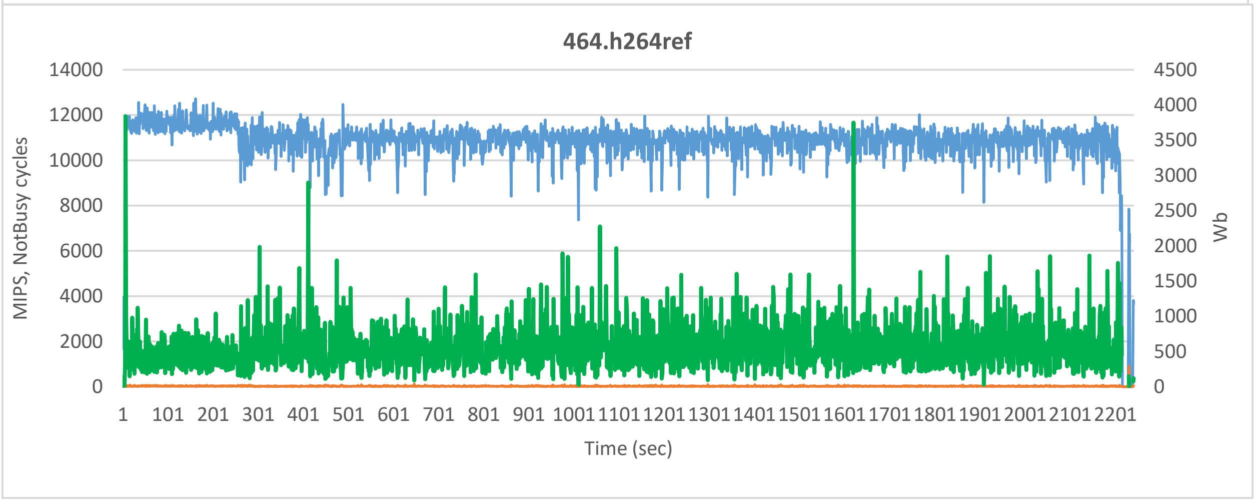


Figure 4.12: MIPS, NotBusy & Wb characteristics of core-benchmarks (456.hmmer, 464.h264ref and 416.gamess)

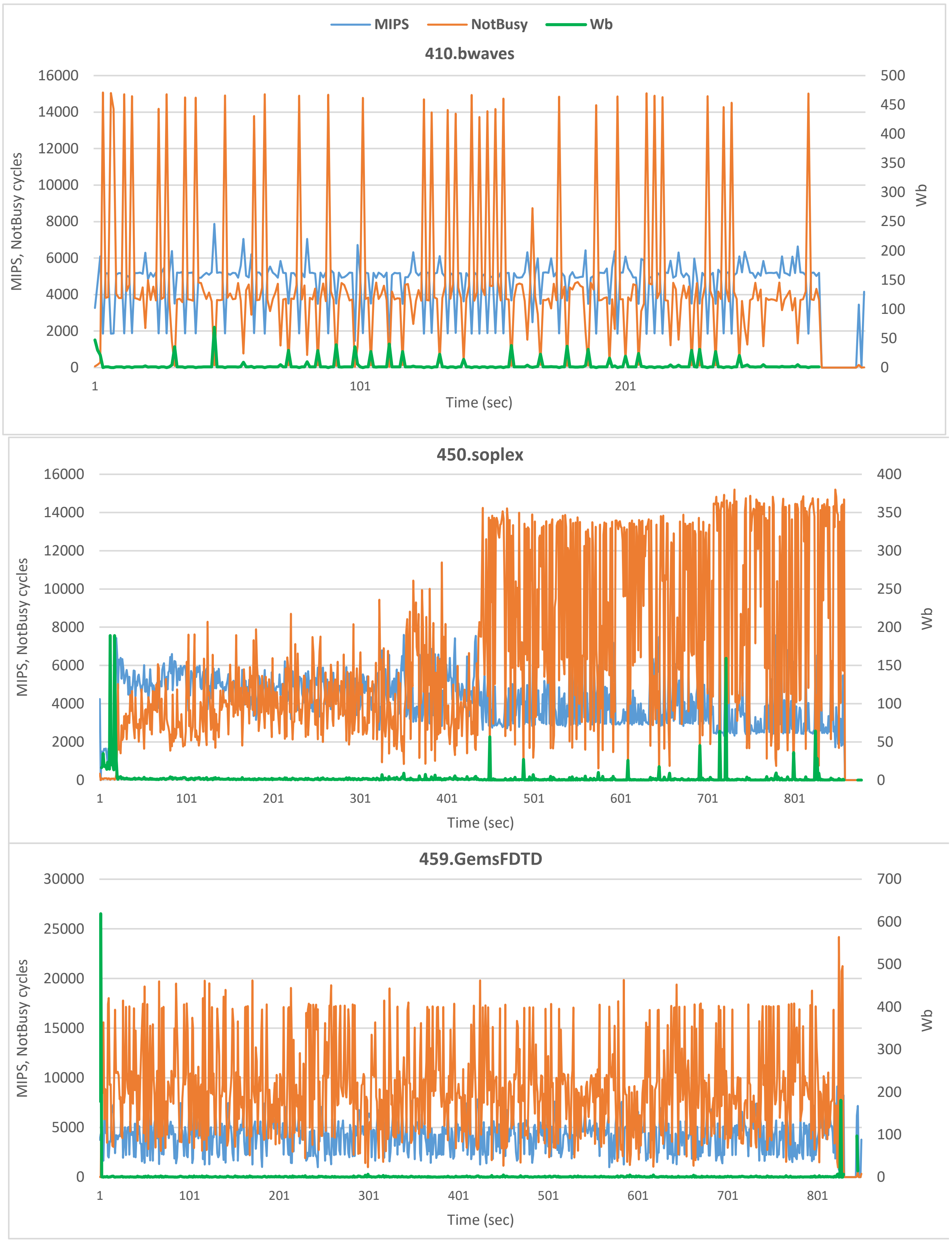
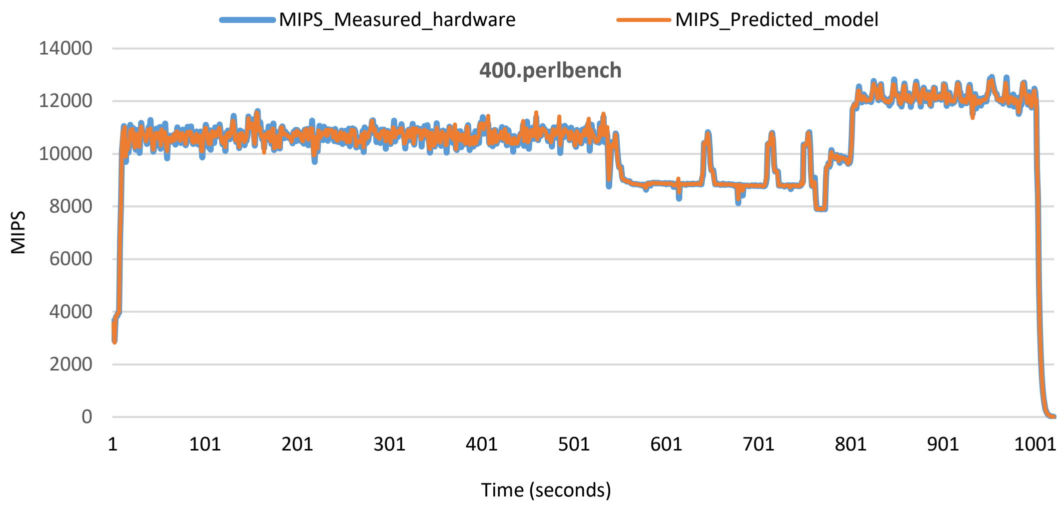


Figure 4.13: MIPS, NotBusy & Wb characteristics of memory-benchmarks (410.bwaves, 450.soplex and 459.GemsFDTD)



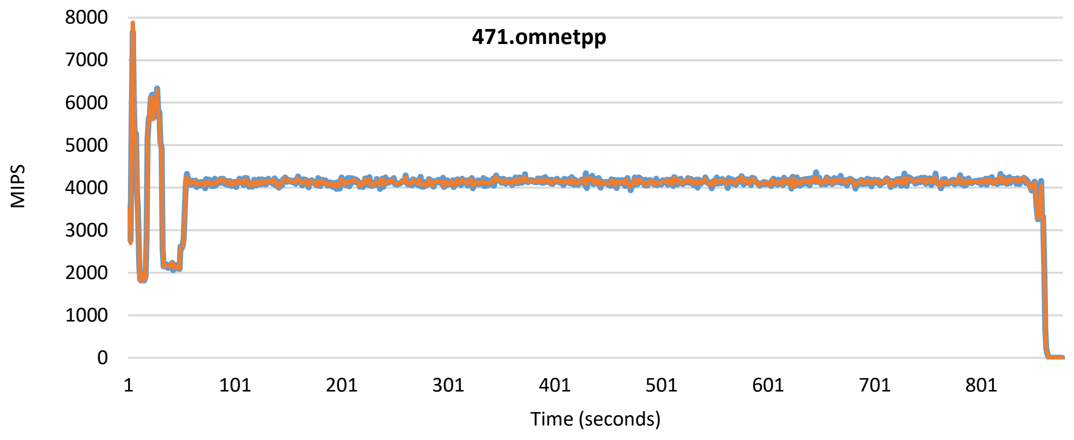


Figure 4.14: MIPS predicted vs measured for 400.perlbench and 471.omnetpp benchmarks

Chapter 5

Voltage Reduced Self Refresh (VRSR) for Optimized Energy Savings in DRAM Memories

5.1 Impact of Reduced Voltage and Temperature on Self-refresh

Refresh essentially requires activation, restore and precharge operations of every row periodically, as shown in Figure 5.1. Before every row activation, the bit lines and sense amplifiers of the selected bank must be properly precharged. The precharge time (tRP) represents the duration, measured in clock cycles, required to end access to an open row of memory and precharge the bitline (BL). Row activation necessitates a specific number of clock cycles (tRCD) for data to become available at the sense amplifiers, although it has not yet been restored to the DRAM cells. Following the activation process, the data restoration operation is completed after a time period of tRAS, starting from the beginning of the activation process. Once this process is finished, the DRAM device is ready to receive a precharge command. The latency of these three crucial operations (tRP, tRCD, and tRAS) depends on factors such as the cell capacitance, bitline capacitance, and array voltage [45].

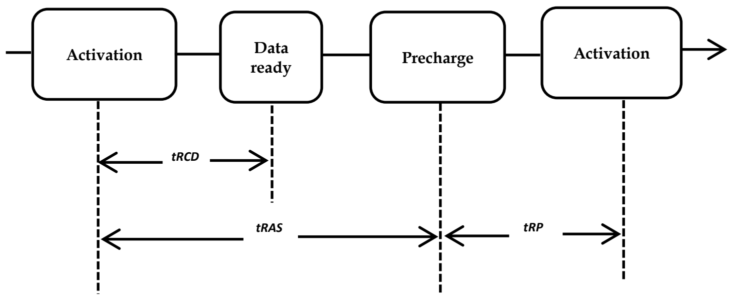


Figure 5.1: Timing parameters associated with refresh operation of a row

Commodity DRAM devices necessitate the refreshing of each cell within a specific retention period, typically every 32ms or 64ms. The retention time of a cell refers to the duration it can retain sufficient charge for the sense amplifiers to detect either '0' or '1', and it is directly influenced by the supply voltage of the DRAM array. Lowering the supply voltage reduces the amount of charge stored in a cell, resulting in a decreased retention time. In typical DRAM configurations, internal voltage generators are utilized to enable operation with a single external power supply voltage (VDD), as illustrated in Figure 5.2. The memory array voltage-down converters (Varray) and the DRAM control and logic circuitry voltage-down converters (Vccp) generate the necessary voltage levels. The memory cell array is subjected to negative back bias voltage (Vbb), bit line precharge voltage (Vbl), and cell plate voltage (Vcp). The boosted word line voltage (Vpp) applied to the word line driver enhances random row access time. In modern DRAM systems [79], to reduce idle power consumption during self-refresh mode, the peripheral circuitry (interface and delay locked loop) is deactivated. Furthermore, the memory controller is disconnected, and refresh operations are autonomously performed by the internal counter. Consequently, decreasing the supply voltage (VDD) solely impacts the reliability, latency, and data retention characteristics of the DRAM array.

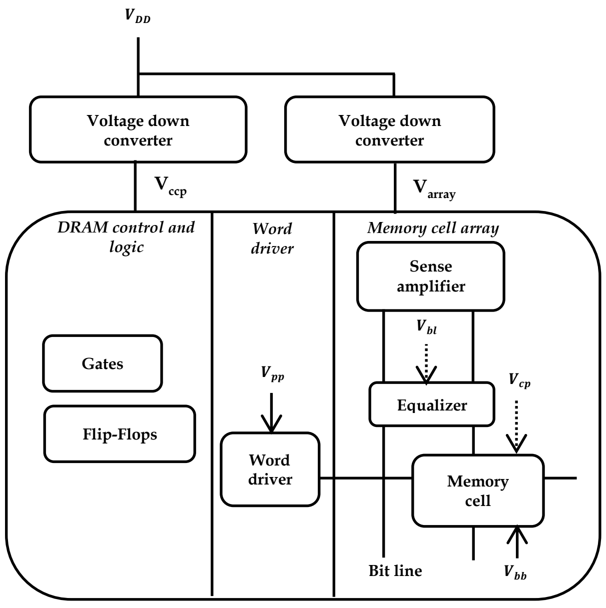


Figure 5.2: Internal voltages in DRAM [46]

The temperature has a significant impact on data retention in DRAM devices [80]. Higher temperatures lead to increased leakage current, requiring higher refresh rates to ensure data integrity. On the other hand, at lower temperatures, refresh rates can be reduced to minimize power consumption while still maintaining data integrity. JEDEC has introduced the LPASR (Low Power Auto Self Refresh) feature to optimize the refresh current based on the temperature range. In this mode, the DRAM adjusts its self-refresh rate according to the operating temperature, reducing it at low temperatures and increasing it at high temperatures [42]. The DRAM device autonomously manages the entry into self-refresh mode within the supported temperature range. Figure 5.3 demonstrates that for the normal temperature range of 45°C to 85°C, the device maintains a 1X refresh rate. For the extended temperature range of 85°C to 105°C, it maintains a 2X refresh rate, while for the reduced temperature range of -40°C to 45°C, it maintains a 1/2X refresh rate.

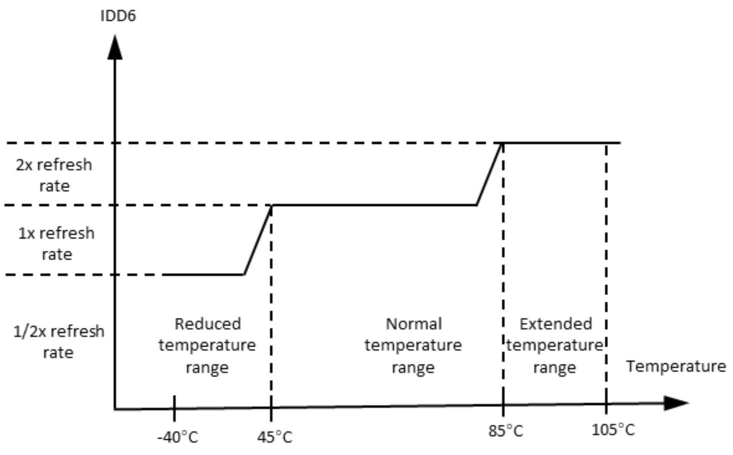


Figure 5 .3: Refresh rates associated with temperature ranges in LPASR [42]

The self-refresh current at extended and reduced temperatures are denoted as IDD6E and IDD6R respectively. Due to increased refresh rate at extended temperature, the IDD6E increases significantly. Conversely, due to lower refresh rate at reduced temperature, the IDD6R decreases significantly.

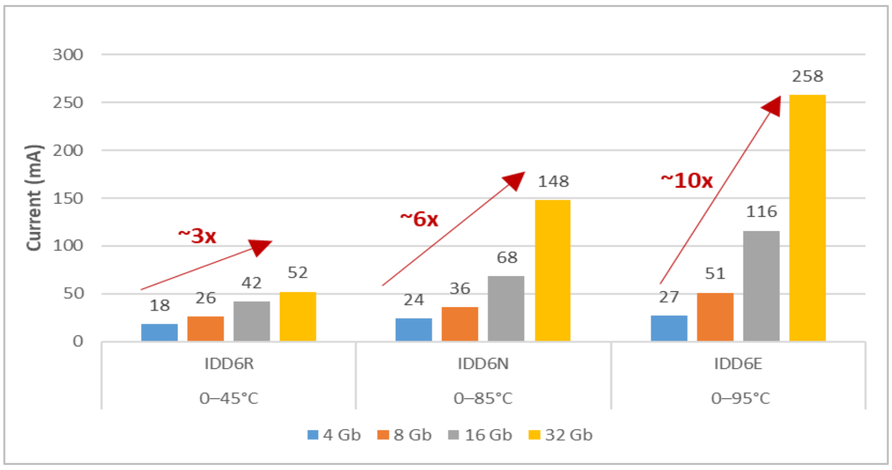


Figure 5 .4: Self-refresh power trend in DDR4 devices at different capacities [42], [8-10]

In this work, an attempt has been made to understand the self-refresh power increase trend and how IDD6E, IDD6R vary compared to IDD6, across different capacity devices. For this purpose, 4/8/16/32 Gb DDR4 devices have been considered. Our study leads to a key observation i.e., as device capacity grows (4 Gb to 32 Gb), the IDD6R, IDD6N, IDD6E increase by ~3x, ~6x, ~10x respectively, as shown in Figure 5.4. Owing to the fact that, the refresh rate is doubled at high temperatures, the increasing trend in IDD6E strongly motivates us to investigate power savings at reduced DRAM supply voltages.

We are the first to present the detailed empirical study of energy savings at reduced voltages across different supported temperature ranges, during "self-refresh" mode of operation in DDR4 DRAMs. At reduced supply voltages, the internal operations (tRP, tRAS & tRCD) require extra time to finish, and this introduces additional latency cycles. In self-refresh mode, these operations are managed internally by the DRAM without the involvement of memory controller. Since there is no data read from a column involved, tRCD is not of significance for our study. The latency impacts of tRAS and tRP operations are measured with the help of SPICE model. For this purpose, the sense amplifier design from 45 nm SPICE model [43] is updated appropriately for the 16 nm transistor model. With these increased latency values identified for a range of six reduced voltage levels, energy savings are quantified at normal (1x standard refresh), extended (2x refresh) and reduced (0.5x refresh) temperature ranges, as supported in LPASR modes.

Eight PARSEC benchmarks as a workload set. To quantify the idle times, we have compared the energy between active and power-down modes which are grouped into G1 and G2 respectively.

G1 group:

- Active: At least one bank is operational, meaning there is no power-down mode (cke=1), and the internal refresh is not performed automatically (the DRAM controller must schedule refresh commands).

G2 Group:

- Precharge Power-Down (PDNP) state: In this state, all banks are closed and precharged, which occurs when the DRAM is in the IDLE state and cke=0. There is no internal refresh operation.
- Active Power-Down (PDNA) state: In this state, at least one bank is active, indicated by the cke=0 signal, while the other banks remain closed and precharged. There is no internal refresh operation.
- Self-Refresh (SREF) state: In this state, all banks are precharged and closed, and the DRAM initiates its internal self-timed refresh process. The cke=0 signal is used to trigger this self-refresh operation.

As shown in Figure 5.5 Energy in power-down modes (G2) dominates significantly over active state (G1) and this clearly indicates low traffic generated across benchmarks. Same observation has been validated with memory throughput metric, as shown in Figure 5.6. This ensures that PARSEC benchmarks generate very low traffic are used as a workload set to exercise long idle times. During run time, memory controller will detect idle times present in such low traffic scenarios and command DRAM to enter into self-refresh mode. The benchmarks are simulated using Gem5 [52] and the IDD6 power pertaining to different temperature ranges at reduced voltages is measured using the DRAMPower [54].

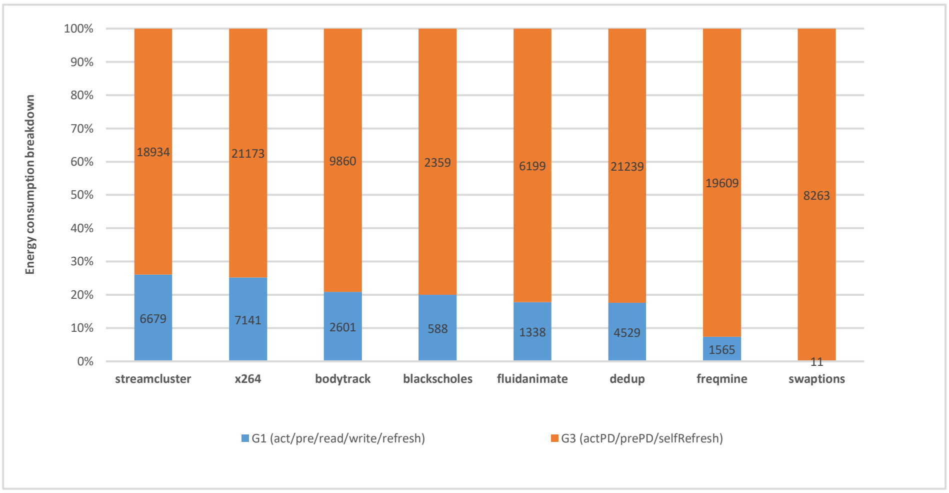


Figure 5 .5: Comparison of energy breakdown between active and power-down modes

These increased tRAS & tRP latencies also increase row refresh cycle time (tRFC) in self-refresh mode and creates a time overhead in the exit latency. As vendors do not disclose their specific implementation of self-refresh scheme, the detailed quantitative evaluation of workload performance impact due to tRFC refresh overhead is studied with auto-refresh scheme as a baseline. Also, given the unavailability of HW setup & lack of voltage control capabilities (DRAM array), our current work focus on the simulation to quantify the energy savings.

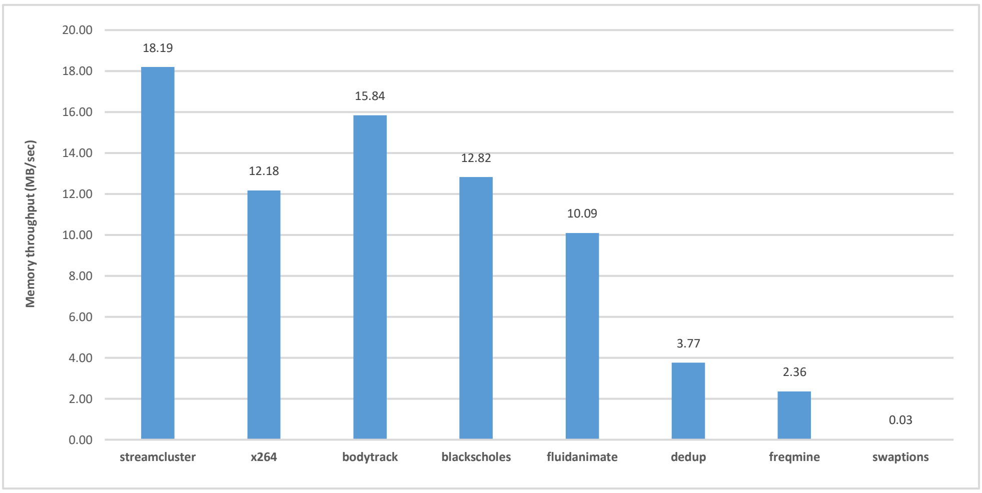


Figure 5 .6: Memory throughput comparison

5.2 Latency Evaluation with SPICE Model

In order to study the latency impacts of tRAS and tRP parameters at lower voltages, we have used 45 nm open-sourced model [58] as a baseline. As our work is based on DDR4 memory, we select 16 nm as a technology for SPICE simulation, mainly for two reasons (a) DDR4 intercept to market is based < 20 nm technology node [82] and (b) PTM has model availability for 16 nm [59]. We leveraged the same DRAM cell array model from [58] and updated the 16 nm technology parameters from PTM [59].

We have selected a 512x512 array as it is a commonly used configuration in modern DRAM chips. The latency of DRAM operations accessing a cell array is significantly influenced by the parasitic resistance and capacitance present on the bit lines and word lines [83]. For our analysis, we assumed specific values for the cell and bit capacitances, namely 24 fF and 144 fF respectively, based on a 45 nm model. To account for scaling trends, we referenced Table 5.1 [81] to estimate the values of the storage capacitor (Cs), storage resistor (Rs), bit-line capacitor (Cb), and bit-line resistor (Rb) for feature sizes ranging from 40 nm to 20 nm. The changes and corresponding values are summarized in Table 5.2.

F, nm	**90**	**70**	**60**	**50**	**40**	**20**	**10**
Rs (Ohm)	210	527	928	1840	4380	1.15×10^5	1.37×10^8
Rb (Ohm)	144	192	228	284	374	932	2600

Table 5.1: Resistances and capacitances in DRAM [41]

- Rs scales very high from 4380 ohm (40nm) to 115 Kohm (20nm), which is ~27x times. We have assumed 150x increase with respect to Cs value of 45 nm, which is 150 Kohm. This very high Rs would impact the latency timings, which is quantified in the simulation.
- Rb scales high from 374 ohm (40nm) to 932 ohm (20nm), which is ~3x times. We have assumed 3x time with respect to Rb of 45 nm, which is 30 Kohm.
- Considering DRAM scaling at traditional pace which is roughly 30% at each node, we have assumed Cs and Cb to be 7 fF and 43 fF respectively [51] for 16 nm. These values are ~70% reductions compared to 45 nm.

Feature	16 nm	45 nm
Storage capacitor (Cs)	24	24
Storage resistor (Rs)	150000	1000
Bit-line capacitor (Cb)	43	144
Bit-line resistor (Rb)	30000	10000

Table 5.2: Summary of resistance and capacitor changes

We have utilized the 16 nm low-power applications (PTM LP) model from PTM, which incorporates high-k/metal gate and stress effects. It is crucial to maintain sufficient storage capacitance and ensure satisfactory performance of the cell transistor in order to preserve the retention time characteristic of a DRAM cell [84]. For the access transistor in the 16 nm model, we have selected moderate dimensions in comparison to the 45 nm model. Specifically, we assumed a length of 0.030 um and a width of 0.220 um for the access transistor, representing reductions of 35% and 40% respectively when compared to the 45 nm access transistor dimensions. Table 5.3 provides a summary of the dimension details for the transistors in the DRAM cell. The access transistors are denoted as M7 and M8, while the dimensions of the sense amplifier and other transistors remain unchanged. To ensure reliable DRAM operation, the following factors are taken into consideration for the bitline measurements to determine the minimum values for tRAS and tRP: (a) the ready-to-precharge voltage, assumed to be 95% of Varray, and (b) the ready-to-activate voltage, assumed to be within 2% of Varray /2. In our revised measurements, we conservatively include the same latency guardband (i.e., 38%) employed by manufacturers for the latency value [43].

16 nm model					**45 nm model**			
FET ID	***Name***	***Length (um)***	***Width (um)***		***FET ID***	***Name***	***Length (um)***	***Width (um)***
M7	nmos16lp	0.030	0.220		M7	nmos45lp	0.085	0.555
M9	pmos16lp	0.27	0.22		M9	pmos45lp	0.27	0.22
M11	pmos16hp	0.16	tw		M11	pmos45hp	0.16	tw
M10	pmos16hp	0.16	tw		M10	pmos45hp	0.16	tw
M4	nmos16hp	0.16	prentw		M4	nmos45hp	0.16	prentw
M5	nmos16hp	0.16	prentw		M5	nmos45hp	0.16	prentw
M6	nmos16hp	0.16	prentw		M6	nmos45hp	0.16	prentw
M12	pmos16hp	0.16	pretw		M12	pmos45hp	0.16	pretw
M13	pmos16hp	0.16	pretw		M13	pmos45hp	0.16	pretw
M3	nmos16hp	0.16	ntw		M3	nmos45hp	0.16	ntw
M2	nmos16hp	0.16	ntw		M2	nmos45hp	0.16	ntw
M1	nmos16lp	0.27	0.22		M1	nmos45lp	0.27	0.22
M8	nmos16lp	0.030	0.220		M8	nmos45lp	0.085	0.555

Table 5.3: Summary of 45 nm and 16 nm transistor dimensions

Table 5.4 lists the latency values obtained from the 16 nm SPICE model, at nominal and reduced voltage points. The latency guardband of 38% is added to account for manufacturing process variation [43].

Voltage	Time taken (ns)		GB (38% added)	
	tRAS	tRP	tRAS	tRP
1.175	19.95	60.04	27.5	13.9
1.150	21.22	60.81	29.3	14.9
1.125	21.50	60.27	29.7	14.2
1.100	21.86	61.00	30.2	15.2
1.075	22.18	60.54	30.6	14.5
1.050	22.50	61.54	31.1	15.9

Table 5.4: Latency values obtained from 16 nm SPICE model simulation

5.3 Proposed architectural changes for practical implementation

Our current work mainly focuses on simulation aspects to quantify energy savings at reduced voltages of self-refresh operation. Validation of the energy savings in real hardware needs a capability to reduce voltage only during self-refresh in DRAM. The existing architecture lacks this ability and hence it becomes a real limitation for validation. However, as a first step, we propose the high-level architecture changes required on DRAM as shown in Figure 5.1, to discuss and motivate for basic implementation.

Our proposed idea requires reduced DRAM array voltage in self-refresh mode to realize energy savings. The extent of voltage reduction purely depends on the design, circuit architectures and process technologies used in the DRAM manufacturing. Hence, each vendor may have a specific voltage point where self-refresh operation can function reliably. We denote this optimal reduced voltage point as VID (Voltage ID).

We propose the scheme as voltage reduced self-refresh (VRSR) mode to be leveraged by the memory controller as an extension to deeper low power modes beyond self-refresh, as shown in Figure 5.8. A mode register set (MRS) register setting in DRAM can determine the mode of self-refresh operation, as shown in Table 5.5. Memory boot initialization flow can program mode register set (MSR) to enable or disable VRSR mode of self-refresh operation. If disabled, the DRAM will select 1.2 V nominal voltage for self-refresh operation. If enabled, the DRAM will select reduced voltage (VID < 1.2 V) for self-refresh operation.

MRx[0]	Self-refresh mode
0	Default (@ 1.2 V)
1	Voltage reduced (@ VID)

Table 5.5: Mode register setting to select self-refresh mode

We have recommended changes on the DRAM architecture to support this setting of voltage and latency parameters, as shown in Figure 5.7. (i) A bit field of a MRS register to determine the voltage setting during self-refresh operation (ii) A voltage generator that generates VID and it is fed to a multiplexer (iii) A multiplexer that has 1.2 V as one input and VID as another input and selects one among them, based on MRS setting (iv) A latency profile selector that holds tRAS and tRP to be applied during self-refresh operation, based on MRS setting.

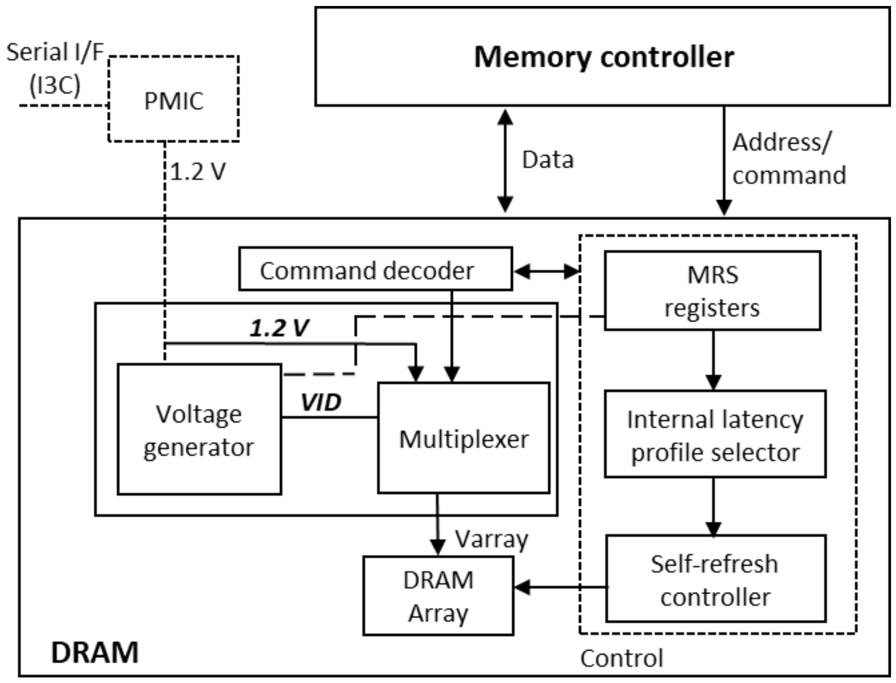


Figure 5.7: Proposed DRAM architecture changes for reduced DRAM voltage operation in self-refresh mode

For DRAMs without internal voltage regulator, an external regulator can deliver voltage, which can be programmed through the serial interface. For DRAMs with internal voltage regulator, the voltage programming can occur with help of MRS register commands along with multiplexer, through the host interface.

MRx[0]=0 enables default mode of self-refresh operation, which is the power-on default setting. For this mode setting, multiplexer will source 1.2 V nominal voltage for Varray. Latency profiler will apply default tRAS and tRP parameters corresponding to 1.2 V. MRx[0]=1 enables voltage reduced mode of self-refresh operation, which is to be set during boot initialization. For this mode setting, multiplexer will source VID for Varray. Latency profiler will apply default tRAS and tRP parameters corresponding to VID.

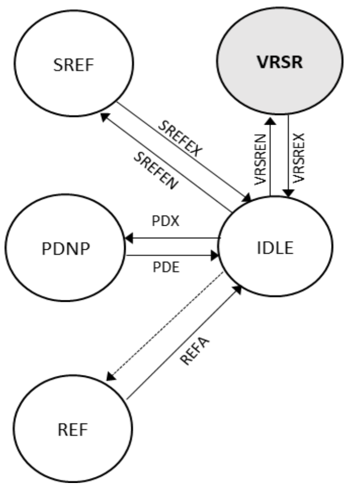


Figure 5.8: State diagram of DRAM commands for low-Power modes, according to JEDEC (proposed modes are highlighted in grey)

During runtime, the controller will select the VRSR mode by sending the entry command (VRSREN). Once the entry command is registered and decoded by DRAM, it will direct the multiplexer to source VID for Varray. It will direct the latency profiler to apply tRAS and tRP corresponding to VID. After this, the self-fresh controller must finish refreshes for all rows at least once, with the modified voltage & latency profiles. This concludes the VRSRx entry operation. Once the exit command is registered and decoded by DRAM, it will direct the multiplexer to revert voltage to 1.2 V for Varray. It will also direct the latency profiler to revert tRAS and tRP corresponding to 1.2 V. This summarizes the VRSR entry operation.

While DRAM is in VRSR mode, the performance of the workloads will not be impacted, as there are no mainline read & write operations involved. However, during exit of VRSR modes, there would be additional latencies associated with restoration of voltage and timing settings back to nominal values and this will impact performance. We have quantified the performance impact seen by the workloads incurred due to exit latency of VRSR modes, in the next section.

5.4 Quantitative evaluation of performance overhead due to voltage reduced self-refresh

In a DRAM, it is necessary to refresh each row once every 64 ms (tRET) within the normal temperature range (< 85 C) according to DDR standards. The time interval between refresh requests to a bank is calculated by dividing 64 ms by the number of rows. In the auto-refresh scheme, the memory controller sends 8192 refresh commands to a DRAM bank at an interval of 7.8 uS (tREFI). The value of tREFI remains constant at 8192, regardless of the number of rows per bank, which has increased significantly with higher device densities. As a result, the DRAM needs to refresh multiple rows (M) for each request. Each refresh request has a duration of the refresh cycle time (tRFC), which includes the time to refresh M rows, precharge the bank, and recover the charge pump (tREC). This can be calculated using equation (5.1). The refresh cycle time for a single row (tRC) is equal to the row activate time (tRAS) plus the row precharge time (tRP).

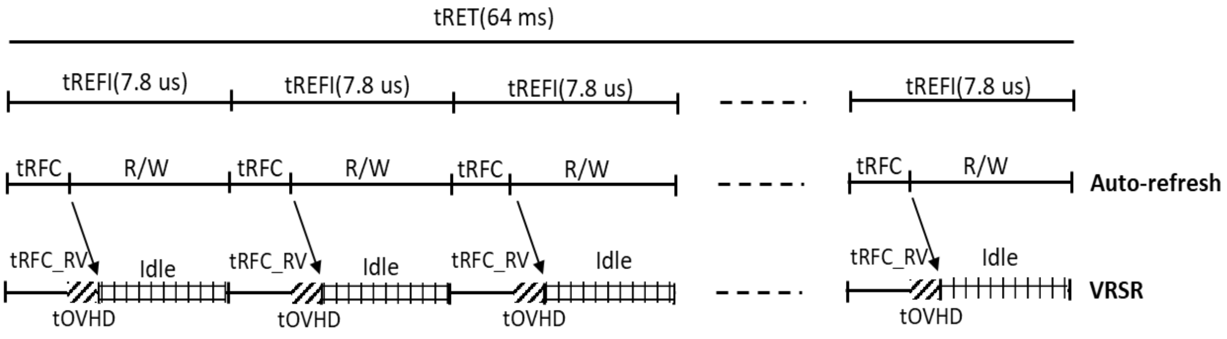


Figure 5.9: Refresh cycle time (tRFC_VR) in VRSR scheme

$$t_{RFC} = (M * t_{RC}) + t_{REC} \tag{5.1}$$

$$t_{RFC\_VR} = \left[\left(M * t_{RC\_VR}\right) + t_{REC}\right] \tag{5.2}$$

Prior work [71] found that multiple rows in different subarrays, are refreshed in parallel when M > 4. Therefore, there is a time compression (tCMPR) achieved through this implementation that optimizes the overall refresh time. For quantitative evaluation of the refresh overhead due to increased latencies of tRAS and tRP in VRSR scheme, the auto-refresh scheme is considered as a baseline to derive some key insights and tCMPR. Refresh cycle time (tRFC_VR) due to reduced voltage accounting for tCMPR can be given as per equation (5.2), where tRC_VR is the row cycle (tRAS_VR + tRP_VR) at reduced voltage).

Voltage (V)	tRAS_VR (ns)	tRP_VR (ns)	tRC_VR (ns)	tRFC_VR (ns)
1.175	27.5	13.9	41.4	391
1.150	29.3	14.9	44.2	414
1.125	29.7	14.2	43.8	411
1.100	30.2	15.2	45.3	423
1.075	30.6	14.5	45.2	421
1.050	31.1	15.9	47.0	436

Table 5.6: Refresh cycle time for reduced voltages

For DDR4-1600 (17-17-17) device used in our evaluation, tRFC calculated is 430 ns, given tRAS and tRP to be 32 ns and 13.75 ns respectively [42] and tREC is considered to be 60 ns. However, actual tRFC is 350 ns as per the device datasheet [42]. It shows that the parallel refresh scheme compresses the overall time by 18%, compared to serial refresh. Table 5.6 shows the increased refresh cycle time (tRFC_VR) due to reduced voltage level as per equation (5.2).

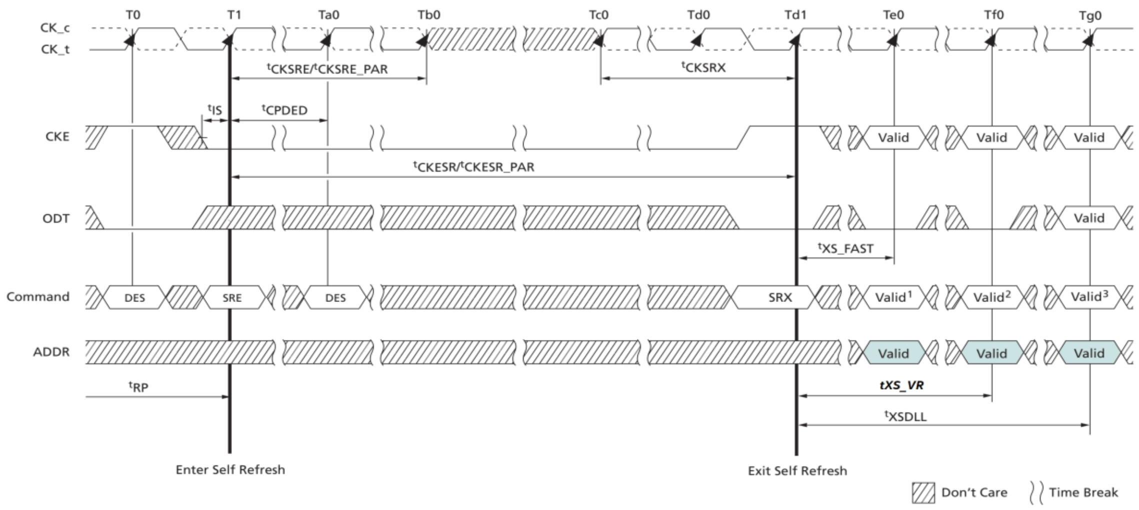


Figure 5.10: Exit timing latency of VRSR (tXS_VR) with reference to self-refresh [42]

The increased delay in tRFC_VR adds time overhead (tOVHD) in the overall refresh cycle. It is due to additional latency cycles in row refresh time (tRC_VR) of M rows at reduced voltage compared to refresh time at 1.2V nominal voltage (tRC), as shown in equation (5.3).

$$t_{OVHD} = \big(M * (t_{RC\_VR} - t_{RC})\big) * t_{CMPR} \tag{5.3}$$

Vendors do not provide detailed information about the specific implementation of their auto-refresh or self-refresh schemes. Figure 5.9 illustrates the analysis of performance overhead for the VRSR scheme, comparing it to the auto-refresh protocol and timing specifications.

Auto-refresh operations can impact regular memory operations and result in performance degradation. On the other hand, with self-refresh, the DRAM internally handles refresh operations without requiring any intervention from the memory controller.

Voltage (V)	**tXS_VR (ns)**
1.175	406
1.150	429
1.125	426
1.100	438
1.075	436
1.050	451

Table 5.7: Refresh cycle time and exit latency for reduced voltages

VRSR is a variant of self-refresh, but with reduced voltage level and increased timing latencies (tRC_VR) to allow row activation and pre-charge operations to complete at such reduced voltage. As there are no read & write operations involved, the overhead (tOVHD) due to these increased latencies will not have impact any performance impact. However, tOVHD will have impact on the exit latency.

The impact of VRSR on exit latency is evaluated by comparing it to the baseline of self-refresh exit latency, as shown in Figure 5.10. In order to enter self-refresh mode, the memory controller must ensure that the device is idle with all banks in the precharge state and tRP satisfied. Once the self-refresh entry command (SRE) is issued, the device remains in self-refresh mode as long as CKE is held low, with a minimum pulse width of tCKESR. To exit self-refresh mode, a sequence of events must occur. First, the clock must be stable before CKE is raised back to HIGH. Once the self-refresh exit command (SRX) is registered, the timing delay of tXS must be satisfied. When using self-refresh mode, there is a possibility of missing an internally timed refresh event when CKE is raised for exit. After exiting self-refresh mode, the device requires a minimum of one additional REFRESH command before it can be put back into self-refresh mode. Therefore, the exit latency, tXS, is defined as "tRFC + 10 ns" according to [42]. In the case of VRSR, the device requires 2 tVCHNG transitions to reduce voltage before the refresh begins, and then revert the voltage back to nominal (1.2V) once the refresh is complete. Therefore, the required exit latency for VRSR is determined by the equation (5.4).

$$t_{XS\_VR} = t_{RFC\_VR} + 2\, t_{VCHNG} + 10\, ns \qquad (5.4)$$

tVCHNG requires Varray to be switched at the output of the multiplexer as shown in Figure 5.1 and is considered as 2.5 ns. Table 5.7 shows the exit latency for reduced voltage level calculated as per the equation (5.4).

We have evaluated the performance (execution time of benchmarks) impact due to increased tXS_VR latency at lowest voltage (1.050 V) baselined to nominal voltage (1.2 V) in section IV.

5.1 Results and Discussion

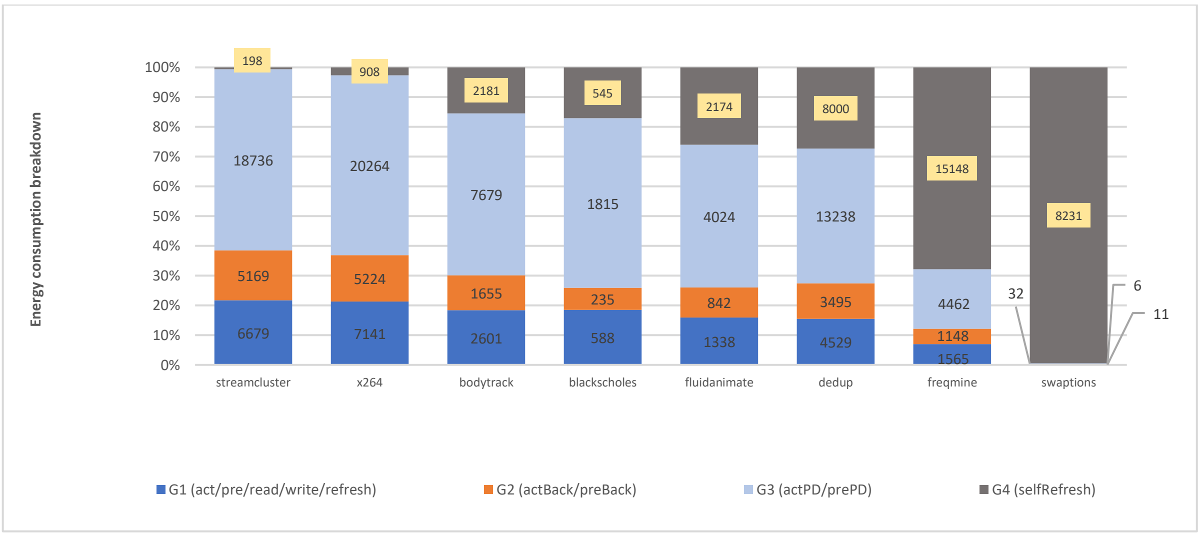


Figure 5.11: DRAM energy breakdown for PARSEC benchmarks

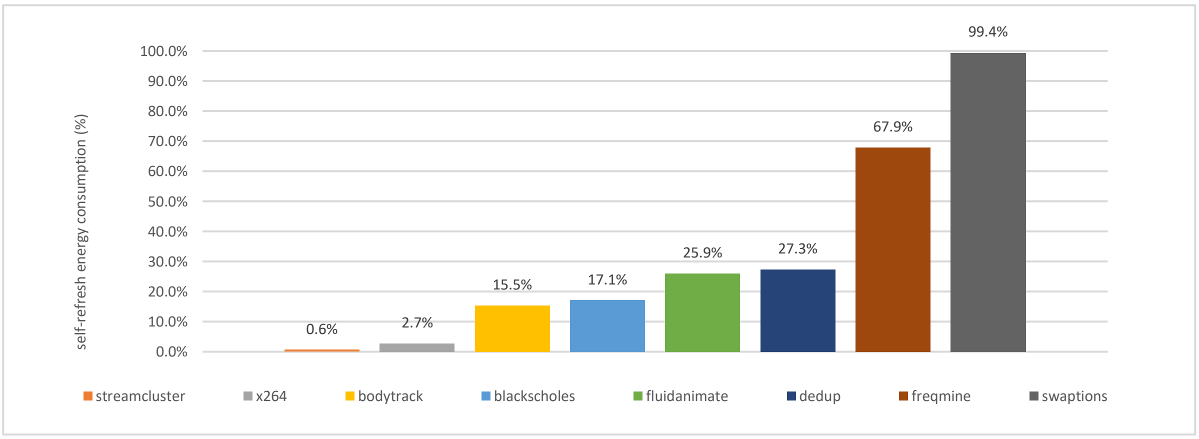


Figure 5.12: Self-refresh energy (%) contribution to total energy

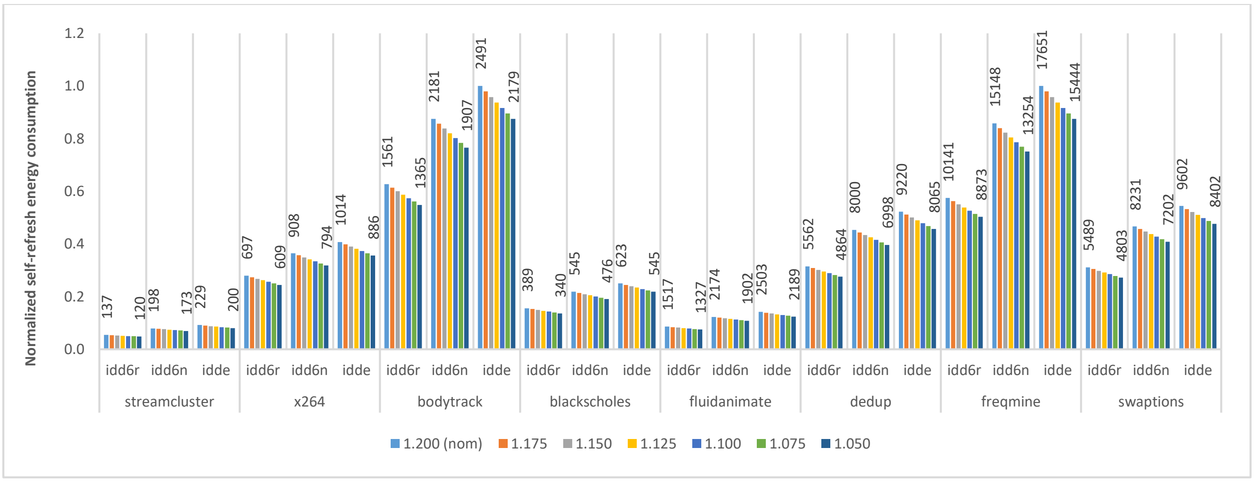


Figure 5.13: Self-refresh energy at reduced voltages and different temperature ranges

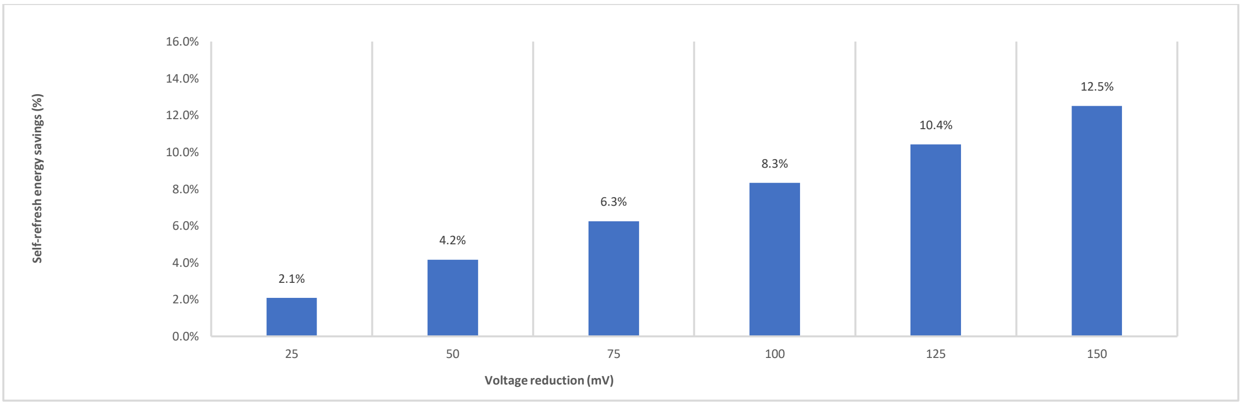


Figure 5.14: Self-refresh energy savings with voltage reduction

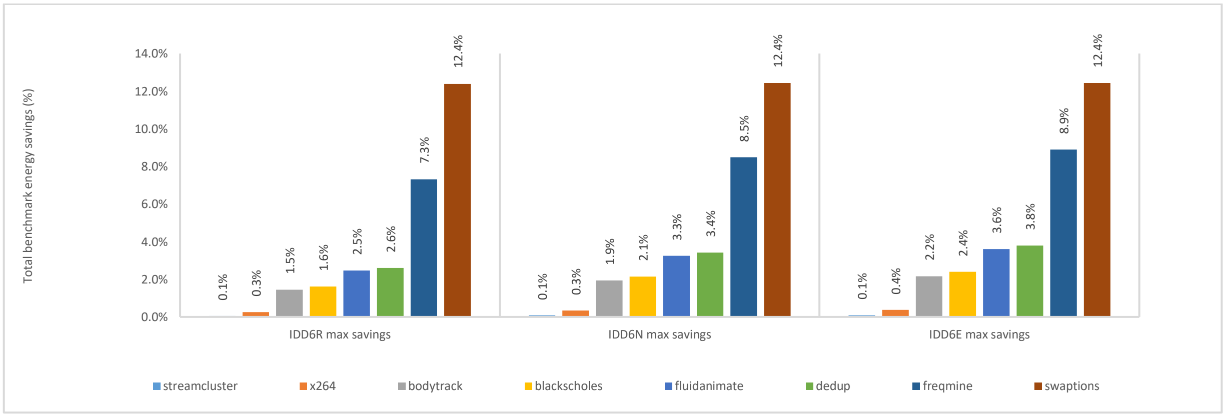


Figure 5.15: Maximum benchmark energy savings

Metrics	**streamcluster**	**x264**	**bodytrack**	**blackscholes**	**fluidanimate**	**dedup**	**freqmine**	**swaptions**
1.200 V	32.639	37.142	16.860	4.144	10.844	38.003	34.127	14.344
1.050 V	32.772	37.386	16.903	4.152	10.855	38.103	34.194	14.347
Time Increase (%)	*0.41%*	*0.66%*	*0.25%*	*0.18%*	*0.10%*	*0.26%*	*0.20%*	*0.02%*
Energy savings (Max)	*0.1%*	*0.4%*	*2.2%*	*2.4%*	*3.6%*	*3.8%*	*8.9%*	*12.4%*

Table 5.8: Execution time increase of benchmarks at reduced voltage (1.050 V) compared to nominal voltage (1.2 V)

The analysis of energy consumption across various benchmarks reveals a nuanced landscape of self-refresh energy (G4) usage in relation to other energy components (G1/G2/G3). This differentiation is crucial for understanding the energy dynamics within DRAM operations. Specifically, the benchmark named streamcluster exhibits the lowest G4 consumption at 198 millijoules (mJ), contributing minimally to its total energy footprint of 20,782 mJ. Conversely, swaptions demonstrates a stark contrast with a G4 consumption of 8,231 mJ, nearly matching its total energy usage of 8,280 mJ. This stark variance in G4 consumption across benchmarks is visually represented in Figure 5.11, providing a clear comparative analysis.

Benchmarks	**IDD6R**	**IDD6N**	**IDD6E**
streamcluster	17	25	29
x264	88	115	128
bodytrack	196	274	313
blackscholes	49	68	78
fluidanimate	190	273	314
Dedup	698	1003	1155
freqmine	1268	1894	2207
swaptions	686	1029	1200

Table 5.9: Maximum self-refresh energy savings (mJ) at 150 mV voltage reduction

Delving deeper into the energy consumption patterns, Figure 5.12 offers a graphical representation of the G4 to total DRAM energy ratio for each benchmark. The data points range significantly, with streamcluster at the lower end, contributing only 0.6% to its total energy, and swaptions at the upper end, with a staggering 99.4% contribution. This wide spectrum of G4 consumption ratios underscores the potential for energy savings across diverse workload scenarios when implementing reduced voltage strategies, as proposed in our approach.

Figure 5.13 further explores the energy savings landscape by showcasing the normalized self-refresh consumption savings across three distinct temperature ranges (IDDR/IDD6N/IDD6E) for each benchmark at five reduced voltage points, cumulatively amounting to a 150 mV reduction from a nominal 1.2 V, in 25 mV increments. This figure not only highlights the absolute energy values at 1.2 V and 1.05 V but also elucidates the consumption patterns across temperature ranges, attributing the highest, moderate, and lowest energy consumption to IDD6E, IDD6N, and IDDR, respectively. The adaptive refresh rate requirement, which varies with operating temperature, is a key factor influencing these patterns. Specifically, IDD6E exhibits 12% to 17% higher consumption at elevated temperatures, while IDD6R demonstrates 23% to 33% lower consumption at reduced temperatures, compared to IDD6N.

The efficacy of voltage reduction as a strategy for energy savings is quantitatively assessed in Figure 5.14, which illustrates the percentage of G4 savings achieved at various reduced voltages, benchmarked against the G4 consumption at a nominal 1.2 V. The findings indicate that even a minimal reduction of 25 mV can yield a 2.1% savings, with the savings potential escalating to 12.5% at a 150 mV reduction.

Expanding on the theme of energy savings, Figure 5.15 categorizes the benchmarks based on their maximum G4 savings at an aggressive 150 mV voltage reduction, relative to the energy consumption at 1.2 V. The benchmarks are segmented into three groups based on their savings potential: low (below 1%), moderate (between 1% and 4%), and high (9% and 12%). This segmentation reveals an average G4 savings of 3.5%, 4.0%, and 4.2% for IDD6R, IDD6N, and IDD6E, respectively, across all benchmarks.

The performance implications of voltage reduction are meticulously documented in Table 5.8, which compares the execution times of benchmarks at the lowest voltage of 1.050 V against the nominal voltage of 1.2 V. The performance impact is remarkably minimal, with swaptions experiencing a negligible increase of 0.02%, and x264 facing the highest impact of 0.66%. This uniform performance stability, within a 0.7% range for a substantial 150 mV reduction, coupled with the observed energy savings, highlights the feasibility of achieving significant energy efficiency with minimal performance trade-offs, particularly for benchmarks like freqmine and swaptions.

Lastly, Table 5.9 consolidates the absolute G4 energy savings across all benchmarks, reinforcing the conclusion that workloads with a substantial proportion of self-refresh energy consumption stand to benefit markedly from reduced voltage strategies, thereby achieving considerable energy savings. This analysis not only underscores the potential for energy efficiency improvements but also emphasizes the importance of tailored voltage reduction strategies to optimize both energy savings and performance across diverse workload scenarios.

Chapter 6

Conclusion and Future Work

6.1 Architecture Slack Exploitation for Phase Classification and Performance Estimation

In this work, we contributed to the following:

(a) We proposed a novel metric to measure architecture slack of workloads and evaluated its application for workload classification over a broader frequency range of 3.5 GHz, using micro benchmarks.

(b) We developed performance estimation models based on this metric and evaluated their prediction accuracy, using SPECCPU 2006 and PARSEC benchmarks, at 2 GHz and 3.5 GHz frequency bounds. It is promising to observe the prediction accuracy is 97% for all benchmarks. We realize that the performance models were regression based which needed offline profiling of benchmarks. We used 19 benchmarks for training the models. These models provided very high prediction accuracy for 19 remaining benchmarks (unseen workloads) and this highly encourages to explore further on realizing accurate online estimation models. As memory phases of workloads possesses significant architecture slack, they can be aggressively exploited for energy savings. It also depends on how much user-demand slack is available to be exploited i.e., a user or software workload manager to specify a performance floor as a percentage of the workload's performance at the maximum processor frequency. With the exception of extremely memory-bound applications, the absence of user-demand slack disallows exploitation of architecture slack.

(c) We also expanded application of this metric for online phase classification of benchmarks having different phase characteristics.

(d) We developed an algorithm to guide dynamic frequency scaling decisions during execution and evaluated runtime impacts of all benchmarks.

In summary, we have developed methods based on architecture slack as key metric, which can be adopted by newer DVFS algorithms for phase classification and performance estimation at runtime. We acknowledge that there is a need for further research and development in the area of online prediction models and algorithms that can effectively utilize architecture slack while considering user-demand slack requirements. In our future exploration, we plan to evaluate performance management along with energy saving benefits, with HPC and server workloads.

6.2 Voltage Reduced Self Refresh (VRSR) for Optimized Energy Savings in DRAM Memories

This paper provided a first detailed study of self-refresh energy savings at reduced voltages. Self-refresh energy savings was quantified with 8 PARSEC benchmarks using DDR4 DRAMs, at six reduced voltage points (up to 150 mV reduction from 1.2 V nominal) and different refresh rates pertaining to broader temperature ranges. The latency increase of row activation and precharge operations was evaluated using 16 nm model. We measured execution time of benchmarks and evaluated the performance impact due to additional latencies, at lowest setting of 1.05 V baselined to 1.2 V nominal voltage.

We used Gem full-system simulations for measurements of energy savings and performance. Our simulation results demonstrated that there is a maximum of ~12.4% workload energy savings realized with one benchmark and an average of ~4% workload energy savings across all benchmarks, for an aggressive voltage reduction of 150 mV. This finding revealed that the workload energy savings clearly depends on two key factors. First is the extent of voltage reduction and second is the proportion of self-refresh energy to the total energy for a given workload. It also revealed that that the performance increase due to increased row activation and precharge latencies, was well within 1% for the lowest setpoint at 1.05 V (150 mV reduction), across all benchmarks.

Finally, some key limitations in extending our work to real hardware have also been discussed. Further to motivate researchers in order to realize a full-scale solution in future, we proposed DRAM architectural changes and additional power modes to exercise reduced voltage operation in self-refresh mode. Combining both, we introduced this new DRAM lower power mode as Voltage Reduced Self-Refresh (VRSR) operation and the simulation results demonstrated significant energy savings with the proposed approach.

Bibliography

1) Shoaib Akram, Jennifer B. Sartor, Lieven Eeckhout, DVFS performance prediction for managed multithreaded applications, in: ISPASS 2016, 2016, pp. 12–23.

2) Basireddy Karunakar Reddy, D. Singh, G.V. Biswas, B. Merrett, Inter-cluster thread-to-core mapping and DVFS on heterogeneous multi-cores, IEEE Trans. Multi-Scale Comput. Syst. 4 (3) (2017) 369–382.

3) James M. Brandt, HPC monitoring & analysis + power 9 specifics, in: Conference Presentation at the Oak Ridge National Lab Monitoring, August 2019, https://www.osti.gov/servlets/purl/1645803.

4) M. Broyles, C.J. Cain, T. Rosedahl, G.J. Silva, IBM EnergyScale for POWER8 processor-based systems, IBM, Tech. Rep., Nov. 2015.

5) Hari Cherupalli, Rakesh Kumar, John Sartori, Exploiting dynamic timing slack for energy efficiency in ultra-low-power embedded systems, in: 2016 43th Annual International Symposium on Computer Architecture, ISCA, IEEE, 2016.

6) G.L.T. Chetsa, L. Lefevre, J.-M. Pierson, P. Stolf, G. DaCosta, A user friendly phase detection methodology for HPC systems' analysis, in: Proc. IEEE Int. Conf. Green Comput. Commun. IEEE Internet Things IEEE Cyber Phys. Social Comput., 2013, pp. 118–125.

7) G. Contreras, M. Martonosi, Power prediction of intel xscale processors using performance monitoring unit events, in: 2005 International Symposium on Low Power Electronics and Design, August 2005.

8) Keeley Criswell, Tosiron Adegbija, A survey of phase classification techniques for characterizing variable application behavior, IEEE Trans. Parallel Distrib. Syst. 31 (1) (Jan. 1 2020).

9) Rado Danilak, Why energy is a big and rapidly growing problem for data centers, https://www.forbes.com/sites/forbestechcouncil/2017/12/15/why-energyis-a-big-and-rapidly-growing-problem-for-data-centers, December 2017.

10) B. Fields, R. Bodik, M. Hill, Slack: maximizing performance under technological constraints, in: Proceedings of the 29th International Symposium on Computer Architecture, ISCA 2002, May 2002.

11) Heather L. Hanson, Venkat R. Indukuru, Francis P. O'Connell, Karthick Rajamani, Tracking pipelined activity during off-core memory accesses to evaluate the impact of processor core frequency changes, U.S. Patent 9600392 B2, Available: https://patents.google.com/patent/US9600392B2, Mar. 21, 2017.

12) Chung-Hsing Hsu, Wu-Chun Feng, Effective dynamic voltage scaling through CPU-boundedness detection, in: Power-Aware Computer Systems, PACS, 2004.

13) IBM power systems E870 and E880, Technical overview and introduction (REDP-5137-00), An IBM Redpaper publication, http://www.redbooks.ibm.com/abstracts/redp5137.html.

14) Canturk Isci, Margaret Martonosi, Phase characterization for power: evaluating control-flow-based and event-counter-based techniques, in: Proc. 12th IEEE Int. Symp. High Perf. Comput. Arch., HPCA-12, February 2006, pp. 121–132.

15) Canturk Isci, Gilberto Contreras, Margaret Martonosi, Live, runtime phase monitoring and prediction on real systems with application to dynamic power management, in: Proc. 39th ACM/IEEE Int. Symp. Microarch., MICRO-39, December 2006, pp. 359–370.

16) R. Kotla, S. Ghiasi, T. Keller, F. Rawson, Scheduling processor voltage and frequency in server and cluster systems, in: Proc. IEEE Int. Parallel Distrib. Process. Symp., vol. 12, 2005, 234b.

17) Sang Jeong Lee, Hae-Kag Lee, Pen-Chung Yew, Runtime performance projection model for dynamic power management, in: Advances in Comput. Syst. Arch. 12th Asia-Pacific Conf. (ACSAC 2007) Proc., in: Lecture Notes in Computer Science, vol. 4697, Springer-Verlag, August 2007, pp. 186–197.

18) W. Liang, S. Chen, Y. Chang, J. Fang, Memory-aware dynamic voltage and frequency prediction for portable devices, in: 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, 2008, RTCSA ’08, Aug. 2008, pp. 229–236.

19) Ankur Limaye, Tosiron Adegbija, A workload characterization of the SPEC CPU2017 benchmark suite, in: Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS’18, 2018, pp. 149–158.

20) Ali Marashi, Improving data center power consumption & energy efficiency, https://www.vxchnge.com/blog/growing-energy-demands-of-data-centers, February 2020.

21) D. Molka, R. Schone, D. Hackenberg, W.E. Nagel, Detecting memory boundedness with hardware performance counters, in: Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering, ser. ICPE 17, Association for Computing Machinery, New York, NY, USA, 2017, p. 2738.

22) OpenPOWER, On chip controller (OCC), http://openpowerfoundation.org/blogs/on-chip-controller-occ/.

23) PARSEC, https://parsec.cs.princeton.edu/.

24) K. Rajamani, H. Hanson, J. Rubio, S. Ghiasi, F. Rawson, Application-aware power management, in: Proceedings of the 2006 IEEE International Symposium of Workload Characterization, IISWC-2006, October 2006.

25) R. Rodrigues, et al., Improving performance per watt of asymmetric multi-core processors via online program phase classification and adaptive core morphing, ACM Trans. Des. Autom. Electron. Syst. 18 (1) (Jan. 2013) 5:1–5:23.

26) Todd Rosedahl, Charles Lefurgy, POWER8 on chip controller, Online, HPC power management: knowledge discovery, http://hpm.ornl.gov/documents/HPM2015Rosedahl.pdf, September 2015.

27) R. Schöne, D. Hackenberg, On-line analysis of hardware performance events for workload

characterization and processor frequency scaling decisions, in: Proceeding of the Second Joint WOSP/SIPEW International Conference on Performance Engineering, ICPE '11, ACM, 2011.

28) Dror Shenkar, Shahar Belkin, Data centers feeling the heat! The history and future of data center cooling, https://datacenterfrontier.com/history-future-datacenter-cooling, May 2020.

29) Timothy Sherwood, Suleyman Sair, Brad Calder, Phase tracking and prediction, in: Proc. 30th Int. Symp. Comput. Arch., ISCA 2003, San Diego, California, June 2003, pp. 336–347.

30) B. Sinharoy, J.A. Van Norstrand, R.J. Eickemeyer, H.Q. Le, J. Leenstra, D.Q. Nguyen, B. Konigsburg, K. Ward, M.D. Brown, J.E. Moreira, D. Levitan, S. Tung, D. Hrusecky, J.W. Bishop, M. Gschwind, M. Boersma, M. Kroener, M. Kaltenbach,T. Karkhanis, K.M. Fernsler, IBM POWER8 processor core microarchitecture, IBM J. Res. Dev. 59 (2015) 2:1–2:21.

31) SPEC CPU2006, http://www.spec.org/cpu2006.

32) V. Spiliopoulos, S. Kaxiras, G. Keramidas, Green governors: a framework for continuously adaptive DVFS, in: 2011 International Green Computing Conference and Workshops, IGCC, July 2011, pp. 1–8.

33) Sharanyan Srikanthan, Sandhya Dwarkadas, Kai Shen, Coherence stalls or latency tolerance: informed CPU scheduling for socket and core sharing, in: Proceedings of USENIX Annual Technical Conference, USENIX ATC, 2016.

34) S. Srinivasan, I. Koren, S. Kundu, Improving performance per watt of nonmonotonic multicore processors via bottleneck based online program phase classification, in: Proc. IEEE 34th Int. Conf. Comput. Des., 2016, pp. 528–535.

35) U.S. Department of Energy, Data center energy efficiency, https://www.energy.gov/sites/default/files/2019/03/f60/femp-data-center-energy-efficiency.pdf,February 2019.

36) U.S. Environmental Protection Agency, ENERGY STAR program requirements for computer servers,

https://www.energystar.gov/sites/default/files/ENERGY%20STAR%20Version%203.0%20Computer%20Servers%20Program%20Requirements_0.pdf, September 2018.

37) Frederik Vandeputte, Lieven Eeckhout, Koen De Bosschere, A detailed study on phase predictors, in: Proc. 11th Int. Euro-Par Conf. Parallel Process., Euro-Par 2005, August 2005, pp. 571–581.

38) David Wood, Rathijit Sen, Article, energy-proportional computing: a new definition, https://www.researchgate.net/publication/318844704_EnergyProportional_Computing_A_New_Definition, January 2017.

39) International Energy Agency. (2020). The Growing Role of Data Centres in Electricity Demand and Carbon Emissions.

40) Vakilinia, A., & Buyya, R. (2021). Energy consumption and carbon dioxide emissions of cloud computing: A review. ACM Computing Surveys, 54(1), 1-38.

41) I. Bhati et al., "DRAM Refresh Mechanisms, Penalties, and Trade-Offs", IEEE Transactions on Computers, vol. 65, no. 1, pp. 108–121, Jan 2016.

42) Micron Technology, "8Gb: x4, x8, x16 DDR4 SDRAM Datasheet", Tech. Rep. MT40A2G4, 2015.

43) K. K. Chang et al., "Understanding Reduced-Voltage Operation in Modern DRAM Devices: Experimental Characterization, Analysis, and Mechanisms", SIGMETRICS, 2017.

44) Michael D. Pardeik et al., "Implementing DRAM refresh power optimization during long idle mode", U.S. Patent, 10304501, 2019.

45) B. Keeth et al., "DRAM Circuit Design: A Tutorial", Wiley, 2001.

46) Tadaaki Yamauchi et al., "Design of High-Performance Microprocessor Circuits", IEEE Press, Piscataway NJ, 2001.

47) Micron Technology, "TN-46-12. Mobile dram power-saving features and power calculations", Tech. Note. TN-46-12, 2009.

48) Micron Technology, “4Gb DDR4 SDRAM Datasheet”, MT40A1G4, 2020.

49) Micron Technology, “16Gb DDR4 SDRAM Datasheet”, MT40A1G16, 2018.

50) Micron Technology, “32Gb DDR4 SDRAM Datasheet”, MT40A2G16, 2019.

51) Winbond, “Combining deep power-down with self-refresh mode”, Technical Article, 2018.

52) Nathan Binkert et al., “The Gem5 simulator”, ACM SIGARCH Computer Architecture News, May 2011

53) Chandrasekar, K. et al., “Improved Power Modeling of DDR SDRAMs”, In Proceedings of the 2011 14th Euromicro Conference on Digital System Design, Oulu, Finland, pp. 99–108, 31 August–2 September 2011

54) Karthik Chandrasekar et al., “DRAMPower: Opensource DRAM power & energy estimation tool”, http://www.drampower.info

55) M. Jung et al., “Optimized active and power-down mode refresh control in 3D-DRAMs”, In VLSI-SoC, 2014 22nd International Conference on. 1–6. https://doi.org/10.1109/VLSI-SoC.2014.7004159.

56) Radhika Jagtap et al., “Integrating DRAM power-down modes in gem5 and quantifying their impact”, In Proceedings of the International Symposium on Memory Systems. ACM, 86–95.

57) L. A. Barroso and U. Holzle, “The datacenter as a computer: An introduction to the design of warehouse-scale machines,” Synthesis lectures on computer architecture, vol. 4, no. 1, pp. 1–108, 2009.

58) “DRAM Voltage Study”, https://github.com/CMU-SAFARI/ DRAM-Voltage-Study, 2017.

59) “Predictive Technology Model”, 2012.

60) Byoungchan Oh et al., “Enhancing DRAM Self-Refresh for Idle Power Reduction”, in ISLPED, 2016.

61) Jamie et al., “RAIDR: Retention-Aware Intelligent DRAM Refresh”, in ISCA, 2012.

62) Jin-Hong Ahn et al. “Adaptive self-refresh scheme for battery operated high-density mobile DRAM applications”, in ASSCC, 2006.

63) Jamie Liu et al., “An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms”, in ISCA, 2013.

64) S. Khan et al., “The efficacy of error mitigation techniques for DRAM retention failures: A comparative experimental study”, in SIGMETRICS, 2014.

65) Moinuddin Qureshi et al., “AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems”, in DSN, 2015.

66) Minesh Patel et al., “The Reach Profiler (REAPER): Enabling the Mitigation of DRAM Retention Failures via Profiling at Aggressive Conditions”, in ISCA, 2017.

67) Anup Das et al., “VRL-DRAM: Improving DRAM Performance via Variable Refresh Latency”, in DAC, 2018.

68) Song Liu et al., “Flikker: Saving DRAM Refresh-Power through Critical Data Partitioning”, in ASPLOS 2011.

69) Dongli Zhang et al., “DIMMer: A Case for Turning Off DIMMs in Clouds”, in SoCC, 2014.

70) R. K. Venkatesan et al., “Retention-aware placement in DRAM (RAPID): software methods for quasi-non-volatile DRAM”, in HPCA, 2006.

71) Aditya Agrawal et al., “CLARA: Circular Linked-List Auto and Self Refresh Architecture”, in MEMSYS, 2016.

72) Matthias Jung et al., “Optimized active and power-down mode refresh control in 3D-DRAMs”, in VLSI-SoC, 2014.

73) Konstantinos Tovletoglou et al., “Relaxing DRAM refresh rate through access pattern scheduling: A case study on stencil-based algorithms”, in IOLTS, 2017.

74) Xing Pan et al., “The Colored Refresh Server for DRAM”, in RTSS, 2019.

75) Eder Zulian et al., “Access-Aware Per-Bank DRAM Refresh for Reduced DRAM Refresh Overhead”, in ISCAS, 2020.

76) Yuhai Cao et al., “DR Refresh: Releasing DRAM Potential by Enabling Read Accesses Under Refresh”, in IEEE TC, 2019.

77) Hongtao Zhong et al., “One-Shot Refresh: A Low-Power Low-Congestion Approach for Dynamic Memories”, in IEEE TCS, 2020.

78) Hoseok Seol et al., "Elaborate Refresh: A Fine Granularity Retention Management for Deep Submicron DRAMs", in IEEE TC, 2018.

79) "JEDEC standard for DDR/DDR2/DDR3/DDR3L/DDR4 SDRAM. ", http://www.jedec.org/standards-documents.

80) D. Shim et al. , “A process-variation-tolerant on-chip CMOS thermometer for auto temperature compensated self-refresh of low-power mobile DRAM. ”, Solid-State Circuits, IEEE Journal of 2013.

81) V. V. Zhirnov and M. J. Marinella, “Memory Technologies: Status and Perspectives”, in Emerging Nanoelectronic Devices, Wiley 2015.

82) “Micron D1α, '14 nm'! The Most Advanced Node Ever on DRAM!”, Semiconductor engineering, Article 2018, https://semiengineering.com/micron-d1%CE%B1-the-most-advanced-node-yet-on-dram/

83) T. Vogelsang, “Understanding the Energy Consumption of Dynamic Random Access Memories,” in MICRO, 2010.

84) “International Roadmap for Devices and Systems, 2022”, https://irds.ieee.org/images/files/pdf/2022/2022IRDS_ES.pdf

85) Kamiya George, “Data Centres and Data Transmission Networks.”, International Energy Association, November 2021, https://www.iea.org/reports/data-centres-and-data-transmission-networks

86) Ben Kepes, “30% Of Servers Are Sitting "Comatose" According To Research”, Jun 2015, https://www.forbes.com/sites/benkepes/2015/06/03/30-of-servers-are-sitting-comatose-according-to-research/?sh=7c0448f959c7

87) Katal, A., Dahiya, S. & Choudhury, T., “Energy efficiency in cloud computing data center: a survey on hardware technologies.”, Cluster Comput 25, 675–705 (2022). https://doi.org/10.1007/s10586-021-03431

88) R. Elmore, K. Gruchalla, C. Phillips, A. Purkayastha, and N. Wunder, “Analysis of application power and schedule composition in a high performance computing environment,” tech. rep., National Renewable Energy Lab.(NREL), Golden, CO (United States), 2016.

List of Publications

[1] **Diyanesh Chinnakkonda**, Karthick Rajamani, M.B. Srinivas. “Architecture slack exploitation for phase classification and performance estimation in server-class processors,” in Journal of Parallel and Distributed Computing, Elsevier, Volume 169, 2022, Pages 157-170, ISSN 0743-7315, https://doi.org/10.1016/j.jpdc.2022.06.017. (https://www.sciencedirect.com/science/article/pii/S0743731522001526).

[2] **Diyanesh Chinnakkonda,** Venkata Kalyan Tavva, and M. B. Srinivas. "Voltage Reduced Self Refresh (VRSR) for Optimized Energy Savings in DRAM Memories", in Memories - Materials, Devices, Circuits and Systems, Elsevier.
Article published – DOI: https://doi.org/10.1016/j.memori.2023.100058

Brief Biography of the Candidate

Diyanesh Chinnakkonda is currently pursuing his Ph.D. in the Department of Electrical and Electronics Engineering at Birla Institute of Technology and Science (BITS) University, Pilani. Prior to this he was an Advisory Research Engineer in the POWER server systems development team, part of India Systems Development Lab, IBM India Pvt. Ltd. Bangalore. He received his M.S. degree from BITS University in 2004. His current research interest includes computer architecture, energy & performance management for cloud & next generation computing and future memory technologies. He holds 123 issued patents and has published papers in IEEE conferences.

Brief Biography of the Supervisor

Dr. M.B. Srinivas received his Ph.D from the Indian Institute of Science, Bangalore, India. He is currently a Professor in the Department of Electrical and Electronics Engineering at Birla Institute of Technology and Science (BITS) University, Pilani. His research interests include nano-scale circuit design, VLSI arithmetic, data converters and ICT for healthcare. He is an active member of IEEE and has served as Chairman of IEEE Hyderabad Section during 2007 and 2008 and founding Chairman CAS/EDS Joint Chapter, Hyderabad Section, during 2012 and 2013. He is a recipient of Microsoft Research 'Digital Inclusion' award in 2006 and Stanford Medicine 'MedTech Innovation' award in 2016.

Brief Biography of the Co-Supervisor

Subhendu Kumar Sahoo completed his B.E. in Electronics and telecommunication engineering from Utakal University, Odisha, India, with honors securing the fifth position in the university. He obtained his M.E. in Electronic Systems and Communication from R.E.C.(NIT) Rourkela. He received the Ph. D. degree in electrical engineering from Birla Institute of Technology and Science, Pilani, in 2006. He is working as a faculty in the Electrical and Electronics Engineering department of Birla Institute of Technology and Science, Pilani, since 2006. Presently he is a Professor at Birla Institute of Technology and Science, Pilani-Hyderabad Campus, India. His research areas are high-performance arithmetic circuits, VLSI Circuits, and architecture for Digital Signal Processing applications.