INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI

Energy and Thermal Management of CMPs by Dynamic Cache Reconfiguration

by

Shounak Chakraborty

A thesis submitted in partial fulfillment for the degree of Doctor of Philosophy

in the

Department of Computer Science and Engineering

Under the supervision of

Prof. Hemangee K. Kapoor

February 2018

Declaration of Authorship

I, Shounak Chakraborty, hereby confirm that:

- The work contained in this thesis is original and has been done by myself under the general supervision of my supervisor.
- This work has not been submitted to any other Institute for any degree or diploma.
- Whenever I have used materials (data, theoretical analysis, results) from other sources, I have given due credit to the authors/researchers by citing them in the text of the thesis and giving their details in the reference.
- Whenever I have quoted from the work of others, the source is always given.

Shounak Chakraborty
Research Scholar,
Department of CSE,
Indian Institute of Technology Guwahati,
Guwahati, Assam, INDIA 781039,
c.shounak@iitg.ernet.in, shou46@gmail.com

Date: February 28, 2018
Place: IIT Guwahati

Certificate

This is to certify that the thesis entitled **"Energy and Thermal Management of CMPs by Dynamic Cache Reconfiguration"** being submitted by **Mr. Shounak Chakraborty** to the department of *Computer science and Engineering, Indian Institute of Technology Guwahati*, is a record of bonafide research work under my supervision and is worthy of consideration for the award of the degree of Doctor of Philosophy of the Institute.

Prof. Hemangee K. Kapoor
Department of CSE,
Indian Institute of Technology Guwahati,
Guwahati, Assam, INDIA 781039,
hemangee@iitg.ernet.in

Date: February 28, 2018
Place: IIT Guwahati

Dedicated to

Late Shyamapada Adhikary (my maternal grandfather)

and

Late Nandan Adhikary & Late Gautam Adhikary (both of my maternal uncles)

and

my loving parents & all of my teachers.

Acknowledgements

It is great pleasure for me to thank all the people whose instinct supports and encouraging attitudes always boost me up to engineer my initial research career. First and foremost, I thank to my supervisor Prof. Hemangee K. Kapoor for her guidance, patience and encouragement over the last five and half years. Her unwavering enthusiasm for Computer Architecture kept me constantly engaged with my research work. Needless to say, the journey of PhD is a small demonstration of our life, which has uncountable number of ups and downs. The strong association of five and half years with her taught me about handling different situations with a gentle smile. The plethora of wonderful discussions with her helped me a lot to understand how to visualise something from a philosopher's point of view.

I take the opportunity to thank my Doctoral Committee members: Prof. Diganta Goswami, Prof. Purandar Bhaduri and Dr. Arnab Sarkar for their fruitful and constructive suggestions, which help me enough to shape up my final thesis. Sincerely, I would also like to thank Prof. Kalpesh Kapoor and Dr. Arnab Sarkar for numerous academic/non-academic suggestions. Discussions with them are always fun irrespective of the topics. Furthermore, my sincere thank to Prof. S. V. Rao, the Head of the Department of Computer Science and Engineering and other faculty members for their constant supports and helps.

I sincerely thank to Mr. Souvik Chowdhury, Mr. Raktajit Pathak, Mr. Nanu Alan Kachari, Mr. Bhriguraj Borah and all other institute's staffs who helped me at different times during my stay at IIT Guwahati. I must mention and thank to the Student Affairs section for providing on-campus hostel facility. I am conveying my appreciation to all the hostel and canteen staffs, security guards, housekeeping staffs who made my life smooth at the campus.

Life is incomplete without a set of good friends. I am fortunate enough to have such a good set of friends during PhD who constantly push me to enhance my creative thinking. I really feel privileged to have the company of Mandar, Suman, Abhradeep, Mitali, Arindam, Pramit, Krishnanjan, Srimoy, Niladri, Mriganka, Jinat, Gourhari, Gundappa, Atanu, Bihan, Susmita, Anwesika, Dr. Shuvendu, Dr. Sibaji, Dr. Dishari, Karnika, Arya, Ramyani, Soheni, Arunangshu, Dr. Himadri, Omkar, Deepanjan, Debojit, Abhinandan, Arunabha, Soumi and Sayan. Uncountable number of discussions especially reagrding different forms of performing arts along with our own creations really added some special flavours to my PhD life,

which are inexpressible. I am also thankful to my Tabla guru Mr. Dhriti Gobinda Dutta for his unforgettable inspirational lectures. During my PhD, I have also spent some joyous moments with my fellow research scholars like Biswajit, Dr. Debanjan, Dr. Pradeep, Mrityunjay, Rahul, Basant, Piyoosh, Rajesh, Monojit, Shilpa, Sandeep, Awnish, Durgesh, Akash, Ranajit, Amit, Shubhrendu, Dr. Suddhashil, Dr. Suchetana, Dr. Ashok, Dr. Niladri, Dr. Mayank, Satish, Dr. Nilkanta, Subrata, Anasua, Aparajita, Abhijit, Shrestha, Ujjwal, Hema, Sumita, Sanjukta and many more. I must mention the name of Dr. Sandip Chakraborty, a iconic character in my life, whose constant support during initial stages of my PhD is unforgettable. I am also thankful to Mr. Lalatendu Behera for adding some special colours to my PhD life.

During my PhD, I got the opportunity to work with Dr. Shirshendu Das, Dipika Deb, Sukarn Agarwal, Palash Das, Sheel Sindhu Manohar, Arijit Nath, Khushboo Rani, T. Susma Devi, Major Alankar Umdekar, Prateek Halwe, Gibran Iqbal and Arpit Agarwal. Sharing of knowledge with them, during countless number of technical/non-technical discussions were amazing and simply helped me to carry out my research work. I would also like to convey my respects and special thanks to Mrs. Manjari Saha, Mr. Avijit Bose, Prof. Ranjani Parthasarathi, Mr. Arnab Biswas and Mr. Biswajit Sanyal. I am also thankful to all the anonimous reviewers of my papers & thesis and my friends inside & outside the IIT Guwahati.

Finally, I would like to share a few words about my family. Their constant support along with strong motivations always kept me focused to my research work. I am extremely thankful to all of my beloved family members who have channelised enough mental strength and encouragement during all ups and downs of my PhD life. I cannot express enough thanks to my parents, the two most important persons in my life, who show me the correct path with the onset of my life. Instinctively they encouraged me in each step of my life with due respect to my every thoughts and decisions. Lastly, towards completion, I should express my thanks to all of those people who demoralised/discouraged/refused me to help at different moments/stages of my life, as it indoctrinated me about taking something as a challenge with sound cultivation of my own thoughts.

Abstract

Ever increasing demand of processing speed and parallelism, along with the modern shrunk transistors, motivates the architects to increase the number of cores on a single chip leading to Chip Multi-Processors (CMPs). To commensurate the data demand of these high number of cores, large on-chip Last Level Caches (LLCs) are integrated. After studying a plethora of prior works, it has been concluded that, LLCs play a vital role in maintaining system performance by accumulating more data on-chip. But large sized LLCs are accounted for their significant leakage energy consumption, which has a circular dependency on the effective temperature of the chip-circuitry. In addition to curtailing the circuit's reliability, this increased chip temperature (caused due to heavy power consumption) has enough potential to damage the on-chip circuitry permanently, and to exacerbate the battery life in embedded systems.

Towards leakage minimised on-chip cache design, with due consideration to the Locality of Reference, we propose a set of performance constrained Dynamic Cache Resizing techniques to reduce leakage in LLCs. The resizing is done by turning off/on some cache banks which can be implemented by power gating at circuit level. The cache resizing decision is triggered based upon the dynamic (cache) usage and change in system performance. We get 65% savings in leakage energy consumption. In the next contribution, apart from bank level granularity, we proposed cache resizing at way-level granularity, where performance degradation is handled by incorporating DAM (Dynamic Associativity Management) techniques. In this policy, we obtain 70% improvement in the leakage energy. Both of these techniques outperform a prior state preserving leakage reduction technique called, Drowsy cache; both in terms of leakage savings and EDP gains. All of these techniques are evaluated for tiled CMP having a multi-banked shared L2 cache as its on-chip LLC.

From the thermal efficiency perspective, it has been noticed that, larger caches fabricated in smaller technology nodes are the potential candidates for generating hotspots similar to the CPU cores. The increased power density at heavily used cache zones can heated them up faster, whereas the lightly used cache portions unnecessarily consume high leakage, which has a circular dependency with chip temperature. To mitigate these issues, our proposal selectively turns off LLC banks to reduce temperature by reducing their power consumption. Moreover, these turned off cache portions are utilised as on-chip thermal buffers that reduce

effective chip temperature. The proposal is evaluated for tiled CMP architecture as well as CMPs having large multi-banked centralised LLCs. These approaches reduce average chip temperature by around 4°C and 6°C, respectively, for both architectures. The results are also compared with DVFS methods.

The thesis has thus demonstrated that, large LLCs on-chip need resizing for optimal power management and they can also assist in controlling chip temperature.

Contents

List of Figures

List of Tables

Abbreviations

CMP	Chip Multi-Processor
LLC	Last Level Cache
CPI	Cycles Per Instruction
LRU	Least Recently Used
MRU	Most Recently Used
AMAT	Average Memory Access Time
MPKI	Miss Per Kilo (Thousand) Instructions
IPC	Instructions Per Cycle
NUCA	Non Uniform Cache Access
NoC	Network on Chip
SNUCA	Static NUCA
DNUCA	Dynamic NUCA
TCMP	Tiled based CMP
CCMP	CMP with Centralised on-chip LLC
DAM	Dynamic Associativity Management
CMP-VR	A DAM based architecture (Victim Retention)
CMP-SVR	A DAM based architecture (Set Associative Victim Retention)
RT	Reserve sTorage
NT	Normal sTorage
EDP	Energy Delay Product (Energy $\times$ CPI)
TDP	Thermal Design Power
DVFS	Dynamic Voltage & Frequency Scaling
DTM	Dynamic Thermal Management
IPS	Instructions Per Second

Chapter 1

Introduction

A few years back, in the city of *Dieland* a job advertisement was published.

Vacancy!! Vacancy!! Vacancy!!

A set of identically efficient employees are needed for information management at **Dieland Corporate House (DCH)**. The selected applicants have to maintain a set of information during office hours. They have to provide information to the foremen whenever needed, and have to manage the information carefully if foremen have modified it. The missing information, if any, has to be brought from the subordinates. Workload may vary with the situations.

Reward-UFS, the *Uninterrupted Food Supply* during office hours.

Almost a couple of weeks later, on a fine morning a set of identically efficient and jobless people came for the interview at the DCH-office. Surprised after seeing such a crowd, the manager arranged an interview and selected a chunk of people, who were found fit for the job. These newly appointed employees formed a group,

named *Basket*. The governing body of DCH was very pleased with the situation and *Basket* started enjoying UFS facility during the office hours.

The first quarter was over. Everything was fine at the *Basket* end. However, suddenly one morning, the manager noticed something. Most members of *Basket* were busy with the appetiser whereas very few of them working hard with a mammoth amount of workloads; and another few having a bit lesser amount of work to handle. Hence, there was a mass of employees relaxing and energised heavily just by enjoying the UFS and creating disturbances at office. The manager was worried about the horrified situation and tried hard to get a solution. Alas! no idea clicked in his wobbly mind even after thinking continuously round the clock for over a week.

At the beginning of the next week, while buying fuel for his car, an idea came to manager's mind. "UFS!! the principal fuel of *Basket*...should be restricted right now!!...no food supply to the idle ones." On his way to office, the rules were prepared and thrown in the *Basket*. *Basket* opened the rule-book and the rules revealed:

1. (R1) More food consumption is bad for one's health and may reduce your lifetime. Therefore, food will be supplied only when you will work and be honest in helping others.

2. (R2) Rearrange/Optimise the movements of your organs in such a way that you can provide the same performance with lesser amount of food as only some of your body-parts will be active.

3. (R3) Those who handle heavy workloads are allowed to sleep for a stipulated time-span (hence no food supply); UFS is however restricted to the people having lesser or no workloads to handle.

4. (R4) Recently, we have noticed that, some of you have become over energised due to UFS and are disturbing the foremen adjacent to you. Hence, there will be restricted food supply to the neighbours of the foremen.

To get rid of these rules, a meeting was arranged where the whole *Basket* decided to resign from DCH. Meanwhile, the whole corporate world of *Dieland* started following the same set of rules, which *Basket* was not accepting. Finally, the news spreaded all over and soon, the *Basket* returned back at DCH following the set of rules.

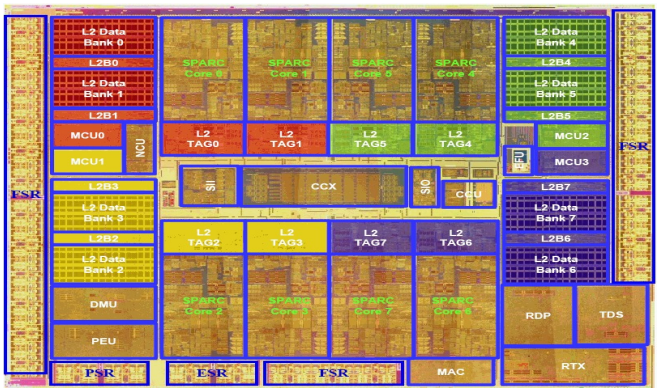
When I came to know about this tale of *DCH*, from a computer architect's point of view, I was surprised that, the characters can be mapped and fixed well in my domain. Making the foremen as CPU cores and Basket as caches with manager as the cache controller, the scenario can be visualised well enough in the world of real computer hardware.

Basically, over the past decade (with end of Dennard Scaling [8]), the single-core systems touched the upper-limit of maximum attainable performance, i.e. the highest operating frequency, primarily due to its higher power consumption. The architects and designers got motivations from this to develop processor chips with multiple cores that can operate at lower voltages and frequencies than their single core counterparts, while offering the same performance with lesser power consumption. Furthermore, the ever increasing demand of processing power to run modern heavy applications needs heavy amount of parallel processing, which can no longer be satisfied with single core systems. With due consideration to Moore's Law for on-chip transistor counts, and depending upon the modern integration technology, the number of cores in a chip is expected to increase every year [9, 10]. This integration of multiple cores on a single chip is known as Chip Multi-Processors (CMPs).

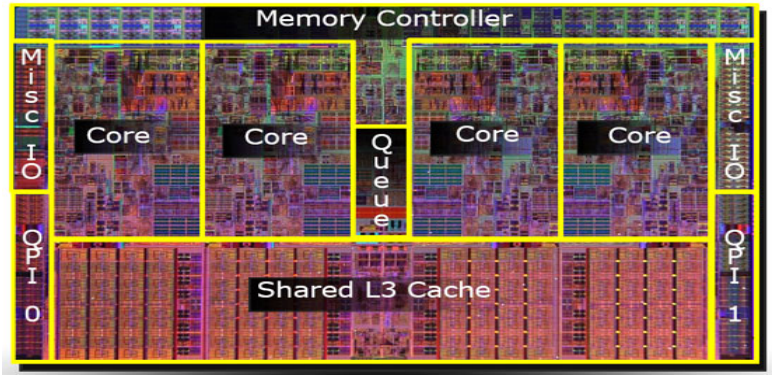
1.1 Modern Chip Multi-Processors (CMPs)

Modern CMPs are usually equipped with a large number of cores, each with same (homogeneous) or different (heterogeneous) processing power. To commensurate the high data demand of these cores, multi-level on-chip caches (i.e. the *Basket* of DCH) are attached [11]. Eventually, the large CMPs need strong communications

among all of its components, which can be offered by a modern on-chip network system, called as Network on-Chip (NoC). Figure 1.1 shows floorplan outlines and designs for two modern CMPs- UltraSPARC T2 and Nehalem [12].



(a) UltraSPARC T2 [13]



(b) Nehalem [10]

FIGURE 1.1: Modern CMPs: Design and Floorplan outlines

1.1.1 Components in CMPs

On-chip components of a modern CMP can be classified into three categories:

1. **CPU Cores**. as the computational units.
2. **On-chip Caches**. to accumulate most recently used data on the chip.
3. **Network-on-Chip**. for communicating among the on-chip components.

Computational performance refers to how much time the system takes to execute an application. It is usually measured in terms of Total Execution Time, Instructions Per Cycle (IPC), Instructions Per Second (IPS), speed-up factor to a base system, etc [14]. The resulting performance in a CMP further depends on how much parallelisation is achieved while the application is executed. The parallelisation is achieved by multi-threading where an application is grained into a certain number of threads.

Depending upon the nature of majority of its instructions, an application is tagged either as Compute Bound or Memory Bound [15, 16, 17, 6]. In the case of memory bound application, most of its instructions need to access memory. Hence, system performance has direct dependency upon both core's clock frequency and memory latency. Memory latency further depends upon the following: (I) How quickly memory finds the desired block after it has been requested, (II) How fast the data reaches to the core from the memory subsystem. Former one depends upon the type and structure of the memory, whereas latter one relies on NoC performance. The equation for Execution Time can be formulated as:

$$Total\_Exec\_Time = Computation\_Time + Memory\_Cycles + NoC\_Latency \tag{1.1}$$

1.1.1.1 On-Chip Caches

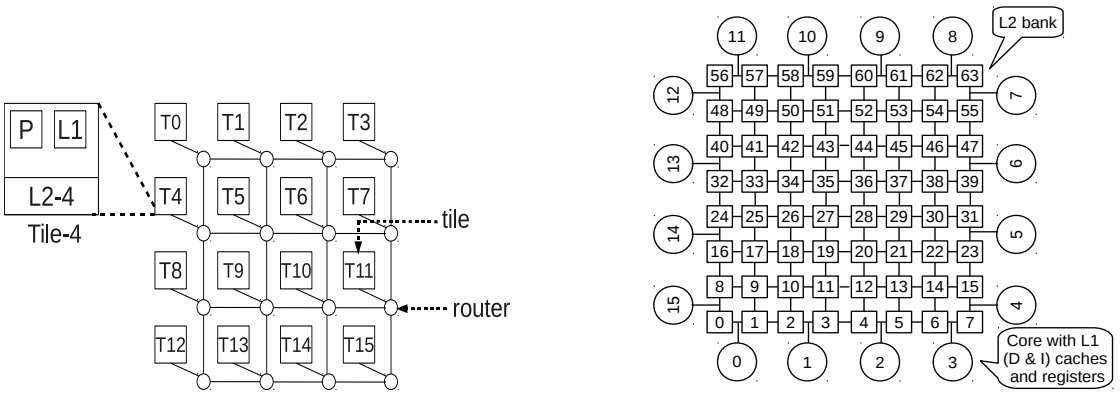
The rapid progress in VLSI technology, and reduced channel length of modern transistors [9] help architects and chip designers to integrate more on-chip cores with the larger multi-level on-chip caches. Caches are labeled according to their distance from the CPU cores. The nearest (i.e. higher) level caches to the cores are named as L1 caches, and next (i.e. comparatively lower) level is named as L2 and so on [14]. However, among all levels of on-chip caches Last Level Caches (LLCs) are the largest in size and occupy a significant area on the wafer real-estate (ref. Figure 1.1). Recent CMPs usually have two to three levels of on-chip caches,

out of which, mostly the LLC is kept as shared cache space, whereas rest are made private to the cores [12].

Accessing a large cache for every memory operation is both time consuming and incurs more power. Additionally, the large on-chip caches are accounted for their significant contributions in total on-chip power consumption. To mitigate this issue, architects split the whole large cache physically into a number of identical slices, each of them is called as cache bank. When the large cache is divided into banks, the division may follow either of the two patterns-(a) set wise distribution, where all ways of a set are present in a same bank, (b) way wise distribution, where a particular way of all the sets are kept in the same bank. Some optimised structures also follow a mixed distribution of the above two. The multi-banked caches not only reduce the access latency but also provide more space to the circuit designers for power optimisation.

Data in the caches are stored as cache blocks. For a CMP, these cache blocks/locations may maintain a uniform (UCA) or non-uniform (NUCA) cache access time from the cores. Most of the state-of-the-art CMPs follow NUCA architectures, as maintaining uniform latency with larger caches when coupled with a large set of cores is impractical. In the case of NUCA, cache blocks can be accessed with lesser latency if the blocks are located near to the requesting core, and reverse situation may also arise, which drastically curtails system performance. Static NUCA (SNUCA) always maintains a fixed cache location for a particular block, whereas Dynamic NUCA (DNUCA) moves the block to the close proximity of the requester core [18]. However, too much block movements across the larger cache increase power consumption in DNUCA. The proposed designs given in this thesis use the multi-banked SNUCA LLC structure with set wise distribution.

While fetching cache blocks from the lower level memory to multi-level caches, data allocation can follow either multi-level inclusion or multi-level exclusion property. In the former one, cache blocks present at the higher levels are the proper subset of the blocks present at the lower level caches. Whereas in the latter one, blocks



(a) Tiled CMP architecture (b) CMP with centralised LLC

FIGURE 1.2: A Tiled CMP and a CMP with Centralised LLC.

present across the cache levels are disjoint. In this thesis, we use multi-level inclusive caches.

1.1.1.2 Placement of LLCs

Tiled CMP (TCMP) A Tiled CMP architecture divides the chip into multiple number of identical tiles [19, 18]. Figure 1.2(a) depicts a model of a Tiled CMP, where the chip is constructed by 16 identical tiles. Each of them contains a CPU core with its private L1 Data & Instruction caches, and an L2 cache bank. A 2D mesh NoC [20] connects all the 16 tiles to form the whole chip. Each of the tiles is also associated with its own router for communicating through the NoC with other tiles. The L2 cache is here physically distributed but all banks are shared among the 16 cores. Accessing a data block from the home L2 bank (i.e. the bank located with the same tile of the requester core) incurs less latency than the far remote banks (located in some other tiles). L2 is considered here as on-chip LLC. The number written in each tile (like T4) represents the tile id. The tiles are numbered from 0 to 15.

CMP with Centralised LLC (CCMP) In case of CCMP, the whole chip is also divided into several tiles, where the tiles are not identical. Each tile either contains an LLC bank or can contain a core along with its private L1 Data &

Instruction caches. The LLCs are large in size and mostly the number of banks are more and placed centrally. Figure 1.2(b) illustrates the architecture, where 16 cores are available and tiles containing cores are located along the periphery of the CMP. Note that, in this figure, L2 is considered as the on-chip LLC similar to TCMP architecture.

1.2 Power Consumption in CMPs

For each of the on-chip components, the total power consumption can be divided into three major parts [4, 21, 22]-(A) Dynamic Power, (B) Static Power and (C) Short Circuit Power.

1.2.1 Dynamic Power

Dynamic Power is consumed due to switching activity of the transistors present on-chip due to charging/discharging of the output capacitances. However, for computational elements dynamic power is nothing but the processing power which can be written as follows [23, 22]-

$$P_{Dyn} = \alpha.C.V^2.f \tag{1.2}$$

P_{Dyn} represents the dynamic power of the cores, and α, C, V and f denote activity factor, capacitance, supply voltage and running frequency of the core, respectively, which indicates that, core's power consumption has direct dependency on its operating frequency.

For the on-chip caches, dynamic power is consumed during cache accesses. The cache is accessed either for a read operation or for a write operation. Usually, in the case of cache memory (constructed with SRAM cells), reading and writing power consumption are almost the same [7, 21, 24]. The detailed power/energy model of the cache is discussed in the next chapter.

The NoC of any CMP is constructed with two basic components-(a) Routers and (b) Connecting Links. The routers are complicated part of NoC which use several routing algorithms for sending data optimally across the chip. The data is transmitted through the Links, which are primarily a set of metal wires. Routers consume dynamic power during routing operations which usually comes from three basic sub-parts of routers [25]-

- Router Clock- The essential component of any synchronous design.
- FIFO Buffers- That maintain the order/sequence of incoming/outgoing data blocks.
- Allocators and Arbiters- That channelise data blocks to reach the proper destinations.

1.2.2 Static Power

On-chip circuitry draws some amount of power even when the circuit is not performing any job, so this power is termed as static power. The static power mainly indicates the circuit's leakage power which is due to two important leakage components-(a) Subthreshold Leakage and (b) Gate Leakage [23, 22, 26, 27]. The former one has direct dependency with the supply voltage and running chip temperature. High chip temperature breaks the co-valent bonds in the atoms of semiconductor materials and releases electrons which start flowing through the reverse bias, and draws a current, called Subthreshold Leakage current. The power consumed due to this Subthreshold Leakage current is known as Subthreshold Leakage power. On the other hand, down-scaling device size reduces the thickness of gate-oxide materials, which increases the Gate Leakage power. The detailed analysis of the root causes of leakage power is out of scope of this thesis.

Following equation shows the direct dependency of subthreshold leakage upon the running temperature and supply voltage [28, 23]:

$$P_s = K_1 V_{DD} T^2 e^{(\alpha V_{DD} + \beta)/T))} + K_2 e^{(\gamma V_{DD} + \delta)} \tag{1.3}$$

P_s denotes the static power consumption due to subthreshold leakage for a CMOS circuit. V_{DD} is the supply voltage and T implies the current temperature. K_1, K_2, α, β, γ and δ are the empirical constants which represent different circuit parameters.

1.2.3 Short-Circuit Power

This power is consumed due to non-zero rise/fall time of the CMOS circuitry. A very short time-span when both NMOS and PMOS are active simultaneously, makes a direct link between supply voltage and ground. This power consumption is very negligible and often ignored while analysing power consumption of modern CMPs [23, 22, 26].

In the case of modern CMPs, among all the on-chip components, cores are usually accounted for their high dynamic power consumption whereas caches are considered for their significant leakage. The heavy dynamic power of the cores are reduced by applying some classical techniques like- DVFS, DPM etc. In case of cache leakage, either state destroying or state preserving techniques are used, and in a few cases they are clubbed together. In some of the recent CMPs it has also been found that, the large on-chip LLCs increase the number of transistors with large area occupancy. This large chunk of LLC consumes heavy leakage and plays the role of a significant contributor to the total on-chip power consumption. For some recent CMPs, Table 1.1 shows the percentage contributions of on-chip caches to the total power consumption of the chip, which can motivate the researchers to reduce these numbers. This thesis focuses upon the leakage power consumption of the on-chip LLCs, which has become a limiting factor in designing modern CMPs with shrunk transistors.

Microprocessor	**Power Consumed by on-chip Caches with respect to total power**
ARM 920T	44%
Strong ARM SA-110	27%
21164 DEC Alpha	25-30%
Niagra	12%
Niagra2	21%
Alpha 21364	13%
Xeon (Tulsa)	13%

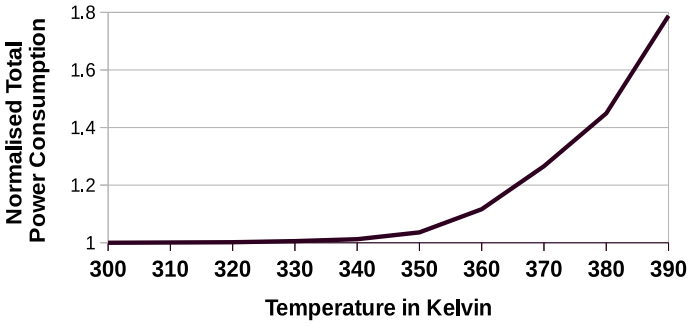
TABLE 1.1: Power consumed by on-chip caches [4, 5]

1.3 Thermal Issues in CMPs

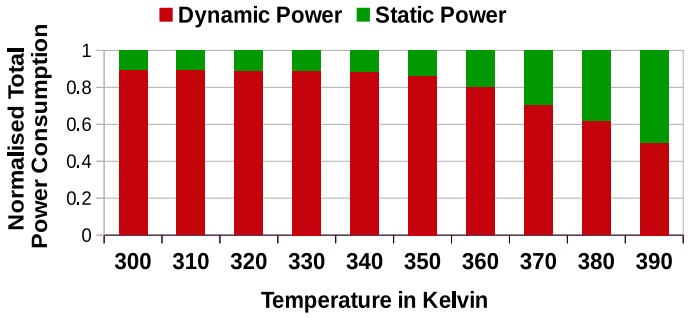
Rapid advancements in VLSI technology reduce the channel length of modern transistors, for which modern CMPs are well equipped with giant number of on-chip components to enhance system performance. Increment in number of on-chip components increases the on-chip power density which in turn dissipates more heat and increases chip temperature. The raised up chip temperature quadratically increases the leakage power (ref. Equation 1.3). The increased leakage power further increases the effective chip temperature, and thus forms a circular dependency with temperature [28, 23]. However, on-chip circuitry in high temperature not only starts malfunctioning, but can also be damaged permanently. Figure 1.3 shows the change in total power consumption with the increment in chip temperature, and this increment is due to the increase in leakage power. At high temperature, leakage consumption is even as same as dynamic power. The values in the figures are collected by running the same workload with Niagara2 CMP in different temperatures in McPAT simulator [21]. The temperature values are given in Kelvin in both the figures.

1.3.1 Thermal Characteristics

From the above discussion, it is pretty clear that power consumption constructs the foundation of on-chip thermal issues. Hence, as a mitigation, most of the in-built thermal management techniques use power consumption as an optimisation



(a) Increment in Total Power consumption



(b) Static vs. Dynamic power

FIGURE 1.3: Power consumed by Niagara2 Chip at different temperatures.

knob for thermal efficient chip design. Although the power consumption can be changed instantaneously even in practical cases, this is not true about temperature. The associated thermal capacitances of every chip element does not allow instantaneous change in temperature of the chip components. During execution the chip temperature gradually reaches its *steady-state*. The intermediate values before reaching at steady-state are known as *transient temperatures*. Figure 1.4 shows how the temperature changes over time during execution. The temperature values shown along Y-axis are in °C and X-axis represents the time stamps during execution. The values are obtained using HotSpot simulator [3].

1.3.2 Mitigations-at a glance

The primary objective of any thermal efficient chip design is to maintain the chip temperature under a certain value, the thermal threshold, commonly known as the critical temperature [29]. Classical chip manufacturing follows a common

industry practice to provide Thermal Design Power (TDP) which is defined as the highest sustainable power consumption of the chip. Even the manufacturers further recommend that the cooling solutions should dissipate TDP in order to ensure that the cooling arrangements are not over-dimensioned. Normally, TDP is not the highest attainable power of the chip, rather, the chips are designed with some Dynamic Thermal Management (DTM) methods. Commonly used DTMs are reactive in nature and they gate the cores' clock, reduce or scale the supply voltage/frequency, migrate tasks from hotter region to colder, regulate fan-speed etc. Few DTMs proposed recently also suggest use of larger on-chip caches in thermal efficient chip design. As caches are usually colder than the hottest core region, and dynamic resizing of large cache does not affect performance abruptly, hence, with due consideration to performance, some cache portions can be turned-off to create thermal buffers for reducing the chip temperature. These thermal buffers will be eventually cooled down and will reduce their own as well as peers' temperature values. Although, there is a plethora of proposed solutions for DTM [30, 29], yet, thermal constraints are still the biggest limiting factor for modern chip design, especially in the current era of *short channel length*.

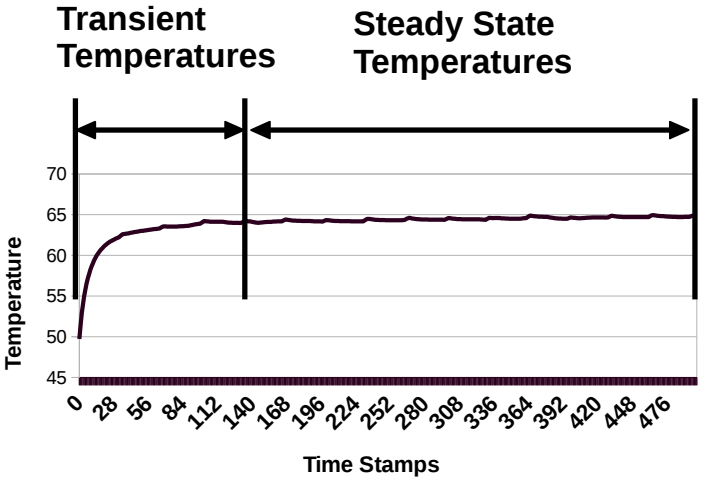


FIGURE 1.4: Change in Average Chip temperature during process execution.

1.4 Cache based Mitigations

1.4.1 Leakage Minimisation

According to Table 1.1 on-chip caches are one of the major contributors to the total power consumption of the chip. Moreover, cache leakage plays a pivotal role for increasing the power values. Most of the cache based power optimisation techniques hence focus upon leakage minimisation, which is usually done in two ways [4, 5]-

1. **State Destroying Method:** This method turns off a portion of cache to save power. The turn-off decision is either taken dynamically at run-time or statically at compile time. Power gating is used as a backbone of turning off circuitry. A portion of the cache can be turned off by shutting down cache banks in the case of multi-banked caches or shutting down some cache ways for higher associative caches. Blocks are usually evicted from the cache portion before turning it off and hence, the associated metadata with the cache controller is completely destroyed, as the name indicates.

2. **State Preserving Method:** This policy does not evict the cache blocks from the caches, rather it keeps both data and tags but reduces the supply voltage to a level which can preserve the state of the data. Maintaining a low voltage level for state preservation is commonly known as drowsy caches [31]. However, when these data are required to be accessed, the voltage level needs to be raised up. This voltage transition may incur some idle clock cycles, which can slightly increase the memory latency.

State destroying methods are very effective while saving power consumption aggressively. This policy can take advantages by anticipating cache access patterns by exploiting *Locality of Reference*, a classical cache access property. Turning off the least used cache portions for saving power is a better option, i.e. lesser performance degradation prone, while using state destroying method. However, for

state preserving ones, putting least used cache portions in low power mode can also help to maintain performance.

1.4.2 Controlling Temperature

Larger LLCs in modern CMPs have enough potential to create hotspots at the cache area [32]. These hotspots are generated due to Locality of Reference, as same or its nearby locations are accessed repeatedly over a short time-span. Hence, shutting down of these heavily accessed banks can help to reduce cache hotspots. But, turning-off a heavily used cache area can degrade system performance noticeably, therefore, an optimal cache resizing has to be attached to put a balance between temperature and performance. On the other end of the spectrum, leakage is consumed as long the cache is turned-on, irrespective of its accesses. To get rid of circular dependency of leakage and effective chip temperature, least used cache portions can also be turned-off, which will incur lesser performance degradation. However, the turned-off cache banks will create thermal buffers on-chip which will eventually cool down the chip.

1.4.3 Objectives of this work

The main objective of this thesis is to reduce LLC leakage and control chip temperature by resizing LLC dynamically. The proposed techniques are also compared with the existing ones in order to demonstrate the improvements of our architecture. To achieve the goal, the first two proposals reduce leakage by incorporating state destroying technique in the LLCs, whereas the last two policies have been developed to control the chip temperature. The proposed techniques are completely implemented at LLC, and they are transparent to the higher level caches and cores. The motivation behind our proposals is discussed in the next section.

1.5 Motivations

From the performance and cache access perspective, caches are usually accessed in a non-uniform manner and follow the Locality of Reference. This in turn keeps some cache portions under-utilised, which only consume a significant amount of leakage power. Hence, shutting down these under-utilised cache portions can help us to reduce leakage while maintaining performance within permissible limits, thanks to Locality of Reference. However, these cache access patterns are only known during execution and are diverse in nature across the applications, therefore, no generalised cache resizing patterns can be proposed. Furthermore, for a set of modern long running applications, Locality of Reference is violated, which cannot anticipate the cache usages at the far future. This phenomena motivates us to develop a dynamic cache resizing technique, which resizes the cache in either direction. That is, provide adequate space to the application's working set as per its current requirement.

On the other hand, caches are usually considered as colder on-chip components, whereas cores are accounted for their high temperature. In a recent study, a thermal variance of 30 Kelvin (K) [32] has been noticed at the on-chip LLC, which strongly claims about the presence of cache hotspots. The heavily accessed cache area consumes more dynamic as well as leakage power, which can create the cache hotspots. Hence, shutting down of these heavily accessed cache portions can reduce the cache hotspots, but at the same time, they can aggravate the system performance, if cache resizing is not controllable. Conversely, shutting down least used cache portions reduces leakage and also curtails the chance for creation of future hotspots on-chip. However, powered off cache portions create a large chunk of thermal buffer which can gradually lower the chip temperature by exploiting the principle of Superposition and Reciprocity of Heat Transfer [33].

The main aims of this thesis-work can be pointed out in a nutshell as follows:

- Dynamic Cache Resizing (DiCeR) to save leakage power consumption at bank level granularity.

- Resize the cache at both bank and way level granularities with some DAM based technique to maintain performance while still reducing leakage.
- Reduce cache hotspots by independently turning off heavily used banks and turning-off least used ones to create thermal buffer with reasonable reduction in leakage.
- Create (dynamically resizable) optimal amount of on-chip thermal buffer by shutting down cache banks based on their location and proximity to cores in order to gradually reduce the chip temperature.

1.6 Principal Contributions

This thesis considers a modern CMP having 16 cores UltraSPARC III model with a TCMP and a CCMP architecture, as discussed earlier. The LLC in the whole thesis is implying on-chip L2 unless otherwise specified. The major contributions of this thesis can be summarised as follows-

1.6.1 DiCeR at Cache Bank Level

R1 at DCH : "More food is bad for health"

In our first leakage saving technique, we initially reduce cache-size by only shutting down cache banks till an allowable degradation in the performance. This technique is referred to as BSP. However, this policy cannot provide adequate cache space to the process in case it needs more cache space in future execution. Hence, we further propose a dynamic cache tuning technique which considers performance and locality of reference as system-wide constraints to manage the cache size in our baseline architecture, shown in Figure 1.2(a). In order to save leakage power, based on usage statistics, L2 cache banks are shutdown at runtime and its future accesses are remapped to other active L2 cache banks, called as target banks.

Additionally, this policy also takes care of the sudden increase in the application's WSS during execution by allowing dynamic restarting of the powered off cache banks. System performance is monitored periodically and accordingly L2 bank(s) will be restarted if performance degradation is more than a threshold value. During turning on process, all the remapped contents are brought back to its home bank from its remapped location. The results are compared with BSP and Drowsy Cache [1], an existing policy. In particular, the following variations are proposed : **(1) B_ON_OFF_ALL.** The system can decide to shutdown or turn-on some cache banks periodically throughout the process execution. **(2) B_OFF_ONCE.** The system allows to shutdown banks initially and once the bank restarting initiates, no more shutdown is permitted further. **(3) B_ON_OFF_OPT.** This policy resizes cache like first policy, with some predefined time slices in which cache cannot be resized.

Out of these three policies, both first and third ones save 65% leakage energy on an average which is higher than the second one, where leakage saving is around 46%. But, the unrestricted cache resizing in **B_ON_OFF_ALL** shows lesser EDP gain than **B_ON_OFF_OPT**. The average EDP gains for these three policies are 29%, 27% and 30%, respectively.

1.6.2 DiCeR in combination with DAM technique

R2 at DCH : "Better organ movements with restricted food supply"

In BSP, the bank shutdown process saves static power but reduces the performance of LLC. Due to multiple banks being shutdown the target banks may also get overloaded. Additionally, the request forwarding increases the on-chip traffic. Therefore, to improve the performance of the target banks we use a dynamic associativity management (DAM) technique called CMP-SVR [34]. Furthermore, the

cost of request forwarding is optimized by considering network distance as an additional metric for target selection. These two strategies help to reduce performance degradation.

While using DAM method on BSP, we can further try to reduce leakage energy by turning-off cache ways and apply associativity management. This policy goes through the following phases :

- **Bank shutdown:** Least used cache banks are turned off until either the performance degrades beyond the allowable threshold or if the number of banks turned off reaches a predefined maximum limit.
- **Way shutdown:** It is desirable to turn off some number of ways from every set, when a bank is not suitable for complete power off. DAM is incorporated here to enhance performance after way shutdown.

This policy saves 70% of leakage energy with negligible degradation in performance and outperforms some prior works. The average EDP gains for this policy is around 35%.

1.6.3 DiCeR for temperature control in TCMP

R3 at DCH : "Recess for the exhausted and no food to the idle"

Most DTM techniques apply DVFS or task migration to reduce core temperature, as cores are considered as the hottest on-chip components. On the other hand, modern large on-chip Last Level Caches (LLCs) are significant contributors to on-chip leakage power consumption and occupy the largest on-chip area. As power reduction plays the pivotal role for temperature reduction, hence, DiCeR not only can reduce leakage power consumption, furthermore, it can create on-chip thermal buffers for reducing average as well as peak temperature of the chip without disturbing the computation. In order to use cache to control temperature, the cache

resizing decisions in a TCMP (ref. Figure 1.2(a)) will be taken based upon the generated cache hotspots and/or the access patterns, during the process execution.

The major contributions of this work can be summarised as follows:

1. **Hot Bank [HB]** As leakage power increases quadratically with the temperature, it is hence better to turn off a heavily accessed "hot" bank and distribute its blocks to a colder target bank to reduce the cache hotspot as well as the leakage.

2. **Cold Bank [CB]** As leakage increases quadratically with the temperature and forms a circular dependency on the temperature, so, it is beneficial to shutdown least accessed and comparatively colder banks to make their power consumption zero. The small number of their remapped requests will not drastically affect the targets.

3. Both of these policies are compared with a greedy DVFS technique [2] which employs per core DVFS upon threshold temperature violation.

Through this thermal efficient cache resizing, we are able to reduce average chip temperature by 4°C, at most. The maximum leakage saving achieved through this policy is more than 40% with slight degradation in performance. This leakage savings with slight change in performance offer an average EDP gains of 21% and 29% for HB and CB, respectively.

1.6.4 DiCeR for temperature control in CCMP

R4 at DCH : "Don't disturb your peers"

This contribution analyses the role of a centralised multi-banked SNUCA LLC in thermal management while maintaining system performance. We dynamically

resize LLC to optimally balance the performance and chip temperature by offering two levels of thermal management-(i) controlling cache temperature, and (ii) reducing temperature of the global hotspots by governing conductive heat transfer.

The major contributions of this work can be summarised as follows:

1. Considering performance as a system-wide constraint, we have developed an analytical model by employing Lagrange Multiplier [35] for our architecture to determine the optimal cache size.
2. The analytically determined optimal cache size is used for resizing LLC by following the three thermal efficient patterns.
    - **AltRow.** Shuts down alternate rows of cache banks.
    - **Chess.** Generates a chessboard like pattern in LLC resizing.
    - **OptTar.** Cache banks closer to cores are assigned highest shutdown priority, with optimal management of future requests of the turned off cache banks.

Among all of these patterns, OptTar shows the maximum average temperature which is around 6°C, with 40.3% savings in leakage energy, whereas AltRow and Chess save 26% and 26.5% leakage energy, respectively. The respective EDP gains for AltRow, Chess and OptTar are 11%, 11.5% and 18.7%.

1.7 Summary

To commensurate the high data demand in recent CMPs, equipped with a number of cores, large on-chip LLCs are attached. These LLCs play a vital role in maintaining system performance. But large sized LLCs are accounted for their significant leakage energy consumption, which has a circular dependency on effective temperature of the chip. In addition with curtailing circuit's reliability, this increased chip temperature has enough potential to permanently damage the on-chip circuitry.

In our work, we initially propose a set of performance constrained DiCeR techniques to reduce leakage in LLCs. The resizing is done by turning-off/on some cache banks which can be implemented by power gating at circuit level. The cache resizing decision is triggered based upon the dynamic change in system performance. We get 65% savings in leakage energy consumption. Apart from bank level granularity, we have also proposed cache resizing at way-level, where performance degradation is handled by incorporating DAM (Dynamic Associativity Management). In this policy, we have the 70% improvement in leakage energy. However, from thermal efficiency perspective, these turned-off cache portions are utilised as on-chip thermal buffers to reduce effective chip temperature, especially in CMPs having larger LLCs. The gained energy and thermal benefits are studied for (i) TCMP and (ii) CCMP architectures. For TCMP, we get a reduction of around 4°C for average chip temperature, which is closer to 6°C in CCMP.

1.8 Organisation of Thesis

The rest of this thesis is organised as follows:

- Chapter 2 summarises the background and prior works related to the contributions of the thesis.
- Chapter 3 summarises the simulation framework used to build up the contributions of this thesis.
- Chapter 4 presents the first contribution, which dynamically resizes the LLC to reduce static energy consumption.
- Chapter 5 incorporates a DAM technique to improve cache performance which has been traded to save more power by shutting down more LLC banks and cache ways inside the banks.
- Chapter 6 explores the role of LLCs in Tiled CMPs for minimisation of chip temperature.

- Chapter 7 provides the analytical and simulation frameworks for selecting optimal cache size to reduce chip temperature under a certain performance constraint.
- Chapter 8 finally concludes the thesis.
- Appendix A describes our closed loop simulation framework, used for our dynamic thermal simulations.

Chapter 2

Background

As mentioned in Chapter 1, the LLCs in modern CMPs are usually shared and the largest in size. Apart from its remarkable leakage consumption, the performance of a CMP largely depends upon the performance of its LLC [11, 36]. The main aim of this thesis is to design cache based energy/thermal efficient techniques by considering performance as a system wide constraint. Hence, before discussing about state-of-the-art energy/thermal efficient techniques developed over the decades, we initially discuss some preliminary concepts/results regarding shared LLCs, relevant to our work.

2.1 Access Patterns of Shared LLCs

Locality of Reference in addition with the existing diversities in the number of memory operations across the applications generate non-uniform accesses to the caches, and LLCs are not an exception. These non-uniform cache access patterns are generated by the applications at different levels of granularity during the process execution.

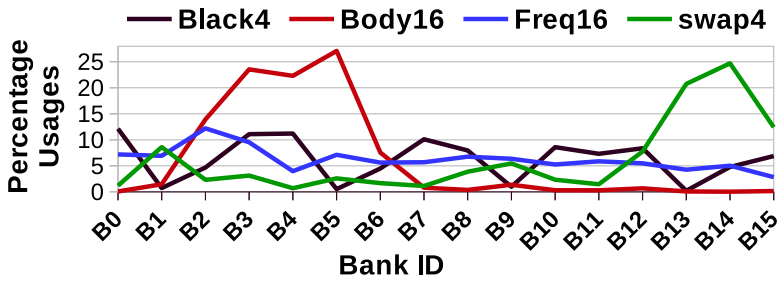
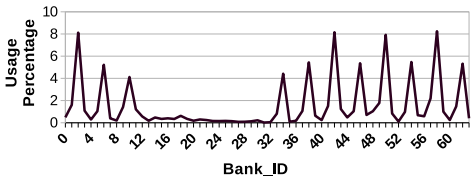


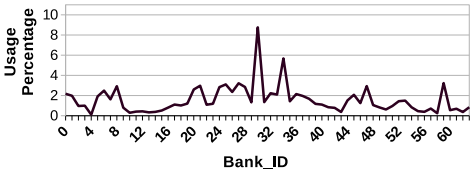
FIGURE 2.1: Variable bank usages across different benchmarks in a TCMP as shown in Figure 1.2(a).

2.1.1 Bank Level Granularity

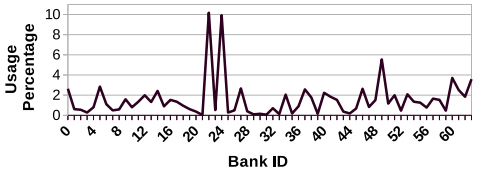
The runtime cache accesses across the banks are unevenly distributed in a multi-banked SNUCA based cache architecture. For four PARSEC [6] applications, we derived the bank accesses shown in Figure 2.1 for a TCMP architecture having 16 L2 banks. We have also plotted the same for three PARSEC applications in Figure 2.2 for a CCMP having 64 L2 banks. These figures show noticeable diversities in cache accesses across the applications at a certain time-stamp during execution.



(a) black16



(b) body16



(c) fluid16

FIGURE 2.2: Non uniform distribution of cache bank accesses in a CCMP like Figure 1.2(b).

According to Figures 2.1 and 2.2, each application has different cache space requirement and the cache banks that receive frequent data accesses are also not fixed across the applications. Therefore, cache bank usages can only be known at run-time as to which bank(s) is(are) the mostly accessed while which are utilised

the least. As all the cache banks are neither fully nor evenly utilised, it would be helpful to shutdown selective cache banks, specifically, the least used ones to save cache power consumption. Note that in the above discussion, the bank usages are calculated by using the formula:

$$L2BankUsage(i,j) = \frac{totalAccesses(i,j)}{totalL1Misses(j)} \times 100\% \tag{2.1}$$

where, i implies L2 bank-id and j is the remap-period. So, the term $L2BankUsage(i,j)$ represents bank usage percentage for bank-id i in jth interval.

The non-uniform cache access patterns discussed above are based on SNUCA architecture, where mapping of a particular cache block is always fixed to a particular bank. The accessing and allocation strategies are changed in the case of a DNUCA architecture, where a cache block is moved dynamically towards the core requesting it repeatedly. Tiled DNUCA [18] with its different variations are proposed recently to mitigate the performance bottleneck of SNUCA based LLCs. Although, DNUCA helps in performance enhancement, but, their implementation is critical in real hardware. Additionally, significant data movement across the cache banks can increase the dynamic power consumption of the on-chip circuitry. However, detailed discussions on this topic remain out of scope of this thesis.

2.1.2 Set Level Granularity

Similar to the access patterns across cache banks, it has been observed that the accesses within a cache bank are also unevenly distributed among the sets [34, 37]. This uneven distribution increases conflict misses in the heavily accessed sets. Figure 2.3 shows an example of how the conflict misses of each set varies in a bank in the case of our TCMP architecture. As seen in the figure, some sets are used heavily while some other sets remain underused. Such non-uniform load distribution reduces the total utilisation of the cache by increasing conflict and capacity misses at the higher utilised cache sets.

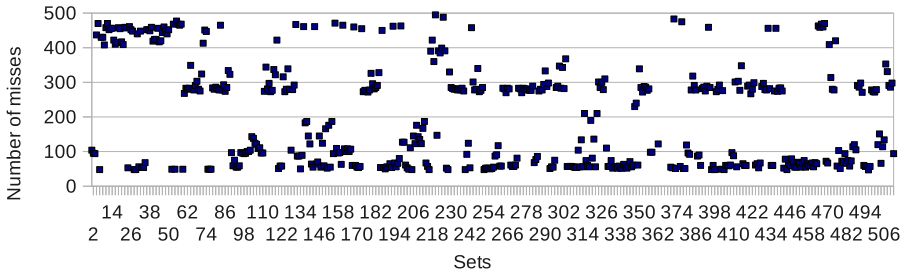


FIGURE 2.3: Set usage profile of a bank in a TCMP. In case of CCMP too, this non-uniform pattern exists.

2.1.3 Dynamic Associativity Management (DAM)

DAM is a technique that can be used to handle the non-uniform load distributions among the cache sets [37, 34]. In DAM based techniques, the associativity of each set can be managed dynamically without changing the actual size of the cache. The heavily used set can use the idle ways of underused sets, thereby effectively increasing its associativity. This will in-turn reduce the conflict as well as capacity misses inside the cache sets. Several DAM based approaches already exist in literature [37, 34, 38]. CMP-SVR [34] is one such DAM based approach that has less hardware overhead compared to other similar techniques. It improves the performance of the cache by reducing the number of conflict misses in the cache.

The non-uniform distribution of memory accesses is a major cause for higher conflict misses in LLC. Associativity of the conventional set-associative caches cannot be adjusted dynamically because of the static one-to-one mapping between the tag entries and data lines. Note that, all of these techniques can be implemented in multi-banked SNUCA caches. In this thesis, we use CMP-SVR [34] to improve the performance of some heavily loaded cache banks, during and after cache resizing.

2.1.3.1 CMP-SVR

Here, the ways of each set are divided into two parts: Normal sTorage (NT) and Reserve sTorage (RT). NT behaves same as conventional cache and RT takes 25% to 50% ways from each set. The sets are divided into multiple groups; each group is called as fellow-group. The sets within a fellow-group are called fellow-sets. A

heavily used set can use the idle reserve ways of its fellow-sets. In this way a heavily used set can distribute its load to some other underused sets. Such way sharing improves the utilisation of the cache and hence improves performance.

Each fellow-group has its own RT section which is created by the reserve ways of all the fellow-sets within the fellow-group. A fellow-set can use any RT locations of its fellow-group but cannot use any location outside its fellow-group. Searching the entire RT, reserved for a particular fellow-group, directly from the cache is an expensive operation as it requires us to search all the sets within the fellow-group. To overcome this problem an additional tag-array, SA-TGS is used for RT. In SA-TGS, each RT location (L) has a corresponding entry, which contains the tag address of the block currently residing in L. An example of CMP-SVR is given in Figure 2.4.

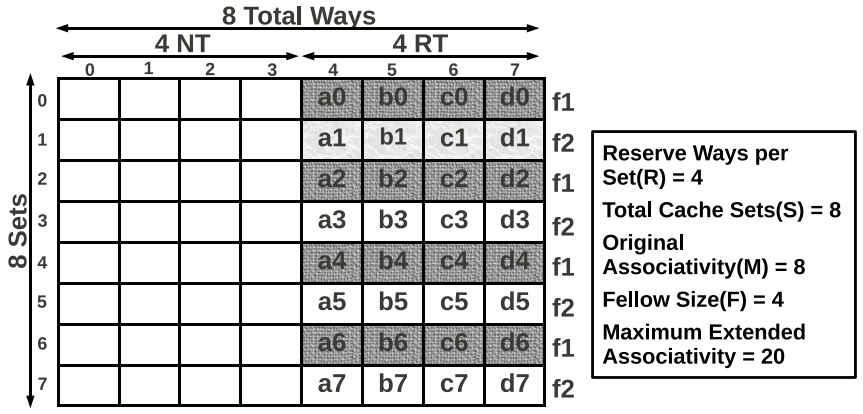


FIGURE 2.4: An example of CMP-SVR

CMP-SVR does not search RT directly, instead it searches the SA-TGS which contains a dedicated location for each corresponding RT location. If a block is not available in its dedicated cache set (i.e. in NT) then the block may be available in RT. In CMP-SVR, tag matching for a block in NT and SA-TGS (not in RT) is done simultaneously. If the tag is found in NT then it is a *direct hit* and if the tag matches in SA-TGS then it is an *indirect hit*. In the case of an *indirect hit*, the block is moved from RT to NT. Moving a block from RT to NT is easy, provided NT has free (invalid) way, otherwise it swaps with the LRU block of NT. During replacement, instead of removing a victim (LRU) block completely from the cache, it moves to RT. Moving a victim block (V) to RT is easy if RT has a free

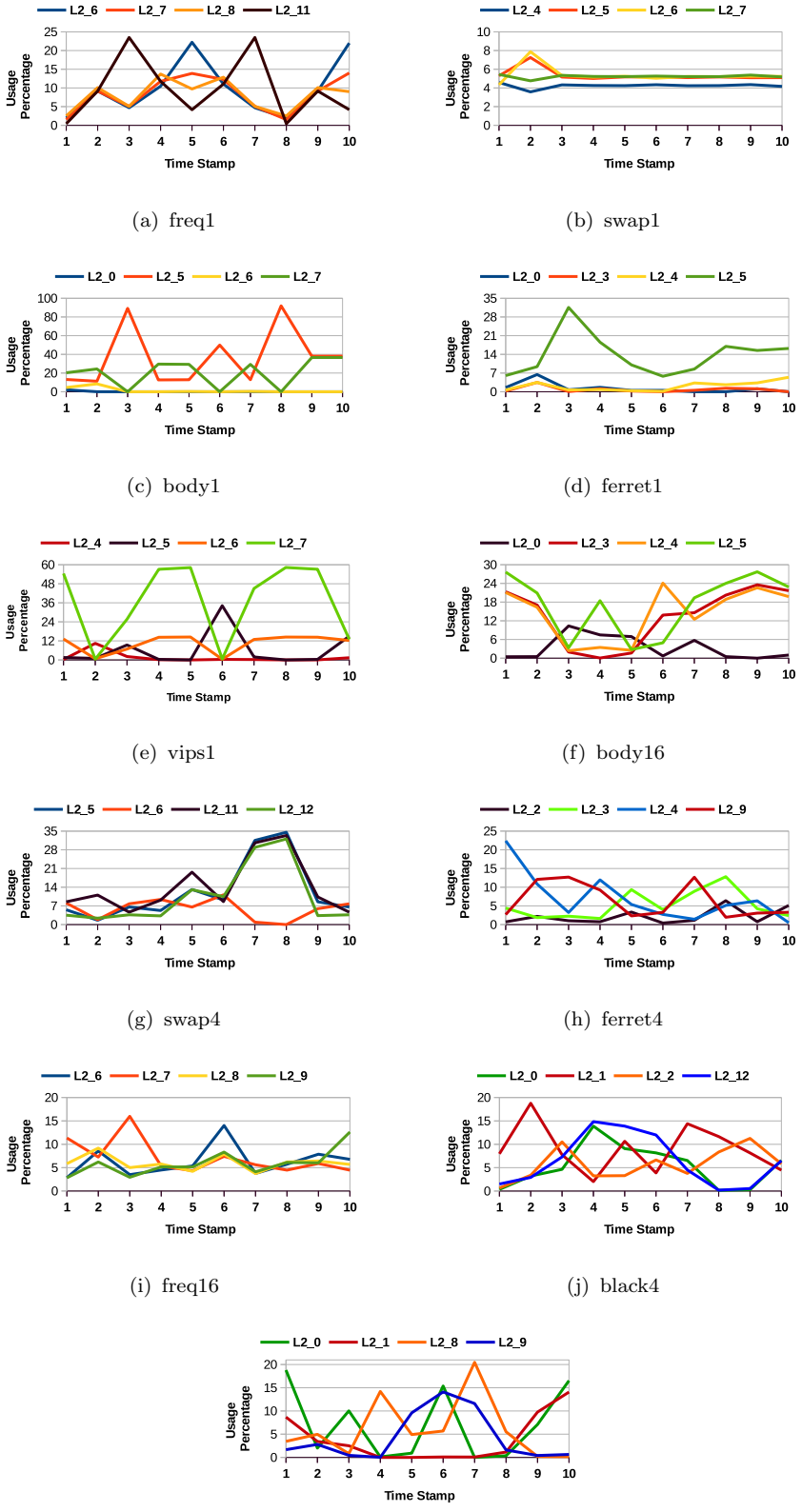
(invalid) location; otherwise it has to replace the LRU block of RT. However, an experimental evaluation with a full system simulation shows CMP-SVR improves overall system performance by 8%, while reducing miss rate by 28%.

2.1.4 Violation in Locality of Reference

Although cache accesses exploit Locality of Reference, but, in the case of long running applications, this property violates. The change in access patterns over long time-span for a set of PARSEC applications are shown in Figure 2.5 for a TCMP architecture. For each application, 4 banks out of 16 have been randomly selected to maintain clarity in the figures. Here, X-axis represents uniform time-intervals. Figure 2.6 depicts the changes in cache access patterns in the case of a CCMP. The access patterns are shown along y-axis for the cache banks over different epochs for black16, body16 and fluid16, respectively. In this case, we have taken 5 banks (along the x-axis) from each of the benchmarks, to show heavily used, moderately used and least used banks in each epoch. Epochs imply a time-span of 10 million cycles, just after the warm up (*Epoch-1*), at the middle of execution (*Epoch-2*), and at the end (*Epoch-3*). The figures show that, for all the applications, accesses to a bank change over the three different epochs. A heavily used bank can become a lightly used one later, or vice versa.

Finally, from the above discussions, following observations can be listed:

1. Diversity exists in cache access behaviour across the applications with significant changes during execution.
2. *Locality of Reference* with respect to *bank id* may not be exploited over a long-run.
3. Access behaviour for the banks are diverse in nature and also change during execution.



(a) freq1

(b) swap1

(c) body1

(d) ferret1

(e) vips1

(f) body16

(g) swap4

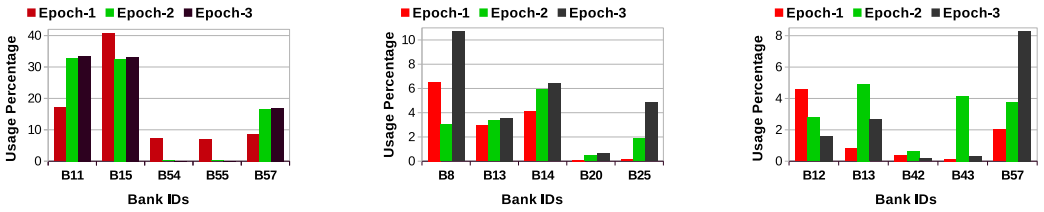
(h) ferret4

(i) freq16

(j) black4

(k) fluid16

FIGURE 2.5: Temporal Change in Bank Usages for different applications in a TCMP having 16 banks.



(a) black16 (b) body16 (c) fluid16

FIGURE 2.6: Change in cache bank access behaviour over time in a CCMP having 64 banks.

From a performance perspective, these observations indicate cache bank turn-off during less utility and turn-on when in greater demand, while saving cache leakage consumption.

2.2 Cache Energy Modeling

2.2.1 Dynamic and Leakage Energy

Energy consumption of the SRAM cells can be divided as follows [39, 7, 21] :

$$E_{total} = E_{Dynamic} + E_{Static} + E_{off-cache} \tag{2.2}$$

$E_{Dynamic}$, the Dynamic energy, is consumed during read or write accesses of the cache blocks. As writing energy is computed by the similar set of equations like read energy, hence, only the equations for read accesses are provided here to avoid redundancy. $E_{Dynamic}$ for a read access is computed as follows:

$$\begin{aligned} E_{Dynamic} = {} & E_{dyn-read-req-net} \\ & + E_{dyn-read-data} + E_{dyn-rep-net} \end{aligned} \tag{2.3}$$

Here, $E_{dyn-read-req-net}$ denotes the energy consumption per read request and $E_{dyn-rep-net}$ represents the energy consumption for replying a read request. Energy consumption during accessing data array, i.e. $E_{dyn-read-data}$, can be written as follows:

$$\begin{aligned} E_{dyn-read-data} = E_{dyn-predec-blks} + E_{dyn-dec-drivers} \\ + E_{dyn-read-bitlines} + E_{dyn-senseamps} \end{aligned} \tag{2.4}$$

The components $E_{dyn-predec-blks}$ and $E_{dyn-dec-drivers}$ imply dynamic energy consumption of predecoder and decoder drivers, respectively. Sense amplifiers' dynamic energy is $E_{dyn-senseamps}$ whereas dynamic energy for reading bitlines is $E_{dyn-read-bitlines}$.

E_{Static}, on the other hand, represents static/leakage energy of the SRAM cell, having direct dependency both upon the running temperature and supply voltage. Modern CMPs with 32nm or lesser technology are equipped with larger LLCs, which increases on-chip transistor counts with shorter channel length, inherently increasing the power density and in turn higher leakage energy consumption. The total leakage energy consumption E_{Static} can be written as follows:

$$\begin{aligned} E_{Static} = E_{leak-req-net} + E_{leak-data-array} \\ + E_{leak-rep-net} \end{aligned} \tag{2.5}$$

$E_{leak-req-net}$ and $E_{leak-rep-net}$ represent leakage energy consumption for the request and the reply networks, respectively. Leakage for data array is denoted by $E_{leak-data-array}$ which is further divided into predecoder's leakage, decoder driver's leakage, sense-amplifier's leakage and leakage of memory cells:

$$E_{leak-data-array} = E_{leak-predec-blks} + E_{leak-dec-drivers} + E_{leak-mem-cells} + E_{leak-senseamps} \tag{2.6}$$

However, the energy consumed by accesses that go to the next level from the L2-cache is called the off-cache energy. We assume the next level DDR2 system which needs $E_{off-cache-access}$ energy per access having a value of 12200 pJ [40]. Thus, the overall value will be as:

$$E_{off-cache} = \#off\_cache\ accesses \times E_{off\ cache\ access} \tag{2.7}$$

2.2.2 Channel Length, Temperature and Leakage

The drastic reduction in channel length reduces the circuit capacitance, which in turn increases the operating speed. With the sound improvement in performance, these transistors with reduced channel length suffer from unwanted side effects: traditionally called as **Short Channel Effects** [41]. The short channel designs exacerbates sub-threshold leakage current that arises from some induced electrons in the channel, even before the establishment of the strong inversion. Basically, this sub-threshold current is made worse especially by the DIBL (Drain Induced Barrier Lowering) effect that increases the injection of electrons from the source.

Modern CMPs are usually fabricated with the transistors having channel length of 32nm or less [5, 4, 7, 21], whereas the short channel effect becomes prominent in the transistors having a channel length of 45nm or less. Thus, leakage consumption of such CMPs plays the pivotal role during its designing phase. The increased leakage consumption along with the dissipated dynamic power together increase the effective chip temperature, which can be written as follows [28]:

$$T(t) = (P'_d.V^2(t).f(t) + \zeta_v + P'_s).R \tag{2.8}$$

where, $T(t)$ is the effective chip-temperature at time t, $P'_d \triangleq \zeta_T.P_d$, $P'_s \triangleq \zeta_T.P_s$ and R is a system's constant. ζ_T and ζ_v are the temperature-leakage coefficient and the voltage-leakage coefficient, respectively, where $V(t)$ is the supply voltage at time t. P_d implies dynamic power and P_s is the leakage power consumption of the chip circuitry.

However, this raised up chip temperature further increases the leakage power consumption as leakage has a quadratic relationship with the circuit's temperature (ref. Equation 1.3). This is how, leakage forms a circular relationship with the chip temperature. As seen earlier, LLCs consume a significant portion of the total on-chip power, and majority of it is coming from leakage, hence, we simulated the contribution of an LLC to the total power consumption of the chip in Figure 2.7 for different temperatures. The change in leakage power for the LLCs are also shown in Figure 2.8 and 2.9. These figures claim that, on account of large number of transistors, the leakage consumption is more in larger caches compared to the smaller ones. Figure 2.10 further shows the domination of static power over dynamic power consumption for an 8MB L2 cache at a fixed temperature of 350K. These values are generated by running a set of PARSEC applications in our simulation framework (which will be discussed in the next Chapter).

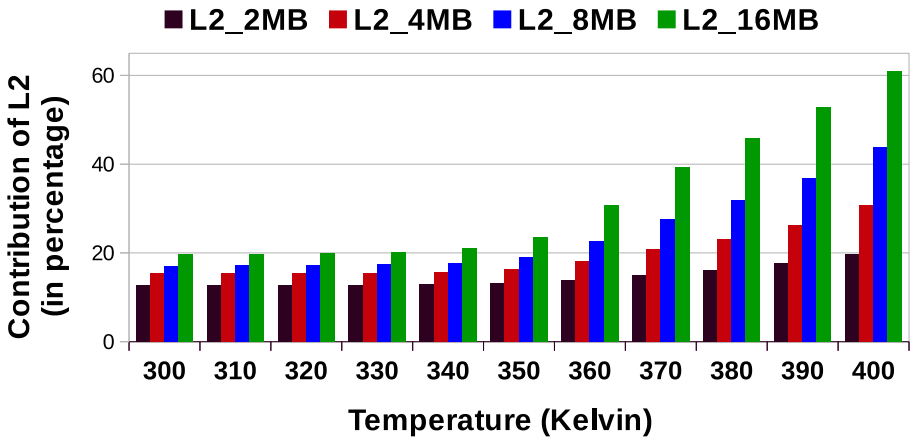


FIGURE 2.7: Contribution of LLCs to the total Chip power consumption.

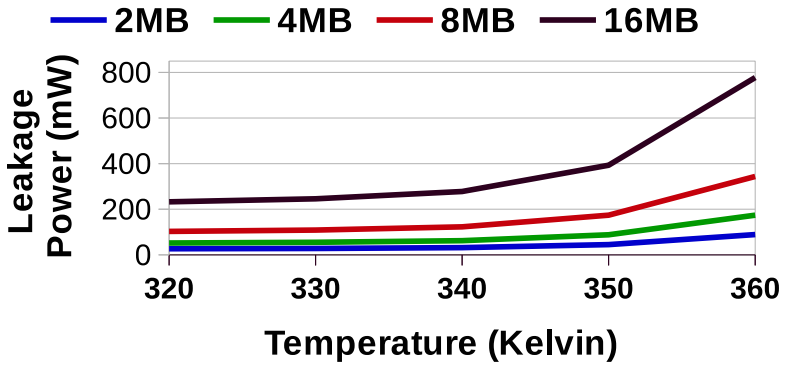


FIGURE 2.8: Increment in Cache Leakage Power in low temperature.

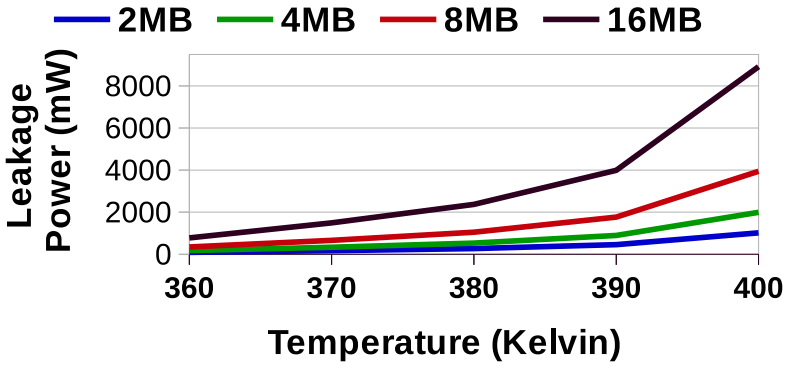


FIGURE 2.9: Increment in Cache Leakage Power in high temperature.

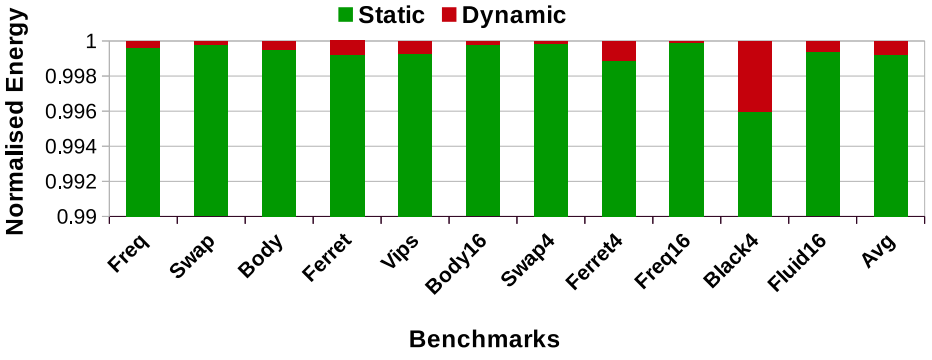


FIGURE 2.10: Distribution of Power Consumption in an 8MB L2 cache.

2.3 Reducing Cache Leakage Consumption

In modern era of green computing with excessively shrunk transistors, leakage minimisation has become a topic of paramount importance across the VLSI research community [42]. The primary objective of the modern CMP design is now

focusing towards minimisation of leakage energy, at the end of Dennard's Scaling [8]. From our earlier discussion, it can be stated that, on-chip LLCs are one of the principal contributors to the total power consumption of the CMPs, and majority of which is coming from its leakage consumption. Plenty of techniques that minimise leakage consumption in caches are constructed based upon the cache (size) tuning methods, where in most of the cases performance is considered as a system-wide constraint. These cache tuning techniques towards minimising cache energy consumption can broadly be classified into two major classes-(a) Off-Line techniques and (b) On-Line techniques [4, 5]. The former one decides about the required cache space/configuration at the design time or during some pre-execution phases, whereas the latter one tunes the caches during execution and may or may not continue tuning over the whole span of execution. Figure 2.11 classifies all of these state-of-the-art on-line and off-line techniques into more sub-categories.

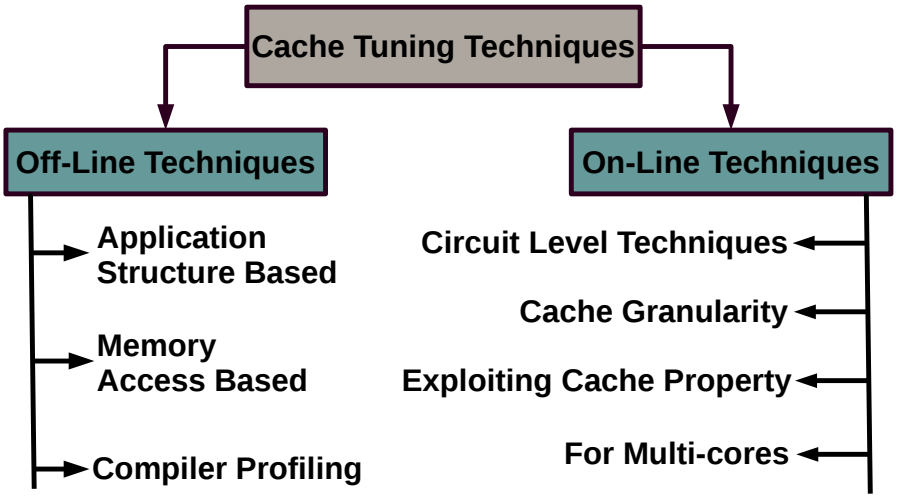


FIGURE 2.11: Classification of cache tuning techniques from a power/performance perspective.

2.3.1 Off-Line Techniques

Off-line cache tuning is also termed as static cache tuning, where designers usually evaluate the applications' structures and system configuration before execution, in order to determine the cache configuration. This configuration remains unchanged during the process execution. Mostly, cache sizes and other configuration parameters are determined off-line through analytical formulations [43, 44, 45, 46, 47, 48,

49, 50, 51, 52] or by evaluating the compilation process [53, 54, 55]. The analytical formulation for cache size determination uses mathematical models which can spontaneously compute the cache misses [4]. This cache size prediction at design time reduces the run-time cache tuning overhead. Hence, in this section we will discuss about some off-line cache size prediction and tuning techniques.

2.3.1.1 Application Structure Based

The spatial and temporal behaviours of the applications determine its run-time cache access patterns, which are actually profiled from the application's branch & control instructions' characteristics. In their work, Ghosh et. al. [47] generated cache miss equations to summarise the cache access behaviour of the loop and its variables. However, the direct computation of cache misses from these equations is an NP-Complete problem. Later in 2004, Vera et. al. proposed an approximation algorithm to estimate the cache misses during execution [44]. But both of these methods have limited applicability in case of applications having perfectly nested loops in their source code. In another work [45], Presburger formulas are used for further extension of the model [47] to include some non-linear array structures. Based upon this, Kim et. al., in their work [56] proposed an estimation of energy efficient memory model (as an extension of [45]) for some video encoding applications. This model shows $70 - 80\%$ accuracy with respect to simulation results. At this same time-frame, Harper et. al. proposed an approximation model that estimates cache misses for any loop structure [43], whereas a near optimal cache size determination was done by an another algorithm proposed later [46] through loop statistics extraction. But, this model has very limited application in modern multi-tasking environment, where thread/task interleaving and unpredicted loop structures are very common. In fact, the models proposed in [47, 44, 45, 4] are complex enough to implement in practical environment, hence, these methods are not widely used now-a-days [57].

2.3.1.2 Memory Access Trace driven

Most of the analytical formulations that estimate/determine (sub-)optimal cache size are almost similar like the earlier ones stated in section 2.3.1.1. However, these methods are usually formulated based upon the memory *reuse distance* between two successive accesses at the same memory location. Ding and Zhong predicted the whole program behaviour with 94% accuracy by analysing the data reusability [58]. Their algorithm analyses the LRU stack distance, pattern recognition and distance based sampling to predict the memory footprint of the process, however, the idea of reuse distance was earlier incepted by Mattson et. al. in 1970 [51].

Later, for the CMP architecture, a different reuse distance prediction model was proposed in [59], which uses the accesses related to the threads while considering the inter-thread contention at the shared cache space. This model although predicts the cache misses for individual threads but, does not consider the cache contention. Later in 2009, a fine grained cache contention was predicted [49] for a multi-threaded environment with a prediction accuracy of 92%. This model also considers system-wide CPI for predicting cache accesses in absence of cache contention. In another study [48], authors have predicted the cache miss rates in a CMP with a high prediction accuracy of 98%, by exploiting the hardware performance counters while running an application on real processor cores. A low-cost profiling technique was further developed to analyse the full execution with a guaranteed precision [52]. All of these developed techniques consider the cache contention models but data sharing among the threads was given lesser importance. In a latter work by Ding and Chilimbi [60], thread interleaving and cache contention are studied together while predicting cache usages with an accuracy of more than 90%. Shi et. al. further studied the data replication across the private and shared caches [61] during prediction of cache access behaviour, having less than 9% error margin using only about 8% of simulation time. But all of these discussed methods are suffering in case of some modern applications where cache access behaviour changes as the execution proceeds.

2.3.1.3 Compiler Profiling/OS level approaches

Apart from analytical prediction of cache behaviour for energy reduction, compiler can also assist to reduce leakage consumption by anticipating the cache requirement through code analyses. Zhang et. al. in 2002 designed a compiler based leakage reduction technique for instruction caches [54], that reduces leakage more than 40% on an average. The whole technique is divided into two parts: (a) Conservative approach, that detects dead instructions and puts those cache lines in a low leakage mode; and (b) Optimistic approach, that puts the cache lines into low leakage mode when the next instruction is detected as dead. The cache leakage power consumption is reduced by putting the lines into both state preserving and state destroying modes whichever is necessary. In another work [55], code has been restructured at compile time for array based and pointer-intensive applications for lowering data cache energy, which is beneficial mostly for the heavy stand alone applications. This policy saves 59% leakage with a performance penalty close to 4%.

Reddy and Petrov designed a profile based off-line algorithm [62] that identifies a beneficial cache partitioning. During context switching OS selects the proper configuration that has to be kept unchanged upto the next context switching. The cache parts are activated or deactivated according to the selected configuration. A $40-80\%$ of leakage saving has been reported in this article. Wang and Mishra, also accumulate the execution profile from the prior executions and heuristics are generated accordingly [63]. These heuristics are used at run-time with proper selection of active cache portions. This method is proposed for a multi-tasking soft real-time environment, with having a significant leakage reduction of $32-49\%$, while maintaining performance. In 2011, Wang et. al. proposed a better algorithm [53] which judiciously finds a cache configuration by analysing static profiling of the applications those executed earlier. This policy works in both private as well as shared caches, and saves leakage upto 30%. As leakage power has a quadratic relationship with temperature, hence, Noori et. al. developed a Temperature Aware Configurable Cache (TACC) [64] that shrinks cache during

high temperature and expands the same in low temperature with a slight penalty in performance. The reduced cache temperature noticeably reduces the cache leakage, by more than 60% for higher level caches at its best.

Most of the off-line techniques mentioned above reduce leakage consumption of the caches, however they have some limitations [4]:

- Most of the time, off-line techniques need prior information about the applications that have to be executed.
- Analytical methods are complex and may not be a good predictor for some modern applications where cache accesses may change for its every occasion of executions.
- Compiler based methods need detailed analysis of the source code which may slow down the compilation process.
- Additionally, violation in Locality of Reference for modern long running applications demands on-line algorithm for better anticipation of the future cache footprints to save leakage accordingly.

Hence, architects move to the on-line solution for better tuning of caches during execution.

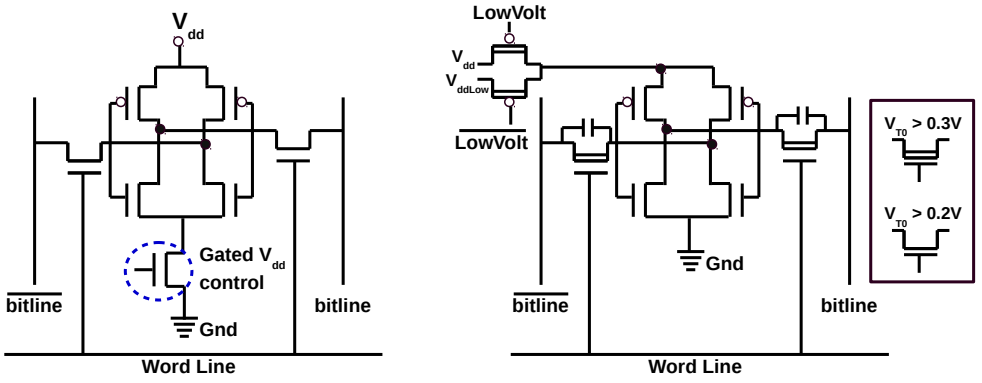
2.3.2 On-Line Techniques

The development of architectural techniques need some circuit level backbone to ensure their proper implementation, therefore firstly we will discuss about some circuit-level or micro-architectural level techniques those are exploited in most of the dynamic architecture level cache tuning techniques.

2.3.2.1 Circuit Level/Micro-architecture Based techniques

Since, on-chip transistor count determines the circuit leakage, hence, deactivation of these transistors can be a promising option for reducing leakage consumption.

The deactivation may be done for the unused or least used cache blocks/sets, for maintaining performance within a given threshold, if any. Gated-V_{dd} is a technique that turns-off an SRAM cell by disconnecting power and/or ground rails with an extra gated-V_{dd} transistor [65]. The corresponding circuits for an SRAM cell is depicted in Figure 2.12(a). The contents of these turned-off cache locations are lost along with their tag entries. Hence, this method of power saving is termed as state destroying policy. This technique now-a-days has become the backbone for most of the state destroying leakage saving techniques in cache memory [66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77], as in a large scale the leakage reduction is remarkably high. On the other hand, during cache space requirement, these cells can also be turned-on by establishing the connection as it works normally.



(a) Gated-V_{dd} [65] (b) Drowsy Cache [31]

FIGURE 2.12: Cache leakage reduction method at circuit level.

Rather than complete shutdown of the cache portions as it is in gated-V_{dd}, a reduced but non-zero supply voltage can also save some amount of leakage, with proper preservation of the meta-data related to the cache blocks. This technique is known as drowsy cache, which compromise the cache power supply between fully active mode and turned-off mode [31]. However, as the SRAM cells are not turned-off fully, this policy does not save leakage as much as gated-V_{dd} [65], but, drowsy cache has lesser waking up overheads that is much lesser than a number of cache misses which occurs in case of gated-V_{dd}. We show the schematic view of an SRAM cell in Figure 2.12(b) after incorporating drowsy cache circuitry. This circuit uses a couple of extra PMOS gate switches for supplying normal voltage (V_{dd}) and low voltage (V_{ddLow}) to the SRAM cell [31]. Several micro-architectural

based leakage saving techniques use this policy to save leakage under some strict performance constraints [78, 79, 80, 81, 1, 82, 83, 84].

2.3.2.2 Optimisation at different Cache Granularity

By exploiting either or both of the state preserving and destroying techniques, researchers have contributed different leakage saving policies at various granularity levels of the cache. Based on these granularity levels, these techniques can be classified as follows: (1) Way level, in which cache tuning is performed by changing the cache associativity through power gating some cache ways or put some ways into drowsy mode [1, 85, 86, 83, 84, 87, 88]. (2) Set-level or bank-level, which is usually performed by gating a number of cache sets/banks [89, 90, 91, 92, 75]. This policy is exploited in case of multi-banked NUCA caches. (3) Block level, where each gating can be controlled at block level inside the cache sets and hence, the cache reconfiguration will be more fine tuned [66, 93, 77, 70, 72, 94].

In case of way-level granularity, changing in cache associativity increases the capacity and conflict misses. Although in state preserving way level leakage reduction through drowsy cache Fitzgerald et. al. saves more than 40% leakage with negligible drops in performance [1]. In another work, implementing a state destroying technique in a DNUCA cache reduces leakage significantly by deactivating $40-75\%$ cache ways which saves a noticeable amount of EDP by 30%, while degradation in performance is close to 3% [85, 92]. The frequently used data will be brought to the nearest possible cache location of the core accessing it and the least accessed blocks will be brought to the farthest possible location and will be turned off eventually. Turned-off cache portion saves cache power whereas bringing MRU data to the nearest possible cache line improves performance. However, this policy considered a uniprocessor system hence, effect on coherence is not studied which may have some adverse effect on NoC performance while being attached with DNUCA. The dynamic cache reconfiguration policy can be implemented more efficiently when the program behaviour including its memory footprints are well known in advance. A trace driven dynamic cache proposed by Tsai and Chen [86] collects

the traces both from the compiler outputs and instruction caches. These traces are stored in a special trace history hardware which are used in latter executions of the same program. This trace history helps in prefetching of instructions in the private cache and lessen the fetching overhead both in terms of time and energy consumption. Authors claim a 75% reduction in energy consumption with 20% improvement in IPC. In spite of these impressive results, on the other end of the spectrum, this policy may suffer from the following overheads-(1) system cost may be increased due to incorporation of a new hardware, (2) the hardware is evaluated in a 90nm technology nodes that has lesser power density, where in modern CMPs having technology node of size 32nm or less may badly suffer from heavy leakage consumption, (3) additionally, the history based/compiler based policies are more useful when same known applications are running repeatedly [83].

Furthermore, shutting down of cache lines can also save energy by 65% [75], but gating circuitry needs to be attached at every cache line. Additionally controlling individual cache lines can further increase the controller's overhead. Hence, in most of the modern techniques apply power gating at set/way/bank level. Such a policy, named "folding" has been introduced in [90] which combines a pair of cache sets to improve the performance. This gained performance has been traded off further by shutting down some cache sets and saves energy by 20% while performance degradation is less than 4%. But the run-time complexity is a bit higher in this set combined algorithm in case of larger modern cache, though authors claimed it scalable enough. At the bank level granularity, Dani et. al. proposed a bank shutdown scheme where they turn-off least used cache banks with a prediction based algorithm [91] for on-chip LLCs. This power saving policy implements their algorithm at the higher level cache controller, which in turn aggravates the performance, however, a 40% of EDP savings has been achieved by this policy.

By exploiting cache colouring, a power estimation emulator has been proposed [70] which reduces cache energy consumption remarkably with negligible performance degradation. The simulation based results claim an EDP savings of 30% that outperforms its prior state of the art techniques. The traditional caches are usually composed of tag and data arrays. In another power saving policy [72], data arrays

of the LRU cache portions are kept in the sleep or low power modes, where tags are not. Once the hit/miss is detected the corresponding data array is turned-on. This technique is well fitted in the caches which support sequential access technique. However, by 1.7% performance degradation, this technique saves a significant amount of leakage energy by more than 40%. The dynamic cache reconfiguration can further affect the performance of its lower level memories. Hence, in [77], Kim et. al. considered caches along with lower level memories and their accessories while saving leakage by tuning of caches. This method saves 31% leakage energy across the cache levels and main memory along with its communication channels.

All of these techniques discussed above are proposed at different granularity levels have their own pros and cons. To get advantages of these both, a few works also proposed some hybrid techniques [68, 64], where more than one granularity levels are mixed together to save leakage power consumption. However, the implementation overhead is increased in case of hybrid policies than the single level ones. All of these energy saving policies resize the caches dynamically, but reducing cache size to its 1/8 or lesser will drastically reduce the performance by incurring more capacity and conflict misses [5, 4].

2.3.2.3 Exploitation of different Cache Property

A set of leakage saving techniques exploit the benefits of some cache properties. Memik et. al. uses the multi-level inclusion property to save leakage [95], by attaching a small chunk of memory which keeps the cache footprint information of the blocks. This information predicts the block location and thus determination of hits or misses can be predicted before accessing the lower level. By using SimpleScalar simulator [96], authors have shown 53% reduction in misses while reducing a maximum of 12% cache power consumption. On the other hand, temporal locality is exploited for leakage saving in another set of policies [85, 92, 91, 81, 82, 97]. In [85, 92], temporal locality is exploited through way adaptation in a DNUCA cache. In another work [91], a prediction based algorithm is used to get the optimal

cache size during execution. The prediction is done by using a Bloom Filter that uses the earlier traces of the same execution. The way prediction based algorithm also has been proposed in [97], in which an additional CAM is required to get the benefit of temporal locality. To reduce the energy consumption, authors also attached a post-technique which compares partial tags instead of doing the whole. However, this technique speeds up power/performance by 15% while reducing energy consumption by 40%. In another exploration [82], a large chunk of tag bits are moved from the cache into an external register (called the Tag Overflow Buffer) which works as an identifier of the current locality of the cache references. This work achieves dynamic energy efficiency, at the same time reduction in tag-bits during comparison also reduces the leakage. The overall energy savings in this policy lies in between the range of $16 - 40\%$ depending upon the cache structural parameters. In [81], an adaptive cache reconfiguration is done that solely depends upon the local cache access behaviour, rather than considering the system-wide performance. By applying this technique overall energy saving has been achieved by 30% on an average with a performance degradation of 2.1%. Apart from studying cache locality, Working Set Sizes (WSSs) of the running applications are also studied in some of the prior energy saving approaches [98, 99, 100, 101]. During execution, the cache requirement for the next phase is predicted and accordingly the adequate cache space is provided to the applications. The remaining cache parts will be gated to save leakage. Among these four techniques mentioned here, $8 - 10\%$ of EDP saving has been noticed in [98], where in [99] and [100] $5 - 82\%$ energy reduction has been achieved for data caches. However, all of these three proposals maintain performance while saving cache energy significantly. While designing a power-aware cache in [101], more than 20% of cache energy has been saved than the conventional ones.

2.3.2.4 Techniques for Multi-Cores/Multi-Processors

Some of the cache tuning techniques used in single core can also serve well in the case of multi-core caches: e.g.-Way shutdown, bank/set shutdown, block level cache management etc. Most of the techniques usually attempt to turn-off the

unused/least used cache space [4]. The power efficient design and cache reconfiguration have to be changed in the case of multi-cores, as shared data consistency, cache coherency, thread interleaving, resource contention need to be taken into account. In 2002, Zhu and Zhang developed a speculation based energy saving technique for CMP caches [101]. The speculation technique mainly anticipates the future hits and misses across the available cache locations, which reduces the access time as well as energy. In another work, an efficient integration of cache reconfiguration in soft real-time systems with a unified two-level cache hierarchy has been done [63]. This policy collects the cache usage profile of the applications and reconfigures the cache during next execution of the same process. This method saves significant amount of energy which is in the range of $32 - 49\%$ in a real-time environment. In their off-line OS based implementation [62], Reddy and Petrov proposed an energy efficient cache partitioning method which ensures interference-free multi tasking environment in a real-time scenario. This policy shows a leakage saving more than 40% for most of the tasks (the authors used). Later in 2013, Bardine et. al. propose a novel hybrid scheme [102] based on a Drowsy and Way Adaptable techniques for a DNUCA shared cache, with a significant leakage energy saving of more than 50% and a performance degradation of 6%. In another exploration [78], level one (L1) data cache is tuned to save energy in a heterogeneous dual-core system where each data cache can have a different configuration. The authors claim a near optimal cache energy saving which is around 26% on an average for this policy with sound management in consistency and data coherency.

2.3.2.5 Frequency of Cache Reconfiguration

In the on-line cache reconfiguration, the time interval at which reconfiguration has to be triggered plays an important role. This time interval is called as reconfiguration period, that determines how frequently reconfiguration can take place. The time-span of reconfiguration period may be fixed (static) or variable (dynamic) [5]. Former one keeps the span fixed during whole execution of the process, whereas, latter one varies the period as execution proceeds. A number of energy efficient

techniques use static period [86, 98, 99, 100, 91], where experiments have been performed before fixing the interval length. According to [91], for a TCMP with 4MB or more shared LLC, 2M to 5M clock cycles can be taken as fixed reconfiguration period. However, fixed interval may suffer from reconfiguration overhead if the interval is small, conversely, in the case of a large interval the system may suffer from inadequate cache space (if the current cache size is small) or from more power consumption (if the current cache size is big). Hence, a moderate interval length has to be chosen, which has been (sub-)optimally solved in most contributions that use dynamic intervals [93, 77, 103].

The dynamically reconfigurable cache architectures directly leverage the circuit-level techniques for varying individual cache parameters at their respective levels of granularity. Based upon the cache parameter adjustment, on-line cache tuning methods reduce energy consumptions by way/bank level management, set level management and block level management. In this regard, trivially, it can be stated that, management at finer granularity offers better energy consumption tuning whereas having more implementation overheads. From our literature survey and the techniques mentioned in [5, 4], the following points can be stated:

- Way/Bank level techniques have limited cache configurability but needs minimal changes in hardware circuitry. Additionally, management at this level has enough potential to reduce bulk amount of energy with minimally changed circuitry. The operating time for this is also very less. However, dynamically shrunk caches experience more capacity and conflict misses.

- Tune cache at set level provides a bit more fine tuning of energy than the earlier one but has more circuit complexity, hence, increment in controller overheads.

- At the block level management, implementation is most critical but this management offers more energy tuning options to the users. Even complex controller takes more time to response during the process execution, eventually aggravates system performance.

In our proposed works, we therefore choose to tune the cache at bank/way level granularity having minimal implementation overheads. Moreover, we also apply DAM based policy for improving performance at the shrunk caches. Unlike an earlier exploration [91], from performance point of view, we implement our policy at the LLC controllers, thus we keep higher level caches transparent to this. Additionally, rather than using some access prediction based cache tuning policy mentioned in [5, 4], we directly exploit locality of reference and other critical performance parameters online, hence, storage overheads for maintaining history have been reduced.

2.4 Thermal Management in CMPs

Rapid progress in chip design technology reduces the channel length of modern transistors, which in turn helps to meet the ever increasing demand of high processing power in modern Chip Multi-Processors (CMPs), by integrating more on-chip components in a single chip. In addition to the noticeable increment in circuit complexity of the new generation of CMPs, this heavy integration increases the power density along with the spatial power variation, introducing severe local hotspots. These on-chip hotspots increase effective chip temperature which raises up reliability concerns of circuit functionality in addition to the higher cooling cost and performance degradation. Finding out an economical and efficient way for reducing chip temperature, still remains a challenging task for the researchers [29]. Hence, in recent years, architects and designers are concentrating on dynamic thermal management techniques while designing modern CMPs. A plethora of such recently developed techniques are discussed in [29]. According to the nature of mitigation provided for thermal issues, these techniques can be classified as follows-(a) minimise chip temperature for a given performance constraint, and (b) maximise performance for a predefined power budget and/or thermal constraint. Both of these two strategies attempt to minimise the power consumption dynamically either at CMP cores or at caches.

2.4.1 Core Level Management

The gradual reduction in channel length of modern transistors fits more components in a single chip, resulting into due increment in power density which creates local hotspots. DVFS and task migration are the two most promising techniques to remove these hotspots from the chip, by controlling the dynamic power consumption of the cores. Leakage power, on the other hand, having a circular dependency with temperature, is efficiently modeled in [28]. The authors have linearised the problem through a piece wise linear approximation method for estimating both temperature and leakage as much realistic as possible. This thermally constrained policy shows 19.6% performance improvement while keeping peak temperature of the chip below the preset threshold of 110°C. Additionally, an quadratic algorithm has been proposed to estimate upper bound on energy savings [104]. As a traditional common industry practice, TDP is introduced in modern CMPs. However, for a given thermal constraint, researchers have attempted to maximise throughput of the processors [105] by employing DVFS as a backbone. The detailed analyses are done by varying a set of critical parameters like initial chip temperature (45°C or more), thermal capacitance and the maximum attainable processing speed while maintaining TDP. This policy achieves a sub-optimal performance by approximating an optimal speed curve. Apart from DVFS, a latter exploration [106] proposes an efficient thread migration method for thermal efficiency. For a given thermal constraint, an optimal mixing of thread migration and DVFS has been done which shows a pareto-optimal performance for the used experimental set-up. This merging of DVFS and thread migration actively participate in scalable cooling of the cores and keeps both peak and average temperature around 370K which shows a temperature reduction around 10K. Through integration of optimal control mechanism [107] or Model Predictive Controller [108], DVFS can be achieved at circuit level as well.

Concomitant to the fact that, global thermal management suffers from scalability for modern CMPs having hundreds of cores, authors in [109] tried to reduce energy consumption in diverse runtime characteristics of the threads. From DVFS

perspective, this work initially analysed the potential loss in performance and power with an off-chip voltage regulator that slowly serves the voltage scaling request. Hence, to fasten the process they have further proposed an on-chip regulator to optimise both power and performance. Even DVFS and PCPG (Per Core Power Gating) together [110] can improve processors' throughput and put idle cores in sleep modes to reduce temperature in large CMPs. By exploiting core-to-core frequency variation, this policy improves system throughput by 57% for a 16-core CMP designed in 32nm predictive technology while obeying a temperature constraint of 100°C. Moreover, insertion of idle time slice to a core with a thermal aware task scheduling can further reduce peak temperature of the chip [111] by more than 9°C on an average for an embedded processor. This significant improvement has been achieved by proposing an approximation solution for an NP-hard task assignment and scheduling problem. This insertion of idle slots has no significant effect on system performance. The above discussed task migration and scheduling techniques are the widely used ones for thermal efficiency in case of modern CMPs supports real-time applications. The proactive dynamic task migration from hotter cores to colder area can reduce the chances for thermal imbalance [112, 113]. This policy maintains a stable core temperature around 80°C during run-time. Although thermal stability is a big concern while managing chip temperature, but, frequent task shifting towards thermal efficiency in this work leads to a costly on-chip ping-pong effect. Mizunuma et. al. in [114] tried to reduce this ping-pong effect by implementing a core search based task migration. This policy reduces migration counts by 39% and overheat time by 22.4% for a negligible performance penalty of less than 2%.

From the available literature and our above discussion, the following characteristics can be listed for the core based DTM techniques:

- Reducing cores' temperatures is the primary concern for the modern chip designers, as on-chip hotspots are mostly created at the cores.
- Task Migration and DVFS are the most promising and preferable options for the core temperature reduction.

- The research directions towards core based DTM are bisected into two broad avenues: (a) improve performance under a set of thermal constraints, or (b) for a certain performance constraint maximise the reduction in the core temperature.

- Thermal imbalance or instability is also a major point of interest while designing some DTM techniques.

- Most of the cases, task migration and DVFS may have noticeable impact on system performance. Especially, task migration is only possible with the presence of some idle cores in the system. On the other hand, uncontrollable use of DVFS can potentially incur huge temperature fluctuation at the cores which may severely damage the chip circuitry.

2.4.2 Cache Based Policies

The large LLCs of modern CMPs are accounted for their significant leakage power consumption [4, 5, 29]. Reduction in LLC leakage can be done either by (a) cache resizing or by (b) reducing cache hotspots. The classical techniques for cache leakage reduction exploit both state preserving and state destroying policies [115, 67]. Even Drowsy cache [31], a state preserving technique, can also reduce leakage power alone, significantly.

A *Power density-Minimized Architecture (PMA)* with a *Block Permutation Scheme (BPS)* [116] decreases cache temperature for leakage reduction by exploiting gated-Vdd [65] at circuit level. This policy gives 53% leakage reduction with compared to a conventional cache design, and 14% with respect to a cache architecture having no thermal-aware power reduction scheme. Leakage also plays the pivotal role of thermal control for a performance cognizant thermal efficient technique, shadow tag [117, 118]. The authors claim a reduction of $7 - 15°C$ in peak temperature of the CMP while using a 3D architecture. This policy uses state destroying cache energy reduction policy towards controlling chip temperature. Noori et. al. further

analysed the increment in cache energy and temperature along with their inter-dependency for a 100nm technology [119]. This study reveals the requirement for incorporation of dynamic cache resizing during change in chip temperature, as higher temperature potentially increases the leakage power of the caches, which has an adverse effect on chip circuitry. In a latter exploration [64], Temperature Aware Cache Configuration (TACC) has been proposed to optimally reconfigure caches at different execution phases by a combination of the offline and online analysis of cache usages. Results show that, this technique reduces leakage power by 61% for instruction caches and 77% for the data caches, while TACC is compared to a configurable cache which is configured for only the corner case temperature of 100°C. Additionally, this policy also shows significant performance improvement which is more than 15% for both types of caches. Sentry tags, in another work [120], eliminate unnecessary cache accesses to minimize power and temperature. By applying this policy, "a peak to peak" cache temperature reduction has been achieved by 10°C for both instruction as well as data caches at the steady-state. No significant performance aggravation has been reported for this policy. Moreover, prediction based core level cache block migration also reduces cache temperature where migration is triggered due to overheating [121].

Modern 3D CMPs suffer from high power density, hence, the increased effective chip temperature [122]. A runtime thermal management is proposed for 3-D chip at way level granularity in [123] which combines DVFS with a novel thermal aware technique for hybrid (MRAM/SRAM) cache. This technique reduces both cores and cache (MRAM/SRAM) layers' temperatures more than 5°C while maintaining critical temperature of the chip.

From the above discussions regarding cache based thermal efficient strategies, we can draw the following conclusions:

- Constructed with shrunk transistors, the SRAM based modern caches are accounted for their heavy leakage, that has a circular dependency on effective temperature.

- From the available literature, it has been noticed that, large SRAM based modern LLCs can potentially increase chip temperature by generating hotspots at the large cache area, although in earlier designs caches are usually considered as colder on-chip components.

- Reduction in cache temperature can only be done by applying either of these two classical techniques: (a) power gating or (b) regulating supply voltage (i.e. drowsy cache). Power gating reduces more temperature than the latter option as gating the power supply at cache area drastically reduces the power consumption.

- Most of the cache techniques developed earlier are attempting to reduce temperature by controlling the cache accesses, i.e. by reducing dynamic power. But, modern thermal cache design also needs attention towards leakage reduction. Furthermore, most of these techniques are built with the larger sized transistors having a minimum channel length of 65nm, whereas recent designs have a channel length of 32nm or lesser.

- By considering the superposition and reciprocity theorem [33], it can hypothetically be stated that, turned off large cache portion can create on-chip thermal buffer which can significantly reduce chip temperature.

- Additionally, cache based policies impact the computation according to its sensitivity towards the dynamic WSS of the process, but can maintain a stable thermal profile.

2.5 Summary

CMPs, the basic building block of modern computing systems, have a number of cores integrated with multi-level on-chip caches. On-chip LLCs are the biggest in size across the cache hierarchy and occupy the largest area on the chip. Usually, these LLCs are divided into multiple banks and shared among the cores. The short channel length of modern transistors further increases the on-chip power density

which in-turn increases temperature and leakage power consumption [5, 4, 29]. According to our survey and simulation analysis discussed so far, it can be concluded that, reduction in both leakage power and chip temperature are the supreme design concern for the architects. Especially, LLCs are the major contributors to the total on-chip power consumption, where LLC leakage dominates the other power components. Additionally, LLCs are evaluated as the comparatively colder on-chip components, but from the earlier discussion it can be stated that LLCs can also generate on-chip hotspots.

Reduction in effective LLC area can reduce its leakage consumption significantly, which can be done either by some off-line techniques or through some on-line techniques. The existing diversities in cache usage patterns of the modern applications in addition with multi-tasking environment motivate us to tune the cache size on-line in either direction; that is, provide ample amount of caches to the applications when it is required and shrink the cache size with reduction in WSS. Reducing cache size through state destroying policy can aggressively reduce the leakage but may incur stall cycles which can degrade the system performance. To address these issues, we propose an on-line cache tuning policy that resizes the cache at bank level granularity based upon the locality of reference and system performance.

Power consumption in a semiconductor circuitry excogitates the thermal issues. Hence, reduction in cache leakage can also help us further to reduce effective chip temperature. In the next part of the thesis, we therefore propose thermal aware cache tuning, which reduces the cache hotspots by turning them off. As leakage has a quadratic relationship with the temperature and by forming a circular dependency these cache portions can also increase the chip temperature, hence, it makes sense to turn-off the least used cache portions. The gated cache banks form on-chip thermal buffer which distributes the generated heat to the components in its close proximity by exploiting reciprocity and superposition theorem of heat transfer [33]. Thus the effective chip temperature gets reduced. We also design thermal aware cache resizing for a CCMP, where cache resizing follows some predesignated patterns to reduce temperature. Both of these thermal aware designs

are performance cognizant and hence, they also turn-on cache banks whenever required.

Chapter 3

Simulation Framework

This chapter elaborates the simulation environment that we have used in our works. All the experiments reported in this thesis are done in a full-system simulation framework. Basically, full-system simulators are able to simulate the entire electronic systems including CMPs. The machine where simulation environment runs is called as the host machine, and the virtual system engineered by the simulator is called as target machine. Moreover, the full-system simulator provides CPU cores, along with multi-level private/shared caches, memory systems and I/O devices, which altogether produce a flavour of a real CMP. Additionally, these system components are connected through a standard NoC module that has also been integrated in this. The full-system simulator further allows to execute real programs through an Operating System (OS) platform installed independently in the target machine, unlike the instruction set simulators. The virtual device drivers of target machine also allows OS to execute all of its modules those run normally on a real hardware.

As simulators are a set of computer programs, hence, any program module developed for target machine's architecture can be modified to meet any new design requirements. For example, a conventional cache designed in a full system simulator can be modified to support Dynamic Cache Reconfiguration at both way as well as bank levels. For design space exploration, we can also easily change some preliminary parameters, like cache associativity, cache size, number of banks etc.

in target machines. A brief on computer architecture simulators are provided in the next sections with their importance in industrial and academic research.

3.1 Computer Architecture Simulators

Simulation is a useful technique for analysing performance and power consumption of any computer system. Among a variety of simulation techniques, computer architects are usually interested in emulators, trace-driven simulators and discrete-event simulations [124]. Although emulators are a kind of simulators, still they have hardware design limitations, hence, we use a set of discrete-event and trace driven simulators in our experiments. A computer architecture simulator is a software that models the hardware devices for analysing power/performance of the modeled system. It can either model a (a) target microprocessor called instruction set simulator, or a (b) full system simulator that simulates the whole computing system.

Physical design of modern CMPs are really expensive and complex to design [125]. Furthermore, for an experimental analysis, changing different parts of it are also required before deployment of the design. For example, while designing any of our reconfiguration technique, we need to have certain level of illustrated results which can produce one or more design choices for the next generation architecture. In this regard, after implementing our technique in the simulator, we have to test it for different number of cache banks, different associativity or may be with different cache sizes. Prototyping and designing some of these real set of hardware are impractical in academic scenario [126, 127]. The expensive tools and complex designing process for making hardware are not possible for an academic research project where a single project comprises of a bunch of design proposals. In our case, reconfiguration of caches at different granularities with their detailed energy and thermal analyses are highly required and recommended. In addition to that, performance analysis also plays a major role while designing architecture. Hence,

to check design accuracy with its power/performance analysis, architects use simulators for their experiments in a timely and cost-effective manner [128, 125]. Most of the prior works listed and described in Chapter 2 are implemented in full-system simulators.

SimOS [129], one of the oldest machine simulator, simulates hardware of a machine by using the services provided by the underlying operating system. Later in 2002, SimpleScalar [96] has been developed which could simulate a set of super-scalar processors. Although this simulator does not support multi-core systems. However, shortly after that, a number of simulators have been designed, for event driven simulation [130], fault analysis [131], and verification tool for microprocessor based on virtual machine [132]. The simulators as they run on some real systems, hence, their performance depends upon the real hardware of the host. Apart from that, some simulators have inherently sequential nature that runs slower than the others who can be parallelised. Despite that, if the real hardware boosts up its processing power, the full-system simulators can model the actual systems without sacrificing performance [125]. Many full-system simulators, developed over a decade or more are available for various requirements [132, 133, 127, 134, 135, 125, 131, 130].

Full-system simulators can be categorised into two categories: functional and timing [126]. Functional simulators mainly replicate the activities or functionalities of an actual system (for which the simulator is designed). Timing simulators, on the other hand, takes care of the real-time behavior of the system that follows a discrete-event simulation technique. In addition with replication of the functionalities of an actual system it also models the timings that when a task has to be triggered. Usually, the timing simulation is required for comparing the LLC performance in modern CMPs, that execute long running applications.

3.1.1 Simics

A full-system simulator Simics [125] is used to generate a complete virtual machine i.e. the target machine which runs on top of a host machine. This simulator

is flexible enough to support a variety of processes like microprocessor design, electronic system design and verification, OS development etc. Simics supports a number of real-architecture models, such as-UltraSparc, Alpha, x86, x86-64, PowerPC, IPF (Itanium), MIPS and ARM. Simics is even fast enough while it executes realistic workloads, including the SPEC CPU2006 benchmark suite [136], PARSEC benchmark suite [6], database benchmarks such as TPC-C, interactive desktop applications, and games.

The main focus of architectural research is to design architecture for the next generation. In this thesis, we propose some power and thermal efficient TCMP and CCMP based architectures which can perform better in future with increased workload. The full-system simulation framework provided by Simics is well suited for the designing of such future hardwares without any physical overheads. Even, from industrial point of view, this quick design and verification of the newly designed hardware in Simics are also useful enough. Furthermore, designing of some softwares for such hardwares can also be done concurrently as the simulated environment provides the flavour of whole system.

3.1.1.1 Limitations of Simics

Although Simics has its own powerful capabilities, still its functional behaviour limits one from performing timing simulations which are needed to simulate CMP based systems. GEMS [126], a timed simulator has been proposed that works on top of Simics. GEMS was actually designed to simulate the complete memory hierarchy of CMP with coherence management and the on-chip communications. Additionally, timing simulation with GEMS helps to compare the performance for different CMP architectures. The details on GEMS are provided in the next section. However, GEMS can not run without Simics, thus, the functional execution is decoupled here with the timing models.

3.1.2 GEMS: An Overview

The three major modules: Ruby, Garnet [137] and Orion [25] construct GEMS [126] together. Ruby models the entire memory system of any CMP based architecture where each component like L1 cache, L2 banks, memory banks, directories etc. are modeled. Individually, each of these components are called as "machine" in Ruby. Each and every machine has its own unique id called machineID which is required to identify a machine during the on-chip communications. The machines in CMP communicate with each other through the underlying NoC, which is managed by Garnet. Any CMP based architecture that can be modeled in ruby uses Garnet for establishing on-chip communications. Garnet simulates the real-time scenarios for transferring messages through the NoC. Moreover, Garnet supports different topologies which helps to simulate NoC with variety of options. Another simulator Orion [25] is attached with Garnet for modeling the energy consumed by the underlying NoC provided by Garnet. However, the Garnet version used in our simulation follows X-Y routing. The on-chip communication cost from both power and performance perspectives are considered for all the experimental analysis done in this thesis.

The requested block (load, store, fetch) from simics processors are passed to the Ruby module of GEMS. The very first level of cache i.e. L1 in ruby detects early if the requested block is a hit or a miss. If a hit is detected then simics continues its execution, else in case of a miss, the request from the issuing core is stalled for a while until GEMS completes its simulation for the detected cache miss. The timing-dependent functional simulation in Simics is determined by ruby, through the control of timing of when Simics moves forward its simulation.

A sequencer is attached with each of the L1 cache for managing the requests from the corresponding core. In case of CMP, multiple L1s and L2s can be available and can also handle cache requests concurrently. For each of the machine, a controller is attached and all functionalities of a machine are performed by its own controller. GEMS provides a domain-specific language called SLICC for modeling of such controllers to manage all the operations of the machine itself and its communications

with the others. Managing coherence, on the other hand, is a major concern in case of CMP based cache structure that uses shared caches, hence, coherence protocol has to be implemented. Different modules of the controller manage the coherence protocol through message passing among themselves. These messages are communicated across the modules are passed through the NoC (modeled by Garnet). SLICC is further responsible for coherence protocol designing as it is a combined task of all the controllers.

3.1.2.1 CMP Architectures Supported by GEMS

GEMS supports SNUCA based TCMP/CCMP cache architecture which is very robust and can be configured with a set of varieties like, cache size, number of banks, number of tiles, cache access latency, miss penalty, hit time, network protocols etc. All of these parameters can be modified by just changing their respective values in a separate configuration file. Apart from that, many other parameters are also there which can be reset based on the configuration demand, such as-number of virtual networks, block size, cache associativity, replacement policy, network flit size etc. For coherence management, we use MESI protocol in our proposed and baseline designs, which is termed as "MESI-CMP protocol" in GEMS. For experimenting with different network topology, changes can also be permitted in the network configuration file in Garnet.

GEMS supports the baseline cache architecture where we implemented our Dynamic Cache Reconfiguration (DiCeR) techniques. We made certain changes at the protocol level to manage the associated coherence mechanism. The code written in SLICC has been modified for implementing the proposed architectures, as by simply changing the parameters our proposals could not be implemented. Furthermore, to provide supports, other internal structures of its memory systems are also modified. Finally, a new compilation of GEMS constructs the new architecture. The migration of cache blocks during bank turning-off/on increases workloads in NoC, which is also simulated in Garnet. In addition to that, the request forwarding after reconfiguration of cache are also taken care by involvement

of Garnet. The integration of Orion also simulates the corresponding changes in NoC power consumption.

3.1.2.2 Result Analysis

The GEMS-Simics integration can run real set of workloads on the simulated (modeled) CMP architecture. During execution, the simulation framework records plenty of information. Some important information recorded by GEMS during execution are:

- Total Cycle Executed: This metric is a summation of all the cycles executed by all the cores. Note that GEMS also records executed cycles for individual cores.
- Total instruction executed: Same as the above metric, i.e. individual as well as the summation of all executed instructions are recorded.
- Total L1 accesses/misses: GEMS also records total and individual access/miss counts for each of the L1 banks, which are private to their respective cores.
- Total L2 accesses/misses: This implies same like L1 accesses/misses, but in LLC (L2). The bank-wise distribution is not provided in the original GEMS but can be implemented easily on demand.
- Average network latency: Garnet outputs this parameter which is the average cycles required for individual message to communicate through the NoC.
- NoC energy consumption: Total energy consumed by the NoC during execution (produced by Orion).

Apart from that, GEMS also provides: cycle per instructions (CPI), instructions per cycle (IPC), average memory access latency (including on-chip/off-chip communication time), average link utilisation, Miss Per Thousand Instructions (MPKI) etc.

The special module, named as Profiler, embedded in GEMS manages all the results during execution. However, Profiler can be initialised at any point of time during execution, so that results can be recorded for any particular time zone during execution. Further, the Profiler can also be modified (or reset on-the-fly) if required.

3.1.3 CACTI

GEMS+Simics environment executes a set of realistic applications by exploiting its underlying architecture. However, this simulation environment can not model or simulate power, area and timing at the different cache level granularities. CACTI 6.5 [7], a simulator which is well known across the computer architecture research community, models caches upto device/circuit levels while taking some architectural parameters as its inputs like-associativity, cache size, cache level, block size, access techniques (UCA or NUCA), number of banks etc. By simulating the architecture at device level it produces cache access time, its power consumption, area overhead etc. as outputs. According to the ITRS [9], SRAM cells that construct caches can be of three types-(a) HP, called as high performance cell which are very fast in their operations, and naturally the power consumption is higher; (b) LSTP, known as low standby power cells, that consumes very low power while not in use, but accessing is slower than HP as transition cycles are incurred before reaching at its accessing mode from the low power standby mode; (c) LOP, the low operating power cells which consume less power both in active as well as in standby modes, and the slowest among the three. In our work, we consider HP cells as backbone of our cache construction. The cache access methodology used in our work is *fast*, where tag and data arrays are searched concurrently for determination of hit or miss. Apart from that CACTI also supports *normal* and *sequential* access patterns. For calculating power consumption in cache cells, transistors' channel length plays the most vital role, which is known as technology parameters in CACTI. We use transistors having a channel length of 32nm which is supported well by CACTI.

For our first two contributions we use Simics+GEMS environment to get the performance simulation, and CACTI was used to model the inner circuitry for further modeling of power. The traces from GEMS are used with the similar cache model of CACTI to get the (dynamic and leakage) power consumption of cache.

3.1.4 McPAT

McPAT 1.0 [21], a trace-driven stand-alone simulator, provides power, area and timing modeling framework with enough support for the comprehensive design of multi-core architectures. At the micro-architectural level, this simulator includes models for the fundamental components of a CMP, with support for both in-order and out-of-order processor cores, NoC, shared caches, integrated memory controllers, and multiple-domain clocking. From designing and technology point of view, McPAT further supports critical-path timing modeling, area modeling, and dynamic, short-circuit, and leakage power modeling for all device types forecast in the ITRS roadmap having a channel length in between 22nm to 90nm. McPAT has its own flexible XML input interface that facilitates many performance simulators, from which traces can be generated. Combining McPAT with a performance simulator like GEMS, enables designers for consistently quantifying the cost of new ideas in perspectives of power and performance frontiers. Note that, McPAT uses the cache memory models of CACTI for power, area and timing simulation of caches.

3.1.5 HotSpot

HotSpot 6.0 [3], a novel modeling methodology, that develops compact thermal models based on the popular and modern VLSI systems. In addition to modeling silicon and packaging layers, HotSpot further includes high-level on-chip interconnects such that the thermal impact of interconnects can also be reflected at the early design stages. This compact thermal modeling approach suits well in pre-RTL and pre-synthesis thermal analysis and is also able to produce detailed steady

and transient temperature information across the die-components. This trace-based simulator takes power traces collected over-time by running application in an integrated version of functional full system and power simulators. HotSpot generates transient and steady-state temperature values which completes the thermal simulation process. In our works, we use this simulator by integrating with the others so that a dynamic thermal aware closed loop simulation framework can be formed. Additionally, by providing the aspect ratio along with detailed area analysis, HotFloorplan module of HotSpot can generate an optimised chip floorplan through a simulated annealing method.

For our last two contributions we need to simulate power and temperature of the whole chip. In order to do so, we need to model the power consumption of all fundamental chip components. Hence, we simulate our designs firstly in GEMS+Simics environment to collect the performance traces and feed them to the input interface of McPAT. Once, McPAT gives the power values as outputs, we send them to HotSpot to get the thermal status of the chip. Accordingly, our designated algorithm will decide about the next cache configuration, that has to be applied at the underlying cache architecture, specifically at LLCs. The details for the developed closed loop simulation environment is described in Appendix A.

3.2 Benchmarks

As we discussed earlier, a full-system simulator like GEMS+Simics framework executes real workloads on the simulated architecture. With different set of results collected from these executions are analysed from performance perspective of the underlying architecture. Furthermore power and thermal simulation of the CMP architectures provide enough room for the hardware manufacturers or researchers to get testing for the new design that represent real-world behavior accurately.

The Princeton Application Repository for Shared-Memory Computers (PARSEC) [6], a benchmark suite that composed of multi-threaded applications which can be used for evaluation and development of next-generation CMPs. A collaborative project

between Intel and Princeton University drive their research community to put their effort for developing such benchmark programs those will help architects during designing future computer systems. PARSEC is freely available and well known in the architecture community. This benchmark suite is used for both academic and industrial research. PARSEC has the following major objectives:

- Focusing on multi-threaded applications.
- Diverse input sizes for each and every workload.
- Real-world problems are formulated.

Before PARSEC came into the picture, most of the benchmarks were application specific and they were available in an unparallelised version [6]. PARSEC 2.1 Benchmark Suite composed of 12 different workloads. Each of these workloads is multi-threaded and parallelised. The applications of PARSEC are basically chosen from different real-world areas, such as-finance, media processing, computer vision, enterprise service and animation physics etc. Table 3.1 contains the detailed descriptions regarding the PARSEC workloads. Multi-threaded applications usually share data among its spawned threads. Data usage details of the benchmarks are also provided in Table 3.2. The properties for each of the application are described in details in this article [6]. These workloads are also termed as programs, applications or benchmarks, alternatively.

Each of these application has its own WSS with a different set of input sizes: small, medium, large etc. According to the requirements, users can run applications with any input sets relevant to the architectural designs.

SPEC-CPU 2006 [15] is also another benchmark suite that is used in CMP simulations, but it is not available freely. The free availability of PARSEC benchmark suite is one of the major reason for which we have chosen it in our experiments. In last decade, SPLASH-2 [138] was a widely used benchmark suite for CMP environment. But for its smaller input sizes current large sized LLCs can not be verified properly.

Program/ Benchmarks	Application Domain	Parallelisation		Working-Set
		Model	Granularity	
blackscholes	Financial Analysis	data-parallel	coarse	small
bodytrack	Computer Vision	data-parallel	medium	medium
canneal	Engineering	unstructured	fine	unbounded
dedup	Enterprise Storage	pipeline	medium	unbounded
facesim	Animation	data-parallel	coarse	large
ferret	Similarity Search	pipeline	medium	unbounded
fluidanimate	Animation	data-parallel	fine	large
freqmine	Data Mining	data-parallel	medium	unbounded
streamcluster	Data Mining	data-parallel	medium	medium
swaptions	Financial Analysis	data-parallel	coarse	medium
vips	Media Processing	data-parallel	coarse	medium
x264	Media Processing	pipeline	coarse	medium

TABLE 3.1: The inherent key characteristics of Parsec benchmarks [6].

Program/ Benchmarks	Data Usage	
	Sharing	Exchange
blackscholes and swaptions	low	low
bodytrack and freqmine	high	medium
canneal, dedup, ferret and x264	high	high
facesim, fluidanimate, streamcluster and vips	low	medium

TABLE 3.2: The data usage behavior of Parsec benchmarks [6].

3.2.1 Parsec Benchmark Suite

In this section, the properties of some PARSEC benchmarks are described those are used in our works for evaluation of our different proposed CMP based architectures. The detailed descriptions regarding the benchmarks are presented in [6].

3.2.1.1 Blackscholes

This application models a financial analysis. It is basically an Intel's recognition, mining and synthesis (RMS) benchmark that analytically calculates the prices for a portfolio of European options with the Black-Scholes Partial Differential Equation (PDE). Blackscholes needs to solve a diverse variety of PDE for the application

in financial analysis. Its program is divided into a number of concurrent threads. Each thread represents a unit of the portfolio.

3.2.1.2 Bodytrack

This benchmark tracks the 3D view of the human body by using multiple cameras. An annealed practice filter tracks the 3D view by using an edge and foreground silhouette. In this benchmark, an input video that contains many frames is used to select as a reference frame. This reference frame at time stamp is used to compute its likelihood. This likelihood is a degree of the 3D body model alignment with its foreground and its edges in the respective images. The likelihood's value is computed by using the two attributes of a particular image, named as the foreground map and the edge distance map. This benchmark also has a persistent thread pool, from which the main thread sends the task to the thread pool. This main thread has to wait for the remaining working threads for finishing their executions before proceeding further.

3.2.1.3 Facesim

Facesim is an Intel RMS application that was developed by Stanford University. This is an animation based application which takes a human face along with a time sequence of muscle activations as inputs and computes a visually realistic animation of the modeled face. The underlying physics are simulated to get the visually realistic results. Basically, human faces are observed with more attention from the users than other details of a virtual world, so that a realistic presentation for animations can be prepared well.

3.2.1.4 Ferret

This process is used in content based similarity search of rich text data in large internet search engines. The rich text data includes audio, images, video, 3D

shapes etc. Ferret toolkit [139], used for searching, has six modules, out of which first and last modules are serially executed while the rests are parallel. The first and last modules are used as input and output, respectively.

3.2.1.5 Fluidanimate

The inclusion of this application in PARSEC is for increasing importance of real time animation and physical simulations for computer games. This is also an Intel RMS application that is based on Smoothed Particle Hydrodynamics (SPH) method [140]. Fluidanimate uses five kernels for simulating an incompressible fluid for the interactive animations. This also generates an output by interpreting and discovering the surface of incompressible fluid.

3.2.1.6 Freqmine

Freqmine is used for the Frequent Itemset Mining (FIMI) [141] with an array based version of the Frequent Pattern-growth method. This is also an Intel RMS benchmark that was actually developed by Concordia University. FIMI is the basis of Association Rule Mining (ARM), a common data mining problem for areas like protein sequences, market data and log analysis etc. PARSEC includes this because of its increasing demand in data mining techniques. This has been parallelised with OpenMP and uses three parallel kernels.

3.2.1.7 Swaptions

PARSEC has included this benchmark due to increase the importance of Partial Differential equation (PDE) and the Monte Carlo simulation. This is used for pricing a portfolio of Swaptions by using the Heath-Jarrow-Morton (HJM) [142] framework. This is also an Intel RMS workload like some earlier applications. HJM model behaves as a non Markovian that prevents the solving of PDE for the computation of prices. Hence, this benchmark employs a Monte Carlo simulation.

Basically, in Swaptions array, the program stores all the portfolio. Each of these array entry represents a derivative. Swaptions further divides the array into the number of blocks that is same with the number of threads that will be spawned and thus, each block is assigned to a particular thread. To compute a price Swaptions iterates through all of its blocks and earlier it calls the function HJM Swaption Blocking.

3.2.1.8 Vips

Vips includes fundamental image operations such as transformation and convolution. This application is based on the VASARI Image Processing System [143] which is able to produce multi-threaded image processing pipelines transparently at runtime. This image transformation pipeline has 18 stages and is implemented in the VIPS operation im benchmark. All of these 18 stages of Vips are implemented in the following kernels:

- Crop- removes 100 pixels from all the edges.
- Shrink- shrinks the image by 10% by using the matrix transformation.
- Adjust white point and shadows- brightens the white point and tries to reduce the shadows for improving the visual quality of an image.
- Sharpen- enlarges the edges of an output image. This kernel also removes the blurring and can produce better overall appearance of an output image.

3.2.1.9 X264

X264 is a H.264/AVC (Advanced Video Coding) video encoder that includes new features in encoding such as increased sample bit depth precision, higher-resolution colour information, variable block-size motion compensation (VBSMC) or context-adaptive binary arithmetic coding (CABAC). This application allows the H.264 encoders for generating a higher output quality in addition to a lower bit-rate

at the expense of a significantly increased encoding and decoding time. Motion compensation is used for removing the data redundancy. X264 is very flexible and is used for different requirements like video conferencing to HD movie distribution. Moreover, H.264/AVC encoding is also required for the next-generation HD DVD or Blu-ray video players.

3.3 Simulation Methodologies

For our experimental analysis, we used several multi-threaded and multi-programmed benchmarks. In the next section, we describe about the multi-threaded and multi-programmed benchmarks that we used/made from the PARSEC benchmarks.

3.3.1 Multi-threaded vs. Multi-programmed Benchmarks

Each PARSEC benchmark itself is a multi-threaded program, where number of threads in each benchmark is completely based upon the input size and the load of the program. Some benchmarks take number of threads as a command line parameter. For all PARSEC benchmarks, during their execution a specific region where the actual multi-threading occurs is termed as "Region Of Interest"(ROI). A magic instruction is inserted in each of the benchmark at the Simics console of the host machine just before starting of the ROI and also after completion of the ROI. This magic instruction pauses the benchmark execution at different points, particularly, it makes the entire simulator to pause its execution at the virtual machine. However, execution can be resumed by manually pressing a "c" or "continue" command. In brief, the real parallelisation of PARSEC benchmarks actually happens in the ROI. Initialisation, input scanning etc. are done before ROI, and on completion of ROI, program terminates after producing outputs.

We further built multi-programmed benchmarks by combining multiple PARSEC applications through Solaris Commands in virtual target machine. The term combining implies that, threads of different processes are bounded on different cores

and they will run on the same core until completion of the execution. For example, by using 4 copies of vips, we can make a multi-programmed benchmark in which each copy of vips can be bounded on 4 cores while considering a 16 core CMP. Note that, every single PARSEC application is a multi-threaded one, hence, in multi-programmed environment also each application has multiple threads where resources or codes are shared during execution. For multi-programmed benchmarks the term benchmark represents the combined workload.

3.3.2 Used Benchmark Applications

Based upon the multi-threaded benchmarks provided by the PARSEC we made different combinations for experiments. They are either a single PARSEC benchmark or a combination of more than one benchmark applications. Table 3.3 gives the details of the benchmarks we used in our experimental analysis. Also note that, all benchmarks mentioned in the table are not used for verification of all of our proposed or prior architectures.

3.3.3 Executing Benchmarks

For executing a multi-threaded benchmark each benchmark on the target machine is run upto the starting of the ROI. Once the ROI is reached the benchmark stops automatically because of the inserted magic instruction. Reaching ROI implies that the initialisation of the benchmark is over and all the threads are spawned. Once all threads are created, the benchmark is run further for 50 million cycles to warm-up. The warm-up period is necessary for avoiding the compulsory misses in caches and also allowing the NoC architecture to settle properly. After warming-up, the Ruby profiler is made clear through a command and from this point the actual execution begins. Few benchmarks are executed upto its termination i.e. until the ROI completion, whereas another set of benchmarks are executed upto a fixed number of cycles. Note that, the number of execution cycles varies across the benchmarks but is never less than 800 million (unless the case of termination).

Multi-threaded benchmarks	
blackscholes (*black*), bodytrack (*body*)	
ferret (*ferret*), fluidanimate (*fluid*), freqmine (*freq*)	
swaptions (*swap*), *vips* and X264 *(x264)*	
Multi-programmed benchmarks	
Benchmark	**Details**
black4	4 copies of *black*.
ferret4	4 copies of *ferret*.
fluid4	4 copies of *fluid*.
freq4	4 copies of *ferq*.
swap4	4 copies of *swap*.
vips4	4 copies of *vips*.
black16	16 copies of *black*.
body16	16 copies of *body*.
ferret16	16 copies of *ferret*.
fluid16	16 copies of *fluid*.
freq16	16 copies of *ferq*.
swap16	16 copies of *swap*.
vips16	16 copies of *vips*.

TABLE 3.3: List of all the multi-threaded and multi-programmed benchmarks used for the experiments in this thesis.

This number of cycles are termed as Simics cycles, which is 4× than the Ruby cycles of GEMS. Hence, 800 million Simics cycles implies 200 million Ruby cyles. The execution process that is fixed for a benchmark is being maintained for all the architectures being compared.

To execute a multi-programmed benchmark the very initial step is to load all the applications (belonging to the benchmark) one by one. Each application is then executed until its ROI. Once threads have been spawned, they are bounded at some cores. Note that, each program may have multiple number of threads and the threads can be bounded with the cores assigned to each application. However, after thread binding the execution of that application remains paused. This binding process is now repeated with the other applications. Once thread binding is done for all the applications, all of them are simultaneously resumed from their corresponding ROI. Rest of the processes, like warm-up and running policies are same like multi-threaded benchmarks. The thread binding for multi-programmed benchmarks are also done through the Solaris commands as we mentioned earlier.

3.3.4 Comparing Different CMP Architectures

For performance analysis of our proposed TCMP and CCMP architecture we compare its performance with other existing architectures in terms of IPC, energy consumption, EDP, running temperature, implementation overheads etc. In order to do so, we have designed all of our TCMP and CCMP architectures on GEMS+Simics (full-system simulator) and then we ran PARSEC benchmarks on top of them. We record different statistics during the execution of each benchmarks as discussed in Section 3.1.2.2. Based on this statistics, we compare the performances of two architectures.

Usually, an architecture is engineered with different configurations and design choices, for example, with various cache sizes, associativities, block sharing capabilities etc. Details regarding this will be provided in the relevant chapters/sections later whenever required. For all the architectures with different configurations, the process of running a particular benchmark is kept same to maintain uniformity. The individual results for each of the benchmarks are reported and the geometric mean (average) of all those are derived in our result sections.

3.3.5 Our Architectural Models

The entire thesis considers SNUCA based TCMP and CCMP as our baseline architecture which are already implemented in GEMS for different coherence protocols, cache sizes and replacement policies. The architectures are shown in Figure 1.2. MESI-CMP protocol is used as coherence protocol in our cache hierarchy. We use different cache sizes and associativities for the experimental versatility. Our first three contributions use TCMP as baseline whereas last one uses CCMP as its baseline architecture. The detailed configurations used in our simulations are provided separately whenever required in the subsequent chapters.

Mostly we perform our experiments by considering L2 as shared LLC having a size of 2MB, 4MB, 8MB and 16MB. Use of even larger caches may not be judgmental

according to the size of inputs provided by the PARSEC benchmarks. Multi-banked LLCs are sliced into equal sized banks. For example, in our baseline design an 8MB LLC (L2) has a bank of size 512KB, where we have 16 banks. The associativity is same across the LLC banks and is maintained uniformly.

Chapter 4

Static Energy Reduction by Performance Linked Dynamic Cache Resizing (DiCeR)

In this chapter, we are going to discuss about the dynamic tuning of LLC size, which is a promising option for reducing the cache leakage in modern CMPs. Towards this, our policy dynamically shuts down or turns on cache banks based upon the system performance and the banks' usage statistics. In addition with savings in leakage energy, shutting down of a cache bank remaps its future requests to another active bank, called as target bank. The proposed technique is evaluated on three different implementation policies.

4.1 Introduction

According to the survey given in [4], power consumption of on-chip memory sub-system shares a major portion of total power consumed by the chip. In modern CMPs, the on-chip caches are organised into multiple levels with the Last Level Cache (LLC) biggest in size. As LLC occupies large on-chip area, it consumes more leakage power that, at times exceeds dynamic power [144, 145].

As we discussed in Chapter 1 and 2, an effective way of reducing power consumption of the on-chip LLCs is by shrinking its size. Some recent works [85, 91, 75] have proposed power optimisation approaches where on-chip LLC has been shrunk. Reduction in LLC size can degrade system performance if the application's cache demand is more or if some heavily used cache portion is powered off. Hence, tuning process of on-chip LLC size should consider the system performance and locality of reference as its constraints. As these constraints are only known during execution of the applications, hence, dynamic/runtime cache size tuning approaches will be more effective for reducing LLC power consumption.

The recent surveys [5, 4] on cache-size tuning techniques from power/performance perspective have broadly classified the cache power reduction techniques into two categories:

1. Power supply control (State Preserving Approach), and
2. Resizing of the Cache Memory (State Destroying Approach).

The former one optimises the power consumption by controlling the power supply at physical circuitry whereas the latter one resizes the cache.

For modern tiled CMPs, recent works [91, 75] propose a set of utilisation based cache resizing techniques, which power off least utilised cache portions and dynamically remaps subsequent future requests to other active parts. In our work, we have taken a similar approach for optimising cache power consumption, by request remapping at L2-controller, unlike the prior works, where remapping is done at L1-controllers.

The current work proposes a dynamic cache tuning technique which considers performance and locality of reference as its constraints for managing the cache size. Towards this, we initially attempt to reduce cache-size by shutting down cache banks till an allowable degradation threshold in IPC which we refer to as BSP, the Basic bank-Shutdown Policy. In order to save leakage power, based on usage statistics, this policy turns-off L2 cache banks at runtime and its future

accesses are remapped to other L2 cache banks, called as target banks. Once, the performance degrades beyond a predefined threshold, the system stops bank shutdown process. However, this policy cannot provide adequate cache space to the process in case it needs more cache space in future, during execution. To address this issue, we developed an extended policy of BSP, which takes care the sudden increment in application's Working Set Size (WSS) during execution by allowing dynamic restarting of the powered off cache banks. System performance is monitored periodically and accordingly L2 bank(s) will be restarted if performance degradation is more than a threshold value. During turning on process, all the remapped contents are brought back to this bank from its remapped location. The results are compared with *Drowsy* [1], an existing policy. Specifically, the main contributions of this work can be listed as follows:

1. **B_ON_OFF_ALL** A performance linked dynamic cache tuning strategy resizes the L2 caches by turning off cache banks. However, if the application needs more cache space, L2 banks are turned-on/restarted.

2. **B_OFF_ONCE** Frequent turning on and off of the L2 banks degrades performance. Hence, we experiment with different on-off patterns. Once the performance degradation reaches a threshold value, the system will not allow any more shutdown of L2 bank(s). After this only turning on of L2 cache banks will be allowed.

3. **B_ON_OFF_OPT** The frequent resizing of L2 cache of first policy may degrade system performance. On the other hand, second policy does not allow the system to save power by turning-off the cache banks once the turn-off process is stopped, even when there is scope to do so in future. These two problems have been rectified in the third policy by putting some restrictions on cache resizing.

Basically, in this work, power saving is done by complete shutdown of the underutilised cache banks. Shutting down of these least utilised cache banks can also aggravate the system performance, if the current application later changes its

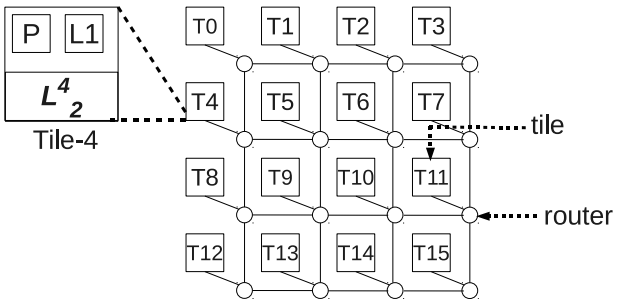


FIGURE 4.1: Tiled CMP architecture

cache-space requirement. In such situations, the powered off banks will also be turned-on when performance degradation is more than a preset threshold value. The baseline architecture used in this work is elaborated in Figure 4.1. According to this figure, the whole chip is a collection of some replicated tiles, where each tile contains a processor core along with its private L1 (Data & Instruction) caches and a chunk of shared L2 cache, called as L2 bank. In this figure tile 4 is elaborated in details. Note that, L2 cache is used here as on-chip LLC and it is physically distributed uniformly among the tiles. The tiles are connected to each other through a 2D NoC and hence, each tile is also attached to an NoC router.

4.2 Proposed Energy Saving Policy

According to our analysis given in Figure 4.2, it can be stated that most of the cache banks are underutilised while executing processes. This characteristic motivates us to tune the cache size dynamically by selectively powering off or turning on the cache banks as it is required. However, the cache space requirements of the applications are diverse to each other i.e. each of them has separate cache space requirements and even the cache banks in which data is accessed frequently are not fixed throughout the execution. Therefore, cache bank usages can only be known at run-time as to which bank(s) is(are) the mostly accessed while which are utilised the least. As all the cache banks are neither fully nor evenly utilised as shown in Figure 4.2, it would be helpful to shutdown selective cache banks, specifically, the least used ones. The load of these powered-off banks will be shared by the remaining set of active banks. Bank shutdown will save static energy but

effectively reduce the cache space. This might lead to more misses and hence affect the performance. Our policy therefore needs to keep track of performance.

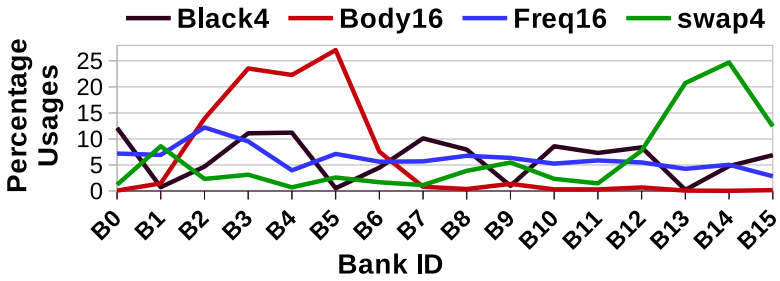
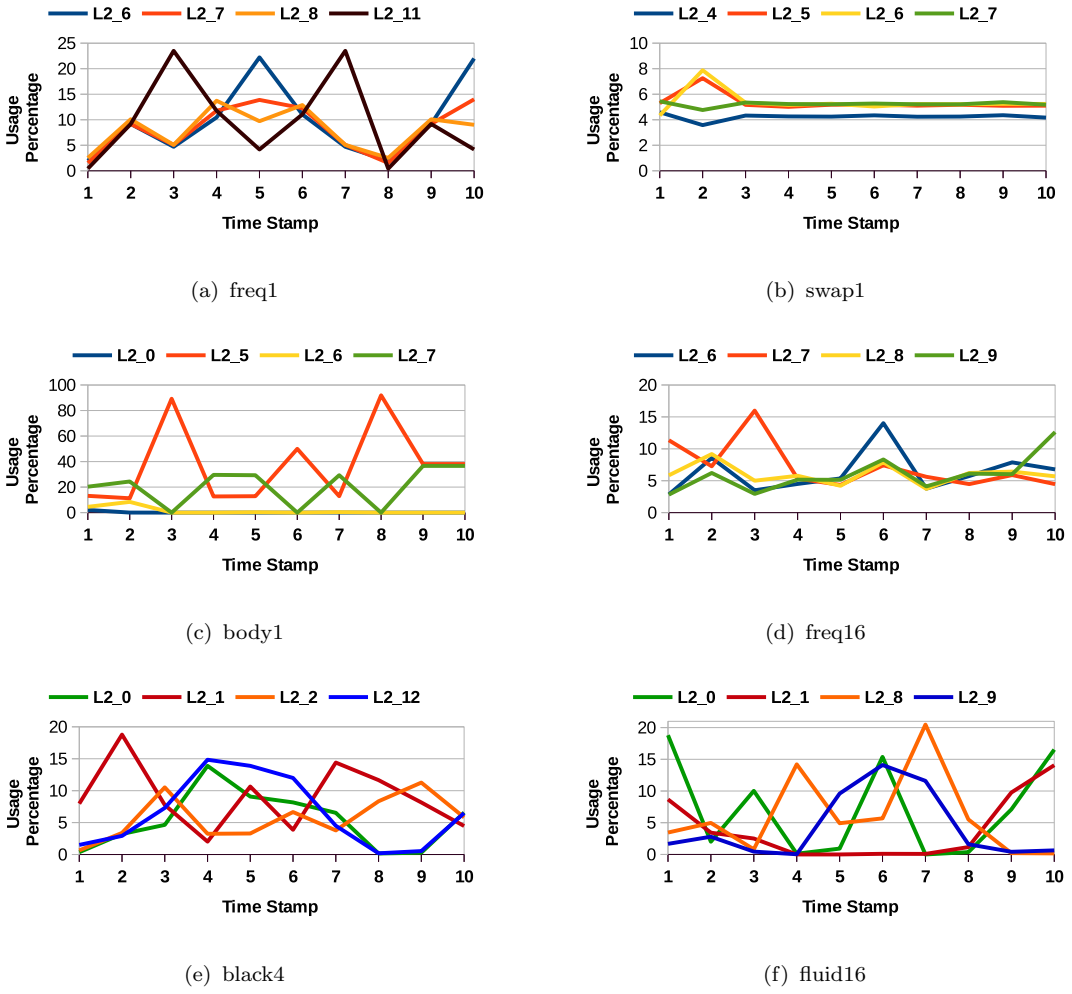


FIGURE 4.2: Variable bank usages across different benchmarks

At some certain point during execution, the *Usage Percentage* of a bank in Figure 4.2 implies what percentage of total L2 accesses (i.e. $\#hits + \#misses$) this bank handles, upto the current time-stamp. Values in Figure 4.2 have been collected for 4 benchmarks by running them upto 40 million clock cycles, after warming up. Moreover, the bank usages for applications even vary with respect to time, i.e. a heavily used bank of initial stage of execution may become a lightly used one in latter stage, or vice versa. For some applications (swap1, body1, ferret1, vips1 etc.), a few banks have almost same amount of usages throughout the execution, as they exploit high *Locality of Reference*. Graphs in Figure 4.3 show the patterns of these temporal changes of bank usages for a set of applications. For each application, we show randomly selected 4 banks out of 16 where each X-axis in the Figure 4.3 represents uniform time-intervals. The bank usages for the applications change over time, as shown in Figure 4.3, hence online stats are considered for dynamic resizing of LLC. Note that, bank usage statistics are taken over a moderate periodic interval for which changes in usages are not significant, which implies locality of reference is exploited within short time-duration, but for longer duration, this property is prone enough to be violated.

On the other hand, energy analysis for on chip caches shows that static energy consumption is lot more than dynamic energy consumption. Dynamic energy of the cache is only consumed during cache block accesses, whereas static energy is continuously spent as long as power is supplied to the bank. Figure 4.4 shows the distribution of static versus dynamic energy for a few benchmarks.



(a) freq1 (b) swap1

(c) body1 (d) freq16

(e) black4 (f) fluid16

FIGURE 4.3: Temporal Change in Bank Usages for 6 different PARSEC applications.

Hence, in order to save energy, static energy saving will be the most useful component, which can be achieved by powering-off some least used cache portions. The dynamic performance tracking helps to decide about the cache size tuning, whereas the dynamic bank-usage profiles are used to determine which banks are to be turned-off or on.

During the process execution, it is noticed that, cache requirement for a process can be changed at any point of time. Hence, a one time shutdown of cache banks (like our BSP) may degrade system performance (and in a few cases it may be costly enough). To compensate this situation, the banks are selectively turned on as per system's requirement. This approach for energy saving is taken with system wide performance constraint.

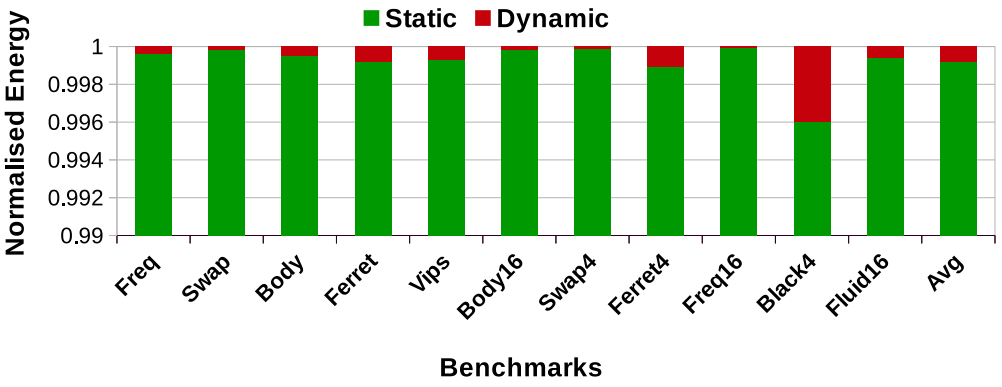


FIGURE 4.4: Distribution of static and dynamic energy consumption of on-chip Last Level Caches(LLCs). Benchmarks are put in X-axis. Suffix '-B' implies that, simulation results are obtained in baseline architecture.

ALGORITHM 1: Performance linked dynamic cache tuning

T : Reconfiguration interval
δ : Permissible percentage degradation in IPC (Instruction Per Cycle)
m : Maximum limit on bank shutdown
$j = 0$: Number of banks turned off. Initially zero.
while application is running **do**
    Run the application for T number of clock cycles.
    Compute degradation in IPC compared to the IPC upto last reconfiguration interval.
    if degradation is lesser than or equal to δ and $j < m$ **then**
        CALL **TURN-OFF** Algorithm (i.e. Algorithm **2)**
        j:=j+1
    end if
    if degradation is greater than δ and $j > 0$ **then**
        turn on the cache bank which was turned off most recently.
        CALL **TURN-ON** Algorithm (i.e. Algorithm **3)**
        j:=j-1
    end if
end while

The L2 bank with the least number of accesses over the last reconfiguration period of the execution will be selected as candidate for shutdown. Our hypothesis is that, according to temporal locality the bank having lesser accesses in the current interval will have fewer accesses over the next (few) interval(s). Therefore powering off this bank may not affect system performance and at the same time will save static power. However, performance degradation needs to be monitored, as at any point of time during process execution resource requirement may change.

ALGORITHM 2: : TURN-OFF *(Manage cache bank ShutDown)*

1: Calculate the usage for every active cache bank.
2: Select the cache bank, B_i, with minimum usage, as a candidate to shutdown.
3: Select another cache bank, B_j, with usage greater than that of B_i as the target bank.
4: Stall all the requests for B_i, however keep the response queues open.
5: Migrate all the valid blocks from B_i to B_j.
6: Inform the controller of B_i to forward all subsequent requests to B_j.
7: Open the request queues for B_i.
8: Shutdown bank B_i.

So, if the performance goes below a given threshold, bank shutdown will not be allowed. In addition, if the number of currently turned off banks is more than zero, then the most recently turned off bank will be turned on. The performance degradation is analysed periodically and depending upon this, bank shutdown or turn on processes continue. This whole cache tuning process is repeated all over the program execution. The shutdown process will be stopped if at any time, the number of turned off banks reaches the maximum permissible limit. This maximum limit can be set by performing experimental analysis [5]. The complete method is given in Algorithm 1, 2 and 3. Note that, either the Operating System (OS) or the master cache (i.e. LLC) controller are responsible to keep track of the cache bank usages along with other technical concerns while implementing our policy.

ALGORITHM 3: : TURN-ON *(Manage cache bank Turn On)*

1: Select the cache bank, say B_i, turned off most recently as a candidate for turning on.
2: Identify the target bank, say B_j, to which its requests have been remapped.
3: Stop the remapping of requests from B_i to B_j.
4: From B_j, migrate all the valid blocks to B_i, whose original location was B_i and were remapped to B_j.
5: Turn on B_i and open the requests for this bank.

4.2.1 Book Keeping and Future Requests

Before turning off a cache bank, two major issues need to be handled: (i) relocation of existing valid blocks in the bank, and (ii) future requests which will come to this bank after turning off. All future requests to this shutdown bank should be redirected to some other bank, called '**target**' bank. The '**target**' bank is chosen based on the usage statistics, which implies that the banks, those are prospective candidates to be turned off in future, will not be selected as '**target**'. Handling of already cached blocks have two options: (a) Writeback all these blocks to the next lower level memory (W), or (b) send/migrate these blocks to the target bank (M). Hypothetically, writing back to next level memory may be an expensive operation, especially (it will be much more expensive in terms of latency) when the next level memory will be off-chip. In this regard, we also performed a set of experiments on PARSEC benchmark applications, which claim the correctness of our hypothesis.

To show the effectiveness of both W and M policies, we implement them in our simulation framework for a TCMP architecture having a 4MB 4 way L2 cache as on-chip LLC. For in depth analysis, our BSP has been modified in this experiment, where, we perform simulations on 4 different configurations:

- BSP_H_M : Turn off heavily used cache banks and migrate blocks from victim to target during shutdown operation.
- BSP_H_W : Turn off heavily used cache banks and write back blocks from victim to the lower level memory during shutdown operation.
- BSP_C_M : Turn off lightly used cache banks and migrate blocks from victim to target during shutdown operation.
- BSP_C_W : Turn off lightly used cache banks and write back blocks from victim to the lower level memory during shutdown operation.

Figure 4.5 shows that, the IPC degradation is more for write-back than migration for all of our applications, as the write-back policy takes more time to settle down

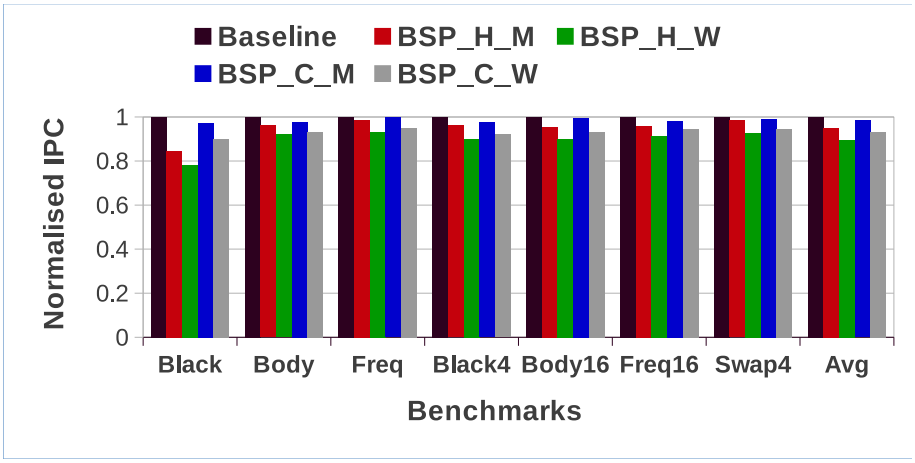


FIGURE 4.5: Comparison between Migration (BSP_H_M, BSP_C_M) and Write-Back (BSP_H_W, BSP_C_W) policies in terms of IPC, while turning off some heavily used and lightly used L2 banks.

than the migration based policy. For write-back cases, average IPC degradation is near to 10%, whereas in migration based policy average IPC degradation is less than 5%. These results have strengthened our hypothesis, and motivate us to use the latter option i.e. migrate the victims' blocks to the '**target**' while resizing the cache.

However, during relocation of these blocks, the bank being shutdown (victim bank) is searched line-by-line to evict its blocks one-by-one for sending them to its target. While block migration initiates, it is only the victim bank who does not handle any further requests. But other banks in the system continue as usual with normal operations. In prior works, the complete system was stalled during the relocation. But in our proposal, only the bank being shutdown (victim bank) is stalled during the shutdown process. Once the block migration (from victim bank to target bank) is completed, the (victim) bank will be shutdown. For turned off banks, future requests are forwarded to the target bank. Dynamic energy consumption for the target bank increases marginally but the static energy of the turned off bank, which is a significant portion, is saved. This whole process will increase the traffic in NoC which will incur a number of stall cycles and increase in NoC power consumptions. Furthermore, cache bank on-off overhead will also incur stall cycles in the system. Our simulated system takes into account all of these system

overheads. The average reconfiguration overhead incurred in our experimental evaluation is reported in Section 4.4.1.

During shutdown process, the migrated blocks are loaded into the target bank from the victim bank. In case no free ways are available in the corresponding set of the target bank, ideally the oldest block among the incoming migrated block and the LRU (Least Recently Used) of that particular set should be evicted. But, an LRU block can only be decided inside the bank. Hence, in our implementation we add an extra field called *Reuse-Counter* of 4 bits to each block like [18], having a negligible storage overhead. This value is also sent with each block during migration. The values of *Reuse Counters* of the incoming migrated block and the LRU block of the target's set are compared, and lower valued one will be eventually evicted. However, conflict cases are handled by evicting the migrated blocks.

Similarly, during turning on process, the *target bank* of the bank being turned on will be stalled and will not be handling any requests during the turning on process. The remapped data of this bank, which is being turned on, will be brought back from its target bank. The bank will be turned on after completion of this migration process and all the pending requests will be processed after that. On completion of turning on process, the bank starts handling its own requests normally. The whole process is transparent to the underlying cache coherence protocol. The request-redirection happens at victims and the states of redirected blocks are maintained by the target banks.

Storage Overhead for Source Bank Tracking Selection of the target bank is decided at runtime based on bank usage statistics of the cache banks. Hence, no separate remap table is required to be maintained. But whenever a cache bank is needed to be turned on all of its remapped cache blocks need to be placed back in it from its target bank. As a bank can become target bank for more than one banks, an extra field, called *Source_ID* is added with every block in the LLC to keep track of the original cache bank ID for remapped cases. To store source cache bank ID, size of this field will be $\log_2 N$, where N is the number of cache banks.

For a 4MB L2 cache having 16 cache banks of equal size with 64 Bytes data block, this additional field will have an overhead of 32 KB extra storage i.e. 0.78%, which is negligible.

Prior works, in which power saving has been done through remapping techniques, have used a remap table at the L1-cache level and whenever any new bank is shutdown the entries in all L1 caches need to be updated. Also keeping and maintaining such tables with L1 caches is an additional overhead. Our proposed method is completely transparent to the L1 caches. The L2 controllers of the banks being shutdown maintain the information of the target bank. For every shutdown bank, there is only one target bank, so, there is no hardware overhead of remap tables.

4.2.2 Constraints to maintain

As it is mentioned earlier that, the requests to the powered off banks are forwarded to their respective target banks after the shutdown process. This redirected/forwarded requests increase some amount of network utilisation and so the overall network power consumption also goes up. However, according to the experimental evaluation this overhead is not significant. The main metric to consider here is the system performance, i.e IPC which is dynamically collected by the controller. The system wide IPC, i.e. average among all the cores, has been tracked during the execution and if it drops below a given threshold, no more shutting down of cache banks will be permitted. Note that, there is a possibility to turn on a powered off bank, if any exists. Mainly, this happens when the working set of the application needs more cache space and shutting down of one full bank causes overload on the target bank. For a set of benchmark applications this happens when the number of powered off banks is greater than half of the total number of cache banks and the system yet shuts down one more bank. Here, the cache misses increases in the target bank, that leads to degradation of IPC. Therefore, the proposed method keeps track of IPC degradation and thus makes a performance aware static energy savings by selectively tuning the cache size.

System wide IPC depends upon two performance parameters: (a) Computation Time and (b) Memory Access Time. The memory access time also depends on the available cache capacity. An analysis of this dependency is employed in Chapter 5.

Config.	**freq1**	**swap1**	**body1**	**ferret1**	**vips1**	**body16**	**swap4**	**ferret4**	**freq16**	**black4**	**fluid16**
Baseline	0.314	0.328	1.92	2.401	1.05	0.94	2.09	0.195	0.83	0.294	2.16
Bank-off	1.32	0.692	2.19	3.20	2.19	2.18	2.7	0.345	1.47	1.17	2.7

TABLE 4.1: Maximum percentage of runtime IPC degradation with 4MB 4Way L2 cache.

However, while running an application, both computational as well as memory cycles may change at any point of time and can increase or decrease IPC according to the program's structure. To see the effect of bank shutdown on IPC, we compare baseline's IPC with the IPC of pure bank shutdown policy (where number of shutdown banks has been limited to 50% of total number of banks), through a set of simulations. Table 4.1 shows the values for these setup, which demonstrates IPC degradation because of bank shutdown. Hence, bank turn-on is further required to restrict this IPC degradation within a limit.

All the usage statistics and shutdown management can be handled by the cache controllers. To keep its logic simpler, we are not allowing shutdown of the target banks. As shutting down of a target bank will incur transitive redirection of cache blocks which will affect the system performance and will increase the network traffic. During the execution, a few clock cycles are required to transfer data between banks and for collecting usage statistics. These extra system overheads are taken into account in our simulations. Moreover, for data redirection, NoC energy consumption is increased which are calculated implicitly in (Garnet-Orion integrated) network model of GEMS simulator.

4.3 Experimental Evaluation

In our experiments, we have used a 16 core TCMP architecture, where each tile contains a processor core with a private L1 cache, and a slice of shared L2 cache,

	4**MB** 4**Way**	8**MB** 4**Way**
Leakage Power per bank	249.851 mW	322.359 mW
Dynamic energy per access	0.188211 nJ	0.232409 nJ

TABLE 4.2: Energy/Power values obtained from CACTI [7] for two different L2 cache configurations. In each of these cases, L2 is uniformly divided into 16 banks.

Components	**Parameters**
No. of Tiles	16
Processor	UltraSPARCIII+
L1 I/D Cache	64KB, 4-way
L2 Cache bank	256 KB or 512 KB, 4-way
Memory bank	1GB, 4KB/page
Flit Size	16 bytes
Buffer Size	4
Pipeline Stage	5-stage
VCs per Virtual Network	4
Number of Virtual Networks	5

TABLE 4.3: System and Network Parameters

Cache Parameters	**Values**
Cache Level	L2
Size of a L2 Bank	256 KB or 512 KB
Block Size	64 Bytes
Technology used	32nm
Associativity	4
Cache Model	NUCA
Operating Temperature	380 K
Actual Cache Size	4 MB or 8 MB

TABLE 4.4: CACTI Configurations

called L2 cache bank. These set of tiles are interconnected with each other through a 2D network mesh. Each tile contains a router in it for communicating among the tiles. The detailed architectural diagram is shown in Figure 4.1.

4.3.1 Experimental Setup

A multicore time-based simulation framework of GEMS + Simics are used here for our experiment, which have been described in Chapter 3. For our memory module simulation we use Ruby from GEMS. The cache coherence is maintained here by

MESI-CMP based cache controller. The configuration details for this simulation setup is given in Table 4.3 which contains details about processor, memory and network configurations. Performance and power simulation of NoC is handled by the Garnet-Orion integration that have been clubbed inside GEMS. The values of the cache parameters used in CACTI simulation are given in Table 4.4. For calculating cache power consumptions, CACTI 6.5 has been used. Table 4.2 gives all the energy/power values obtained for our experimental setup.

PARSEC benchmark suite [6] has been used as application program for validating our proposed architecture, details of which are also given in Chapter 3.

4.4 Results and Analysis

Before evaluating the proposed policy, we initially implement BSP in our simulation framework in which L2 banks are turned off dynamically until the IPC degradation is within a predefined limit. We have taken a 4MB 4Way set associative L2 cache for our experimentation with a 3% performance degradation limit and maximum 8 out of 16 banks (i.e. 50% at max.) can be turned off. The whole execution span is divided into several uniform time-slices, called as reconfiguration period, at the end of which, system takes bank shutdown decision, based upon the performance statistics. To keep performance degradation within a limit, BSP does not allow further cache bank shutdown, once this threshold is violated. But, if the WSS of the running application(s) need(s) more cache space in future, BSP cannot provide the same by turning-on some cache banks. As a mitigation to this fact, we have decided to turn-on some banks on violation of performance degradation threshold.

The proposal suggests cache bank shutdown to save power and turn-on to rectify IPC degradation. The experiments are performed on our main three different policy-configurations, as discussed below:

1. **B_ON_OFF_ALL** The first policy-configuration allows the banks on/off procedure to continue throughout the process execution with a reconfiguration interval of 2M clock-cycles. In other words, after every 2M clock-cycles, based upon the system's performance, decision is taken to either shutdown, or turn-on or no-change. This may lead to constant fluctuations, in that some banks will be turned-off and within few intervals, again they will be turned-on. This may degrade system performance. The next policy tries to rectify this.

2. **B_OFF_ONCE** When the application starts executing, depending on the system performance reduction of cache size is invoked by bank shutdown. This process stops once the IPC degradation reaches a threshold. Henceforth, only bank turn-on is permitted to improve the IPC.

3. **B_ON_OFF_OPT** Cache requirements change over the process execution. The second policy only allows shutdown followed by restart of cache banks. However, the cases when cache requirement may reduce later cannot be handled here. In order to give more flexibility, in this policy the reconfiguration of banks is done periodically. In particular, cache tuning is done for 25M clock cycles and no changes take place for the next 25M clock cycles. Cache is again resized for the next 25M clock cycles and so on.

For each of these policies, the system is evaluated for three different IPC degradation threshold values: 2%, 3%, and 4%. The results are compared with BSP and two different configurations of Drowsy [1] for a 4MB cache with IPC degradation threshold of 3%. Later, results are presented for the trade-off of varying IPC degradation threshold on the power savings. We have also experimented with larger cache of size 8MB. The results for the same is also shown at the end of this section.

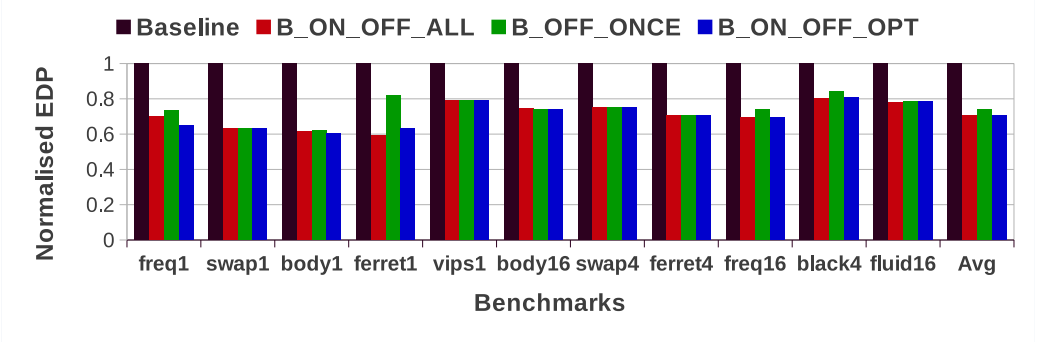


FIGURE 4.6: Energy Delay Product obtained in proposed policies over the baseline, BSP and Drowsy, with 4MB L2 cache. A smaller value is better.

4.4.1 Comparison with baseline architecture

Energy Delay Product (EDP)

Figure 4.6 shows the EDP savings (on Y-axis) for all the policies normalised over the baseline for the different benchmark programs, shown along the X-axis. In this case, energy dissipated for following components, such as-L2 cache, network and DRAM accesses are included. Energy consumed by other on-chip components like L1 cache, CPU cores etc. are not included in calculation of EDP or total energy consumption in this work. Applications having WSS much smaller than the available cache show significant improvement in the savings for B_ON_OFF_ALL and B_ON_OFF_OPT. But, for B_OFF_ONCE a few applications show less savings. These applications need more cache space initially, so lesser number of cache banks have been shutdown. As in B_OFF_ONCE, no cache bank shutdown will be allowed once IPC degradation is more than the given threshold. We get an average savings of 29%, 27% and 30% in EDP for the three policies over our baseline architecture.

Static energy savings

Static energy is the main component of the chip energy which we save here by tuning the cache size dynamically. Figure 4.7 shows the energy consumptions for different policies over the baseline architecture. We saved 66%, 59% and 65% in static energy consumption on an average for different policies over the baseline. Shutting down of cache banks leads to no switching activity in those areas of the

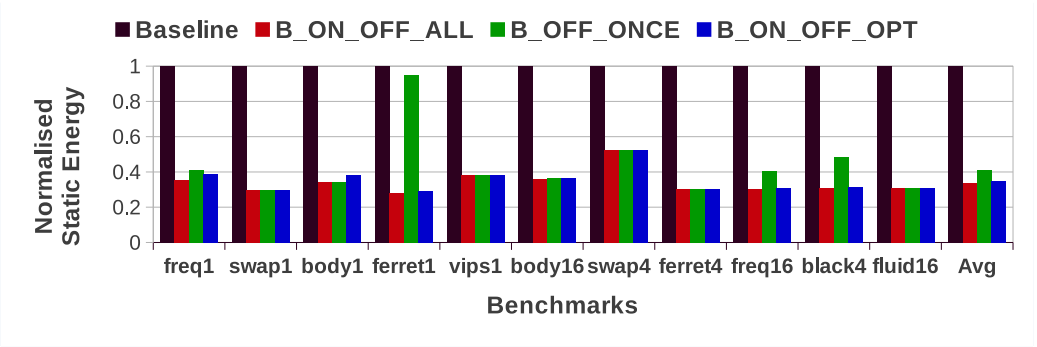


FIGURE 4.7: Normalised static energy consumption, with 4MB L2 cache.

chip which in-turn helps to control the temperature increase in these parts of the chip. Note that, this remap policy can be used to relocate cache requests from hotter part of a chip to cooler ones.

Execution Time

Dynamic cache tuning incurs data migration of valid blocks in the bank being shutdown and also requires forwarding of subsequent requests to the target bank. Implication is that, the network traffic will be increased and also leads to slightly more cache misses at the target bank. These overheads lead to degradation in overall IPC. The average overhead incurred from migration is around 1.3% of the total execution time, which is negligible. Figure 4.8 shows the normalised IPC across the benchmarks for all the proposed policies over baseline. The respective IPC degradation for B_ON_OFF_ALL, B_OFF_ONCE and B_ON_OFF_OPT are 2.9%, 1.5% and 1.8% on average. The excessive cache tuning in B_ON_OFF_ALL increases more number of idle clock cycles in the system and hence the performance degrades. But the controlled cache tuning of B_ON_OFF_OPT reduces the performance degradation significantly than B_ON_OFF_ALL. For B_OFF_ONCE, the cache size will be changed in a very restricted way, so performance degradation is lesser than other two policies, but this policy does not offer more energy savings compared to B_ON_OFF_OPT. However, B_ON_OFF_ALL enables better control on cache size and hence we get more EDP savings than B_OFF_ONCE.

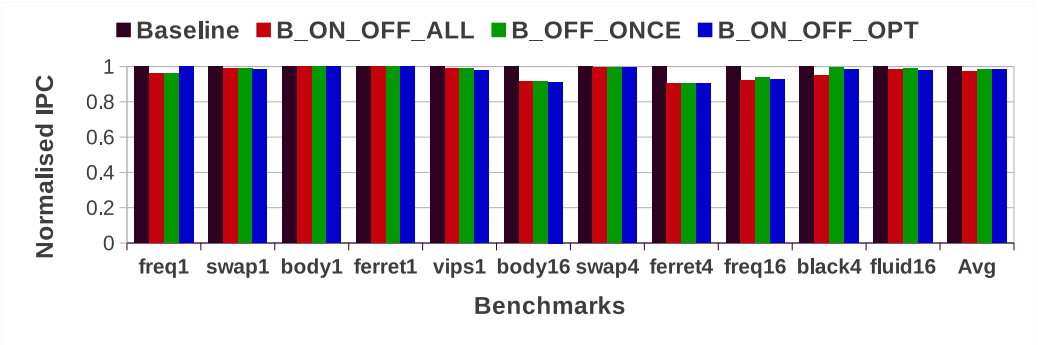


FIGURE 4.8: Normalised IPC value for different benchmark applications, with 4MB L2 cache.

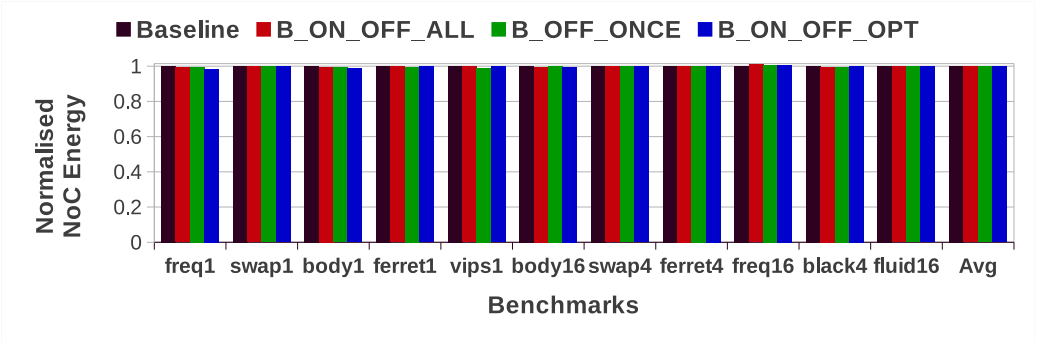


FIGURE 4.9: Normalised network energy value for different benchmark applications, with 4MB L2 cache.

Network overhead

While shutting down the cache banks, its contents must be transferred to the target bank and all future requests must be forwarded to the target bank. Even while turning on a cache bank, all of its existing contents from the target bank must be brought back into this bank before resuming its normal operations. This increases the network traffic as is evident from the increased energy consumption. But the cache portions which are turned off or turned on are the least accessed portions, hence the amount of traffic transferred is not huge with respect to normal baseline communication. However, frequent cache tuning increases the network traffic. The result shown in Figure 4.9 implies that network energy has not been increased significantly. However, this small increment of around 1.23% is compensated by the reduction in static power, which is evident due to an overall 30% savings in EDP.

Additionally, Figure 4.10 shows the change in NoC latency for all the applications.

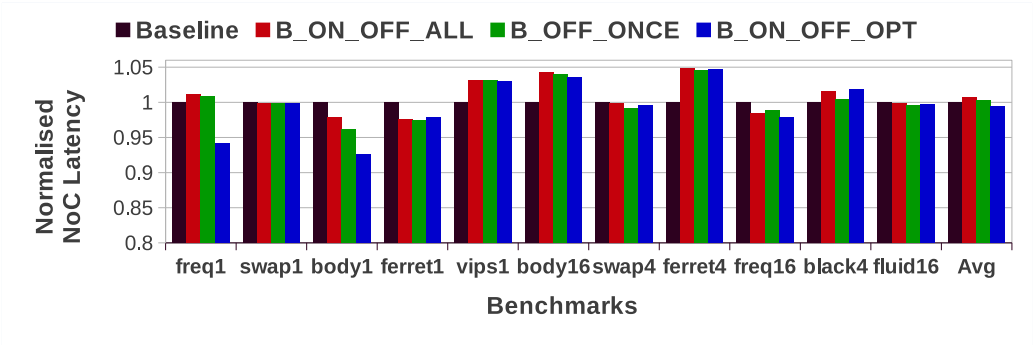


FIGURE 4.10: Normalised NoC latency for different benchmark applications, with 4MB L2 cache.

Most of the cases, the increment in latency is not significant, however, highest increment is 4.7% for ferret4. On the other hand, for freq1, body1 and ferret1 latencies are decreased in some cases. The reduced NoC latencies in Figure 4.10 reflects the performance improvements in Figure 4.8. This decrement in latency happens due to lesser amount of remapped requests after shutting down of banks, i.e. number of requests to the turned off banks have been reduced during execution. The other reason for reduced NoC latency could be due to the target banks being in close proximity of the accessing core compared to the original (victim) bank. Note that, overall bank access patterns change with the policies, hence, there exists diversity in NoC latency for the same application.

Effect in Total Energy Consumption

Savings in static energy reduces the total energy consumption effectively. Figure 4.11 shows reduction in total energy consumption with significant savings in static energy than baseline architecture. The proposed policy B_ON_OFF_OPT, is compared in this figure with baseline architecture (_B and _O are suffixed with the benchmarks' names in this figure to represent Baseline and B_ON_OFF_OPT policy). Slight increment in network energy, due to remapping, is compensated by significant EDP gains due to performance aware static energy savings. Dynamic energy (energy for cache accesses) is added with the network energy in this figure.

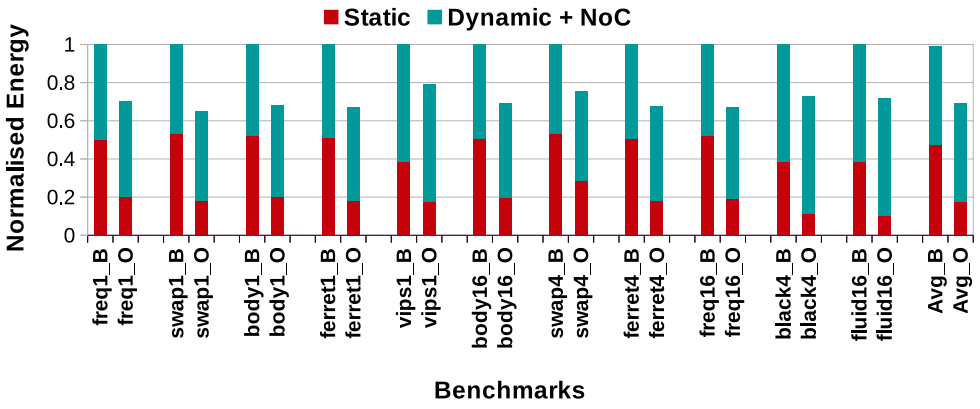


FIGURE 4.11: Normalised total energy consumption with details breakdown of its components, for different benchmark applications, with 4MB L2 cache.

4.4.2 Comparison with BSP and Drowsy [1]

According to the experimental results shown earlier, B_ON_OFF_OPT policy gives the best energy savings among the three proposals. We compare energy savings of B_ON_OFF_OPT with BSP and Drowsy here. As it is discussed earlier, Drowsy cache actually puts some cache portions in drowsy or low power consumption mode; hence, more amount of drowsy cache portion saves more energy, but, may degrade system performance by incurring more idle clock cycles while accessing data from drowsy parts. To address this issue, authors proposed a set of combinations in [1], by which amount of drowsy portion can be decided depending upon the cache size. In our experiment, we use a 4MB 4 way set associative L2 cache, for which two possible drowsy cache combinations have been taken for comparison. The drowsy policies are as follows:

- Drowsy_C1 : has the ratio 1 : 3, with 3 ways in drowsy mode and 1 way in normal mode.
- Drowsy_C2 : has the ratio 2 : 2, with 2 ways in drowsy mode and 2 ways in normal mode.

Figure 4.12 and Figure 4.13 show the EDP savings and static energy savings, respectively, for B_ON_OFF_OPT over Drowsy and BSP, normalised with respect

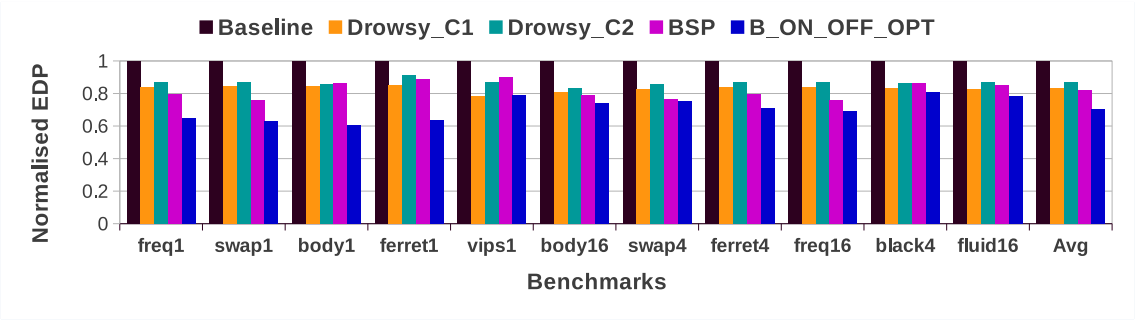


FIGURE 4.12: Energy Delay Product obtained in B_ON_OFF_OPT over BSP and Drowsy, with 4MB L2 cache. A smaller value is better.

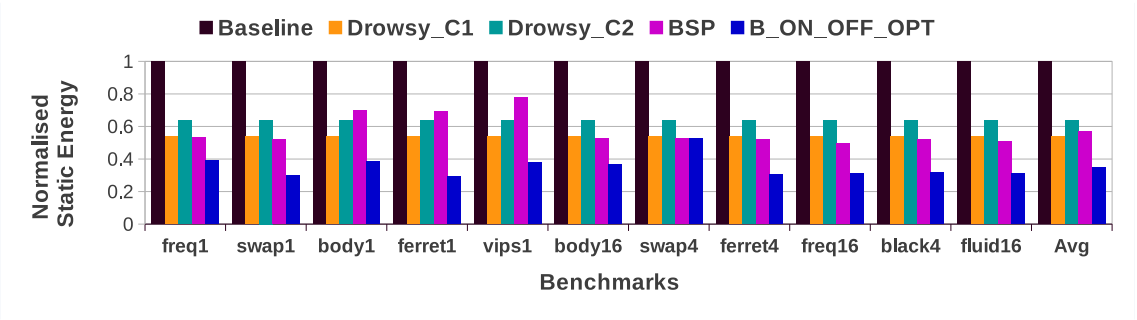


FIGURE 4.13: Normalised static energy consumption for B_ON_OFF_OPT and compared with BSP and Drowsy, with 4MB L2 cache.

to baseline. Average EDP gains for BSP is 20%, and average static energy saving is 44%. B_ON_OFF_OPT, the proposed policy, has 30% gains in EDP and 65% savings in static energy with respect to baseline. Drowsy_C1 saves 17% EDP with 46% savings in static energy, where Drowsy_C2 has EDP gains of 13% with 36% savings in static energy. As drowsy cache incurs extra idle clock cycles for accessing the data from the drowsy parts of the cache, this degrades the IPC in the range of $1.2 - 1.8\%$, on an average. In case of BSP, IPC degradation is more than Drowsy which is 2.5%. However, our proposed policy B_ON_OFF_OPT degrades IPC by 1.8%, on an average, is comparable to Drowsy. With respect to baseline, B_ON_OFF_OPT has 19% and 29% more static energy savings than Drowsy_C1 and Drowsy_C2, respectively.

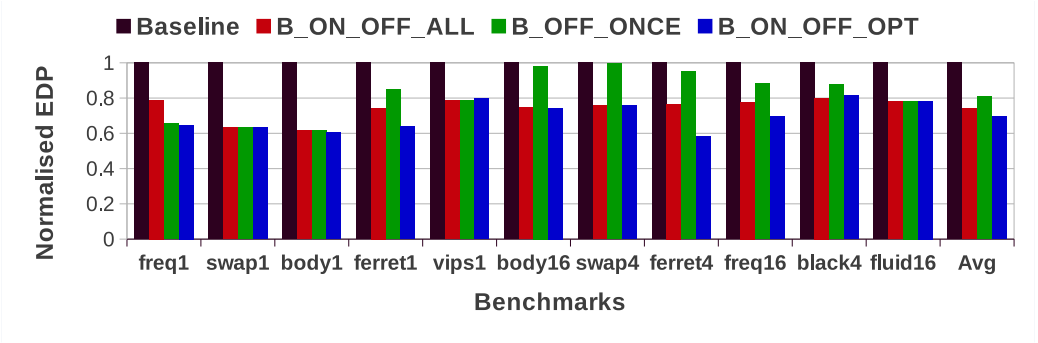


FIGURE 4.14: Normalised EDP value for different benchmark applications for IPC degradation threshold = 2, with 4MB L2 cache.

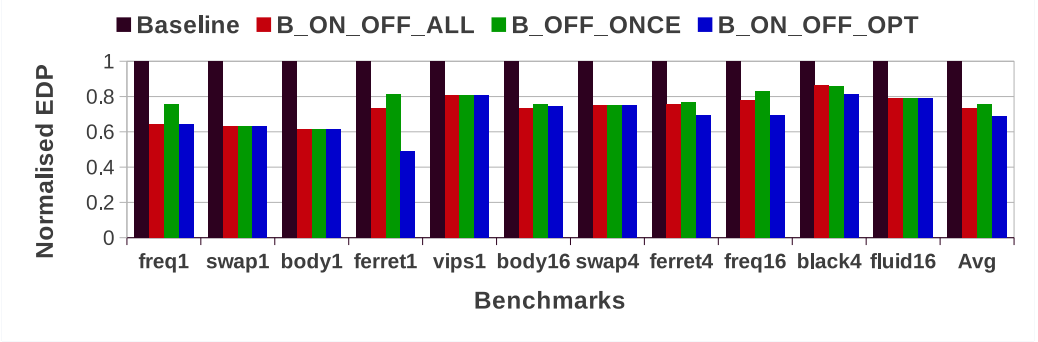


FIGURE 4.15: Normalised EDP value for different benchmark applications for IPC degradation threshold = 4, with 4MB L2 cache.

4.4.3 Analysis of power savings by varying the IPC constraint

The previous section analysed the impact of the policies on IPC and power savings. We further analyse the effect of policies i.e. B_ON_OFF_ALL, B_OFF_ONCE and B_ON_OFF_OPT on power savings by relaxing and strengthening the IPC constraint. In particular, we present the results for IPC degradation threshold values of 2% and 4%.

Energy Delay Product

Figures 4.14 and 4.15 show the EDP gains for IPC degradation thresholds of 2% and 4%, respectively. The graph for 3% threshold is given in Figure 4.6. In particular for 2%, the savings are 26%, 19% and 29% for B_ON_OFF_ALL,

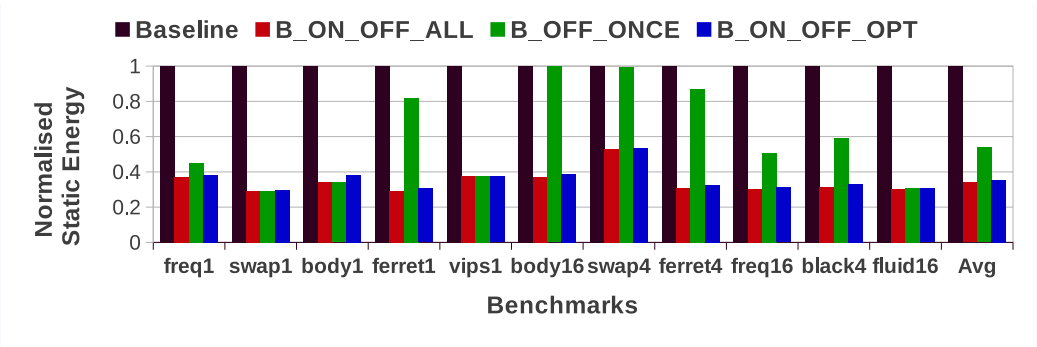


FIGURE 4.16: Normalised static energy value for different benchmark applications for IPC degradation threshold = 2, with 4MB L2 cache.

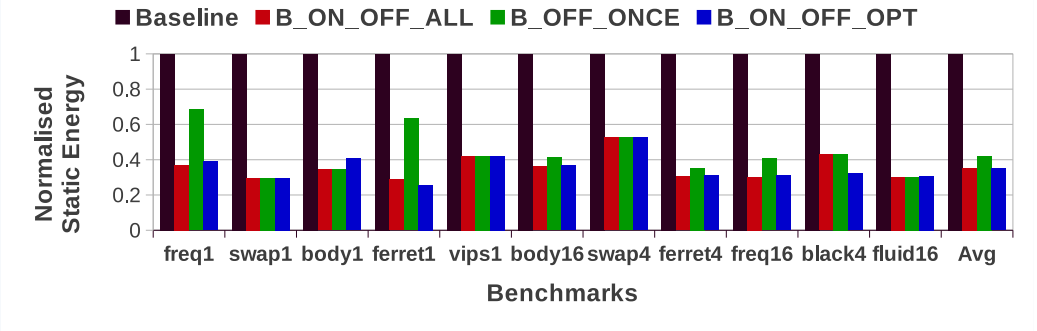


FIGURE 4.17: Normalised static energy value for different benchmark applications IPC degradation threshold = 4, with 4MB L2 cache.

B_OFF_ONCE and B_ON_OFF_OPT, respectively; and for 4% the respective values for B_ON_OFF_ALL, B_OFF_ONCE and B_ON_OFF_OPT are 27%, 24% and 31%. It can be observed that, the savings for B_OFF_ONCE are less compared to both B_ON_OFF_ALL and B_ON_OFF_OPT. As B_ON_OFF_OPT allows resizing at various points during the process execution, it takes the maximum advantage of the proposed policy. Relaxing the performance constraint leads to more power savings.

Static Energy Savings

Figures 4.16 and 4.17 show the static energy savings for IPC degradation thresholds of 2% and 4%, respectively. The graph for 3% threshold is given in Figure 4.7. In particular, for 2% the savings are 66%, 46% and 65% for B_ON_OFF_ALL, B_OFF_ONCE and B_ON_OFF_OPT, respectively; and for 4% the values are 65%,

58% and 69% for B_ON_OFF_ALL , B_OFF_ONCE and B_ON_OFF_OPT, respectively. The savings are more if the constraint is relaxed for B_OFF_ONCE. But in case of both B_ON_OFF_ALL and B_ON_OFF_OPT, relaxed constraint results less cache resizing, whereas the stringent one resizes cache frequently. But static energy saving is almost same for both relaxed and stringent constraints, due to the fact that, same amount of cache portions are shutdown for a same time interval, overall. In case of B_OFF_ONCE, relaxed constraint turns on lesser cache banks than the stringent one, hence the energy saving is more with relaxed constraint.

4.4.4 Analysis of proposed policy on a larger cache

As the cache is larger, there is more opportunity of having unused cache portions to be shutdown. This leads to more energy savings. For IPC degradation threshold of 3% the EDP savings are 40%, 33% and 34% for B_ON_OFF_ALL , B_OFF_ONCE and B_ON_OFF_OPT, respectively. The savings in static energy are 67%, 59% and 63% for B_ON_OFF_ALL , B_OFF_ONCE and B_ON_OFF_OPT, respectively. Figures 4.18, 4.19 and 4.20 show the values for EDP gains for IPC degradation thresholds of 2%, 3% and 4%, respectively; and Figures 4.21, 4.22 and 4.23 show the static energy savings for IPC degradation thresholds of 2%, 3% and 4%, respectively.

Similar to the case of a 4MB cache, relaxing the IPC constraint saves more static energy and overall EDP savings. In all cases, policy B_ON_OFF_OPT performs better compared to B_ON_OFF_ALL and B_OFF_ONCE for both the cache sizes. As B_ON_OFF_OPT runs the reconfiguration at intervals, it has more control on the cache size. It also helps to keep the IPC degradation within limits as there are more chances to rectify.

Table 4.5 summarises the obtained results. We also have compared our results with drowsy cache techniques [1] which has been summarised in Table 4.6.

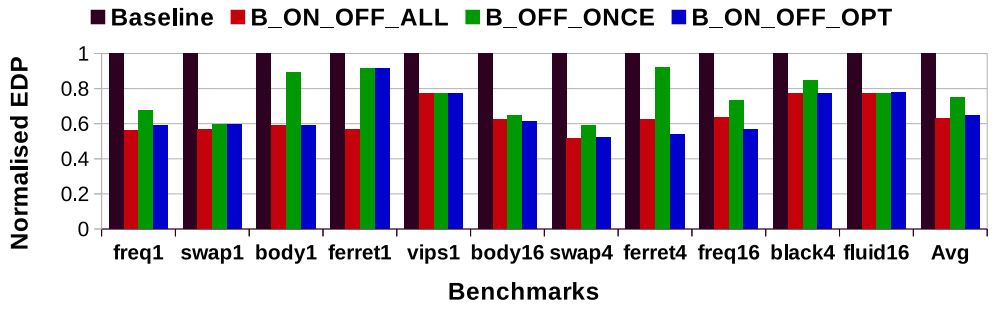


FIGURE 4.18: Normalised EDP value for different benchmark applications for IPC degradation threshold = 2, with 8MB L2 cache.

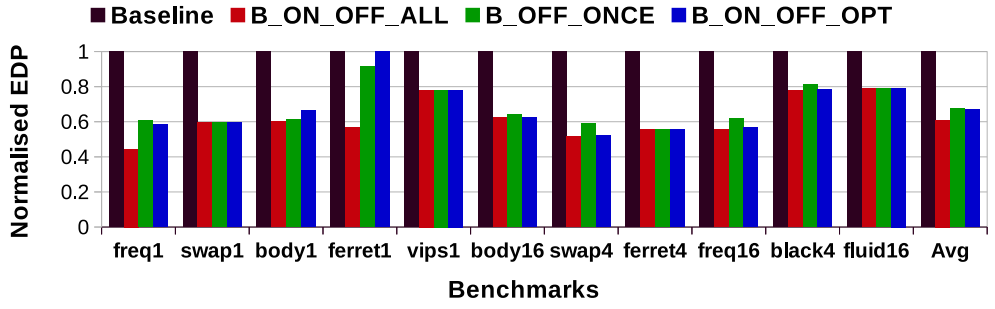


FIGURE 4.19: Normalised EDP value for different benchmark applications for IPC degradation threshold = 3, with 8MB L2 cache.

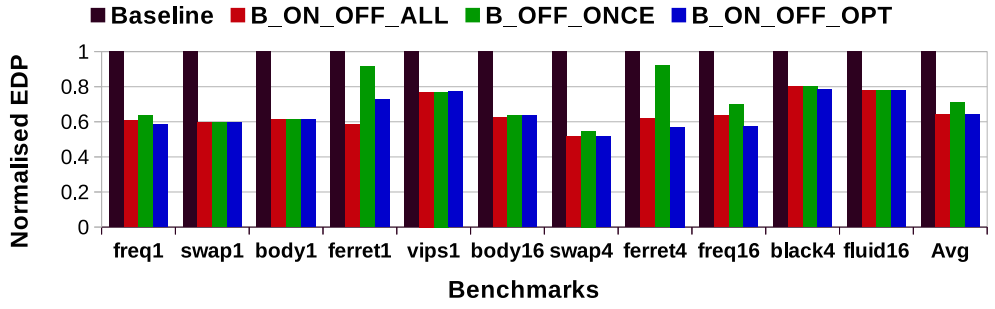


FIGURE 4.20: Normalised EDP value for different benchmark applications for IPC degradation threshold = 4, with 8MB L2 cache.

4.4.5 Summary

Our simulation results claim B_ON_OFF_OPT as the best policy for this performance aware static energy savings, which gives an insightful design choice. Size of on-chip LLC is an important metric while designing TCMP architecture. Large LLCs are better as they provide adequate cache space to fit the WSS of the applications, but larger caches consume huge static energy. As WSS, which is only

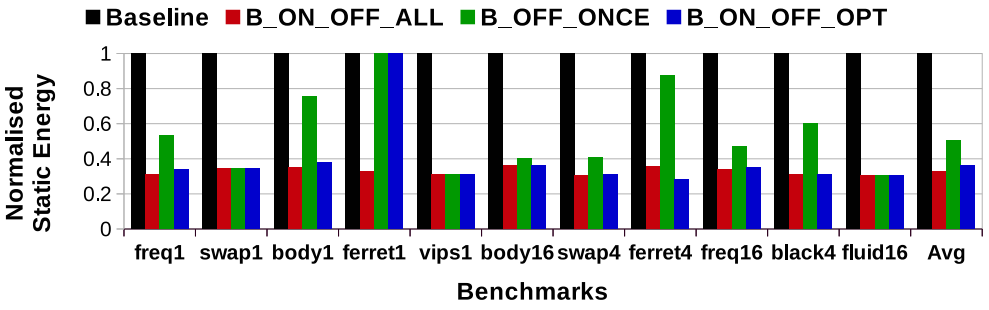


FIGURE 4.21: Normalised static energy value for different benchmark applications for IPC degradation threshold = 2, with 8MB L2 cache.

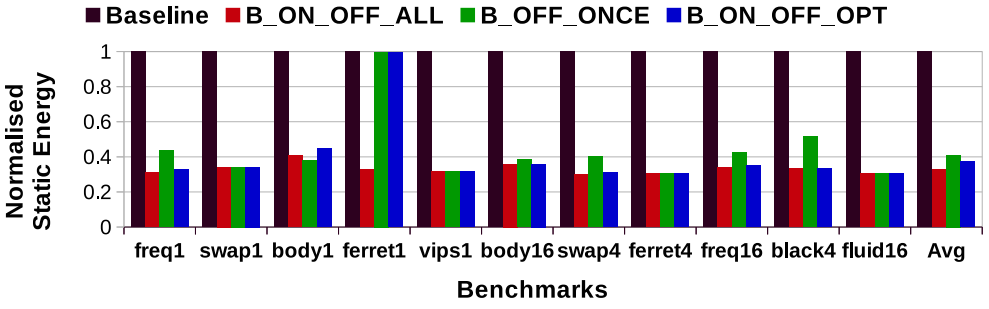


FIGURE 4.22: Normalised static energy value for different benchmark applications for IPC degradation threshold = 3, with 8MB L2 cache.

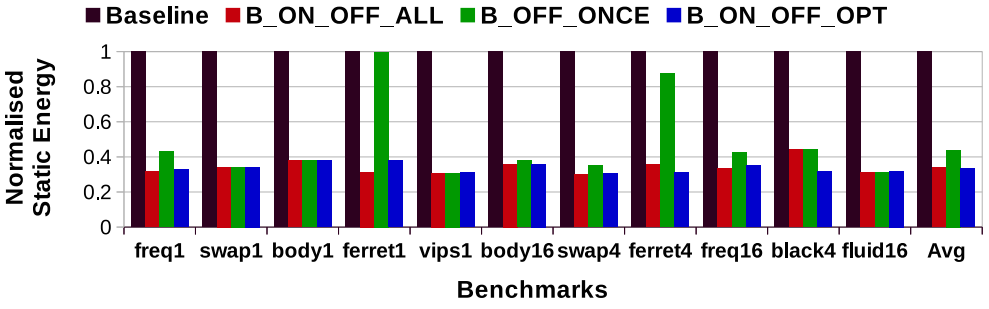


FIGURE 4.23: Normalised static energy value for different benchmark applications IPC degradation threshold = 4, with 8MB L2 cache.

known at runtime, varies across the applications, hence, it is always better to design a dynamically reconfigurable cache which turns off unused cache portions on-the-fly to reduce static energy. And the cache portions will also be dynamically turned on later if application's WSS does not fit in the cache. Moreover, from performance perspective, multi-banked cache design provide such relaxation to the addressing scheme, where blocks can be easily placed inside any of the bank.

Therefore, redirection of addresses at the turned off banks, does not require any extra burden of remapping.

L2 Cache Configurations	**IPC Degradation Threshold**	**Static Energy Savings**			**EDP Savings**		
		B1	**B2**	**B3**	**B1**	**B2**	**B3**
4 MB 4 Way	2%	66%	46%	65%	26%	19%	29%
	3%	66%	59%	65%	29%	27%	30%
	4%	65%	58%	69%	27%	24%	31%
8 MB 4 Way	2%	67%	50%	64%	38%	25%	36%
	3%	67%	59%	63%	40%	33%	34%
	4%	66%	56%	67%	36%	30%	36%

TABLE 4.5: Summary of the results obtained from our experiments with respect to baseline architecture. Here, **B_ON_OFF_ALL**, **B_OFF_ONCE** and **B_ON_OFF_OPT** are denoted by **B1**, **B2** and **B3**, respectively, in this Table.

L2 Cache Configurations	**IPC Degradation Threshold**	**Static Energy Savings**			**EDP Savings**		
		C1	**C2**	**B3**	**C1**	**C2**	**B3**
4 MB 4 Way	3% (for B3)	46%	36%	65%	17%	13%	30%

TABLE 4.6: Summary of the results obtained from our experiments with respect to baseline for Drowsy cache [1] and **B_ON_OFF_OPT** (denoted as **B3** in this Table). **C1** and **C2** represents **Drowsy_C1** and **Drowsy_C2**, respectively.

4.5 Conclusion

A static energy saving technique by tuning on-chip LLC size at bank level granularity is presented in this chapter. Each of the applications used here has a different working set requirement. Even the distribution of data across the cache banks is not uniform. Initially, usage patterns for different benchmarks are collected and it is dynamically decided whether to turn-off or turn-on the cache banks. The content of the shutdown bank is migrated to another powered on bank which will also handle future requests for this shutdown bank. The policy keeps track of degradation in performance during execution and decides the shutdown or turn on process once the IPC degradation reaches an allowable threshold. The shutdown process reduces the static energy consumption but too much reduction in cache size may degrade system performance. To compensate this, turning on process turns on

cache banks to provide application more cache space on-demand. For a 16-core setup with a performance degradation constraint of 3% with 4MB L2 cache, we were able to dynamically tune the cache size and obtained EDP savings of 30% on average. The total savings in static power of 65% compensates the overhead of 1% increment in network power and IPC degradation of 1.8%.

Three versions of the policy are presented: namely B_ON_OFF_ALL , B_OFF_ONCE and B_ON_OFF_OPT. The first policy, B_ON_OFF_ALL allows unrestricted on-off options depending on the performance values. B_OFF_ONCE takes a conservative approach of allowing bank shutdown followed by only restart. Whereas the third policy, B_ON_OFF_OPT, takes a restricted approach of B_ON_OFF_ALL. Policy B_ON_OFF_OPT gives the best savings compared to B_ON_OFF_ALL and B_OFF_ONCE. The EDP gains and static energy savings are maximum in B_ON_OFF_OPT. At the same time the execution time of B_ON_OFF_OPT is better compared to that of B_ON_OFF_ALL, mainly on account of lesser data movement and reconfiguration stall times. For a larger cache the savings are more as there is more opportunity to reduce the cache size. We perform experiment to trade the performance for power, in that if we relax the performance constraint we can obtain more power savings. In particular, relaxing IPC constraint from 2% to 4%, we get 4% extra static energy savings and 2% extra EDP gains. Among all the cases, policy B_ON_OFF_OPT is stable and adapts to the changing cache requirements of the application. B_ON_OFF_OPT gives better savings than existing works like BSP and Drowsy. In particular, for B_ON_OFF_OPT, we get 30% EDP gains and 65% static energy savings which is more than Drowsy for a 4MB 4way set associative L2 cache.

As the remapping information is kept inside the L2 controllers, the L1 caches are totally transparent in this respect. Thus no extra remap information is maintained with the L1 caches which reduces the overhead of maintaining the remap information in the remap tables with L1. Additionally, cache size is tuned by turning off or turning on the cache banks, instead of selected ways from the sets; this in-turn simplifies the power gating circuitry.

Chapter 5

Reducing Static Energy Consumption in Way Sharing LLCs: DiCeR with DAM

While analysing the TCMP based DiCeR proposed in the previous chapter, we can derive the following observations:

- The achieved leakage saving is significant while shutting down the cache banks.
- The performance degradation is also less than 2% especially in the case of B_ON_OFF_OPT.
- Target banks, those handling the additional requests, are heavily loaded and hence they experience more number of misses.
- For a set of applications, heavily accessed banks have non-uniform access patterns across their cache sets/ways.
- Distance cognizant target selection can further reduce the NoC overhead.

The increment in the number of misses in the target banks can be reduced by incorporating some DAM based techniques. In addition to that, leakage energy

consumption can be reduced by using a combination of cache bank shutdown and way shutdown, where the banks with minimal usages are candidates for shutdown like in BSP, and in some moderately used banks a number of ways are turned off to save leakage. The associated DAM based technique will mitigate the impact of smaller set-size in those banks where ways are turned-off.

5.1 Introduction

Dynamic shutdown of the least accessed banks although saves leakage significantly, but, if several banks are gated, it hampers the system performance by incurring more capacity and conflict misses, especially at the target banks in BSP (ref. Chapter 4). These banks are usually heavily loaded as they handle an amount of additional loads of the turned off banks. Hence, with the increased number of turned off banks, the loads at targets are also hiked, that results into increment in conflict and capacity misses at targets. While analysing these load distribution across the sets of a target bank, the diversity in access pattern is noticeably high. Some sets are heavily used and some have lesser amount of loads. Moreover, the heavily used sets have higher miss rates with compared to the others. DAM based techniques have enough potential to reduce this adverse effect by increasing the associativity of the sets at target banks. As a result, the misses at targets are reduced which in turn ameliorates the system performance. Now, this enhanced performance can be traded further to save more leakage by turning off more cache portions.

One can resort to a mid-way solution by turning off some banks completely in a TCMP based architecture (as shown in Figure 5.1) and from the remaining banks turn-off some ways from every cache set i.e. exploitation of cache resizing at multiple/hybrid granularities. Like the existing diversities in load distribution across the sets, cache ways are also used in a non-uniform manner, which are shown in Figure 5.2 and 5.3. These statistics are collected by running Body16 and Freq16 benchmarks for a certain amount of time-span. However, if we can have

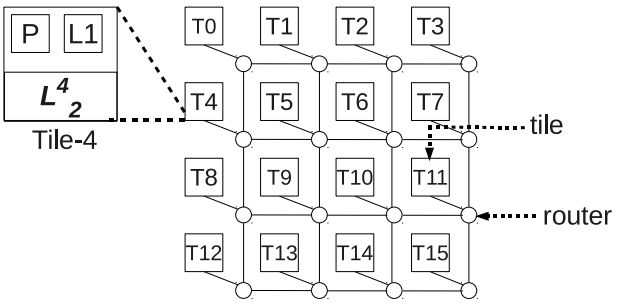


FIGURE 5.1: Tiled CMP architecture

some mechanism(s) to use the cache space of the lightly used sets, it will give us an opportunity to shutdown some ways from every cache set. This is also possible by employing DAM on the cache.

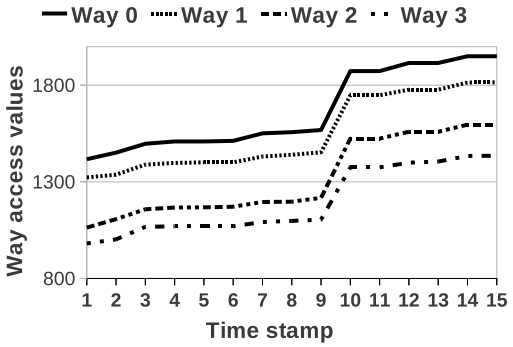


FIGURE 5.2: Way Access pattern in a bank for benchmark: Body16.

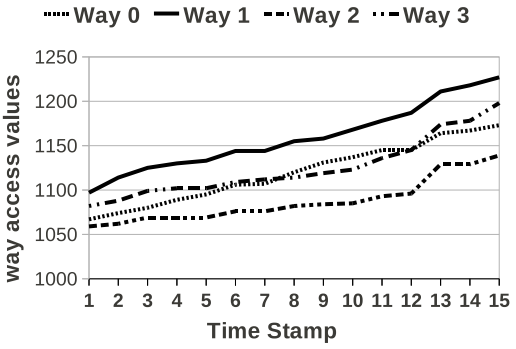


FIGURE 5.3: Way Access pattern in a bank for benchmark: Freq16.

Among the various DAM techniques available in the literature, we have selected CMP-SVR [34]. In this policy, each cache set is divided into two parts: Normal sTorage (NT) and Reserve sTorage (RT). The RT-section from each set can be shared among a group of sets thereby increasing their associativity. This helps to recover the performance loss and also creates new opportunity to shutdown some ways from every set. The CMP-SVR mechanism has already been elaborated earlier in Chapter 2.

The request redirection and data migration to the target banks in BSP and DiCeR play a major role in NoC, from a power/performance perspective, that we are going to discuss in the next section. However, these additional loads increase the NoC traffic, hence both NoC latency and power consumption are exaggerated. To

mitigate this issue, target banks are to be selected within close proximity of the victim, that can reduce the NoC overhead. Furthermore, if a moderately used bank is selected as a target, then NoC congestion at that location will trivially be lesser than selecting a heavily used bank as a target. Additionally, selecting a heavily used banks also increases misses while playing the role of a target bank.

The main contributions of this work can be summarised as follows:

1. Dynamically shutdown the least used cache banks and redirect the traffic to other banks (target banks).
2. Towards mitigating NoC overheads, select target as close as possible to the powered off bank.
3. Apply DAM technique on all the powered-on banks.
4. Progressively shutdown cache ways from the powered-on banks.

In this work, all of these energy saving policies are implemented in a 16 core-based TCMP architecture, which is shown in Figure 5.1.

5.2 Memory Latency in DiCeR

Before going into our proposed policy, in this section we first discuss about how DiCeR effects the system performance (i.e. IPC) in a CMP having multi-banked LLC on-chip. Towards this, we consider a CMP having N number of (homogeneous) cores, that associated with B number of LLC banks (of same size), out of which b banks are turned on. As, system wide IPC depends upon two performance parameters: (a) Computation Time and (b) Memory Access Time, hence the following equations can able to represent the system-wide IPC as follows:

$$IPC = \frac{1}{N} \sum_{i=1}^{N} IPC_i, \tag{5.1}$$

where,

$$IPC_i = \frac{IC_i}{CC_i + MC_i(b)} \tag{5.2}$$

Here, system IPC is the arithmetic mean of IPC_i, where i lies between 1 to N. IPC_i represents the IPC at i-th core, where CC_i and MC_i are the cycles required for computation and memory operations, respectively at the i-th core.

In Equation 5.2, as CC_i includes L1 access latency, hence, MC_i only represents L2 accesses as follows:

$$MC_i(b) = \sum_{j=1}^{b} (h_j^i.d_j^i + a_o^i j.d_o), b \leq B. \tag{5.3}$$

Note that, L2 is considered as on-chip LLC like our earlier assumption. Here, h_j^i is the number of hits at bank j, whose requests have been generated at core i. The d_j^i is the delay at bank j to send block to core i. $a_o^i j$ implies the number of off-chip accesses due to misses at bank j which have been requested by core i, and (uniform) off-chip access latency is represented by d_o.

Initially, in Equation 5.3, if all banks are turned on, then $b = B$. Shutting down of cache banks introduces the target banks, which incurs a few extra cycles for the remapped requests to reach at target. So, Equation 5.3 can be rewritten as follows:

$$MC_i(b) = \sum_{j=1}^{b} (h_j^i.d_j^i + a_o^i j.d_o) + \sum_{k=1}^{B-b} r_k^i.n_t^i k, b \leq B. \tag{5.4}$$

r_k^i in Equation 5.4 is the number of remapped requests at $(B-b)$ number of turned off banks, generated by core i. Number of NoC cycles required to reach at target t (from k) is represented by $n_t^i k$.

During reconfiguration, system also needs a few clock cycles to move data from victim to target or vice versa, which is negligible if it is done limited number of times. However, Equation 5.1 can be rewritten as:

$$\begin{aligned} IPC(b) &= \frac{1}{N}\sum_{i=1}^{N} IPC_i(b) \\ &= \frac{1}{N}\sum_{i=1}^{N}(\frac{IC_i}{CC_i + MC_i(b)}) \\ &= \frac{1}{N}\sum_{i=1}^{N}(\frac{IC_i}{CC_i + \sum_{j=1}^{b}(h_j^i.d_j^i + a_o^i j.d_o) + \sum_{k=1}^{B-b} r_k^i.n_t^i k}) \end{aligned} \tag{5.5}$$

Reduction in b reduces cache capacity, hence, the number of misses i.e. $a_o^i j$ increases, so, the decrement in h_j^i (Equation 5.5). As, $d_o > d_j$, therefore, increment in $a_o^i j$ curtails performance by increasing MC_i. Moreover, r_k^i also increases for more reduction in b. So, Equation 5.5 clearly shows how performance is related to the number of cache banks (b). Hence, it can be stated that, dynamic reduction in cache size can degrade system performance by incurring more misses and NoC cycles, which ultimately curtail the system performance. To address these issues, in this proposed energy saving policy, we adapt CMP-SVR [34] for alleviating performance at the powered on banks and select nearby prospective bank as target for reducing NoC overheads.

5.3 Proposed Energy Saving Policy

The physical distribution of L2 cache in multiple banks offers individual power supply circuit for each bank, due to which dynamically gating of a bank's power supply [65] becomes easier with a simple circuitry. As, locality of reference property of cache memory hypothetically states that, the least used cache banks will have the least usages in the future. Hence, shutting down of a least used cache bank during execution can save leakage energy. However, if we turn-off several

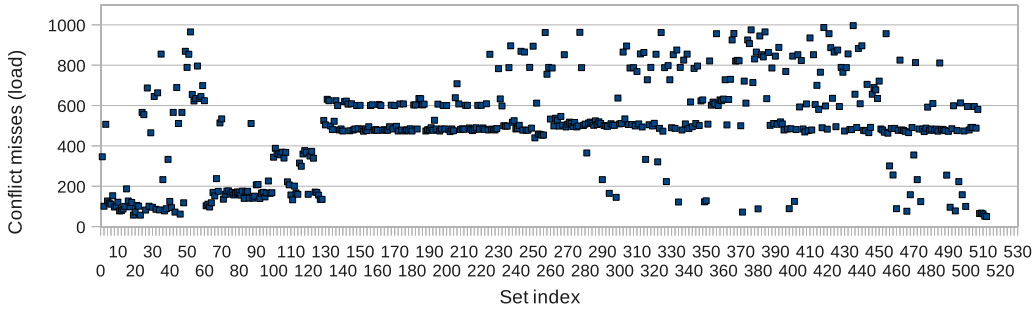


FIGURE 5.4: Set usage profile of a bank in a TCMP.

banks it will drastically affect system performance if the applications' WSS becomes larger than the available cache space. Furthermore, increased workloads at the target banks will abruptly increase the conflict and capacity misses at the sets of target banks, which also follows a pattern like Figure 5.4.

Hence, we initially apply DAM based policies at the target banks to improve performance at targets. Secondly, we select some non-lightly used banks, out of which, the nearest one from this set of banks is chosen as target bank. This modified BSP is termed as BSP_SVR, in our work. The usage aware and distance cognizant target selection improves performance of BSP in BSP_SVR, which we further traded to save more leakage. As a result, with the same performance degradation, we are able to save more leakage energy than BSP. Additionally, location of targets within close proximity reduces NoC latency and power consumption as well. This improvement in NoC energy and performance also increase the gains in EDP across the applications.

5.3.1 BSP vs. BSP_SVR: A Comparative Analysis

To see the effect of BSP_SVR, we have done a set of preliminary experiments in our simulation setup. The results for a 4MB 4 way cache are shown in Figure 5.5-5.8. We have achieved 66% savings in static energy which is shown in Figure 5.5, with 45% average gains in EDP (ref. Figure 5.8) in case of BSP_SVR. Both of these values claim the superiority of BSP_SVR over BSP. More EDP gains in BSP_SVR over BSP is achieved due to reduced NoC energy consumption and lesser

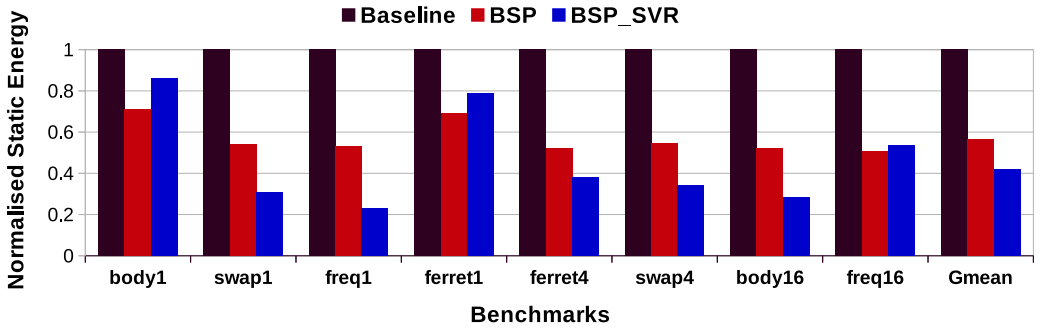


FIGURE 5.5: Savings in Static Energy with BSP_SVR.

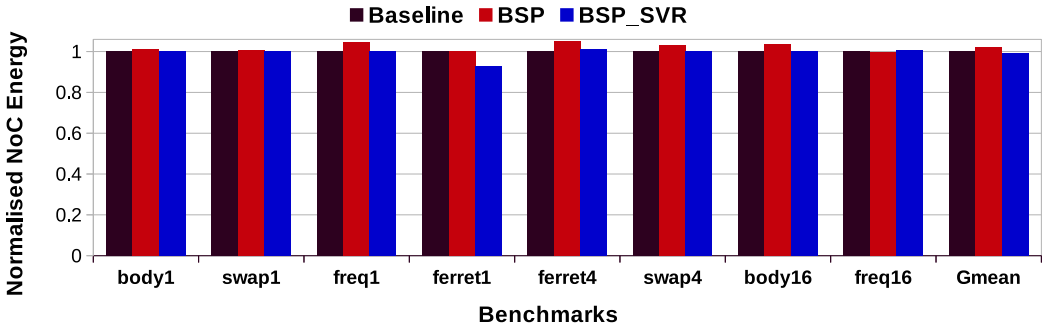


FIGURE 5.6: Savings in NoC Energy with BSP_SVR.

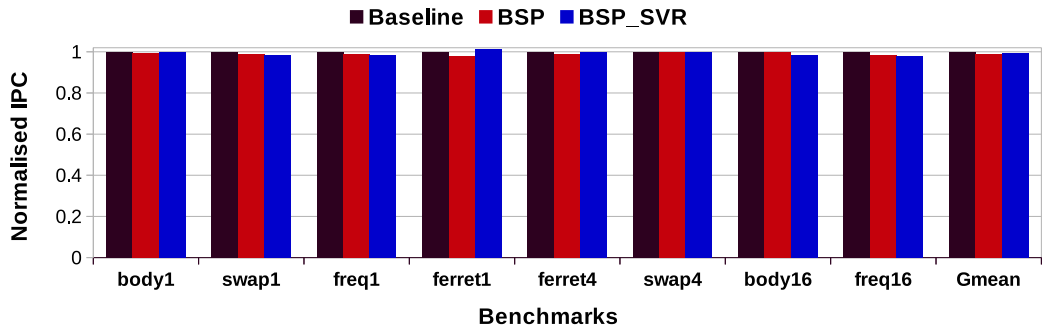


FIGURE 5.7: Change in IPC: BSP vs BSP_SVR.

IPC degradation than BSP, which are shown in Figure 5.6 and 5.7, respectively. The performance degradation threshold has been fixed at 3% for both BSP and BSP_SVR while performing the simulations.

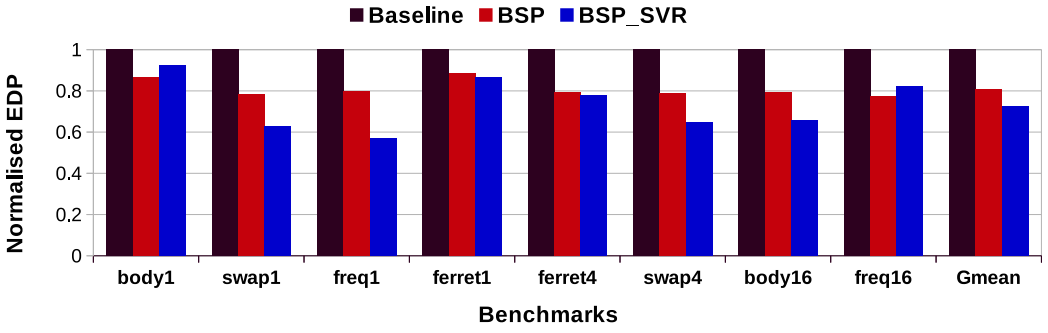


FIGURE 5.8: EDP gains: BSP vs BSP_SVR.

5.3.2 DiCeR with DAM at Multiple Granularities

Applying CMP-SVR in a cache bank increases its effective associativity, hence, more leakage saving can be achieved by shutting down some of the cache ways. Therefore, we can also resort to the technique to way shutdown at the powered-on banks by using power gating circuitry [65]. Here again, the effect is that due to way shutdown the associativity of the set reduces thus increasing the conflict misses. But, performance aware way shutdown can help us to keep the IPC within a permissible (degradation) limit.

The proposed dynamic cache resizing saves more leakage energy of LLC (L2 cache in this case) without significant degradation in performance. It goes through three main phases:

- Bank shutdown: In the bank shutdown phase, least used cache banks are selected and turned off. The future requests to these banks are redirected to other powered on (target) banks. This phase of bank shutdown continues until either the performance degrades beyond the allowable threshold or if the number of banks turned off reaches a predefined maximum limit.

- Way shutdown: By bank level shutdown we are able to save on leakage energy. However, banks with average usages are not suitable candidates for complete power off. Instead, in such banks it is desirable to turn off some number of ways from every set.

- Associativity management: After way shutdown the effective associativity of the bank reduces, thus increasing the number of conflict misses. In particular, the sets which are in high demand will suffer from more misses and might lead to poorer performance. To mitigate these two issues, we apply CMP-SVR to dynamically increase the associativity of the sets. This allows better utilisation of the available bank capacity.

Algorithm 4 gives the details of the process. The application is run for certain number of cycles before deciding to reconfigure. This interval can be decided by profiling. The cache bank usages are collected by the controller, by attaching a counter in each bank for dynamic monitoring of the accesses, and the banks with minimum usages are decided to be shutdown. This process of bank shutdown is repeated on other banks until the system is able to maintain the performance. In case the performance starts degrading beyond a predefined limit, the bank shutdown process is stopped. The process is also stopped in case the number of banks turned off reaches a maximum limit (which can be set by using profiling). This limit also prevents application thrashing. After selecting the bank to turn-off (called as victim bank), its valid data blocks are transferred to a selected target bank like BSP/DiCeR. The placement of redirected/forwarded data in target banks also follows the same mechanism like DiCeR, i.e. an implementation of Reuse Counter [18].

The target bank selection is done by choosing another active bank with average usage and which further has to be in close proximity (as per network hop distance) to the shutdown bank like BSP_SVR. Distance cognizant target bank selection reduces network overhead of data migration and also the latency for subsequent requests forwarding from shutdown bank to the target bank as it was in BSP_SVR. The target bank is chosen with average usage because a bank with heavy usage will be unable to handle the additional load whereas a lightly used bank is a possible candidate to shutdown. Like DiCeR, the shutting down of the target banks is not permitted in order to avoid transitive redirection of requests. During block

ALGORITHM 4: Algorithm for cache resizing

T : Reconfiguration interval
δ : Permissible percentage degradation in IPC
m : Maximum limit on bank shutdown
$j = 0$: Number of banks turned off. Initially zero.
while $(j < m)$ **do**
  Run the application for T number of clock cycles.
  Compute degradation in IPC compared to original average IPC.
  if degradation is greater than δ **then**
    do not shutdown bank.
    break.
  end if
  CALL *manageShutdown()*
  j++
end while
Run application with available number of cache banks for T number of clock cycles.
Call *wayShutdown()*
Keep current configuration until end of execution.

Function : *manageShutdown()*
Calculate the usage for every active cache bank.
Select minimum used cache bank, B_i, as a victim bank.
Select another bank, B_j as the target bank, that is closest to B_i and has average usage.
if CMP-SVR is not activated on B_j **then**
  activate CMP-SVR on B_j.
end if
Stall requests for B_i, but keep the response queue open.
Migrate all valid blocks from B_i to B_j. Remap all future request to B_j.
Turn-off bank B_i.

Function : *wayShutdown()*
Activate CMP-SVR in non-target powered on banks (if any).
Turn off some ways from the NT partition of the active banks.
Run the system for T number of Clock cycles.
if IPC degradation over last reconfiguration period is lesser than δ **then**
  Turn off ways from RT partition of average usage powered on banks.
end if
Return.

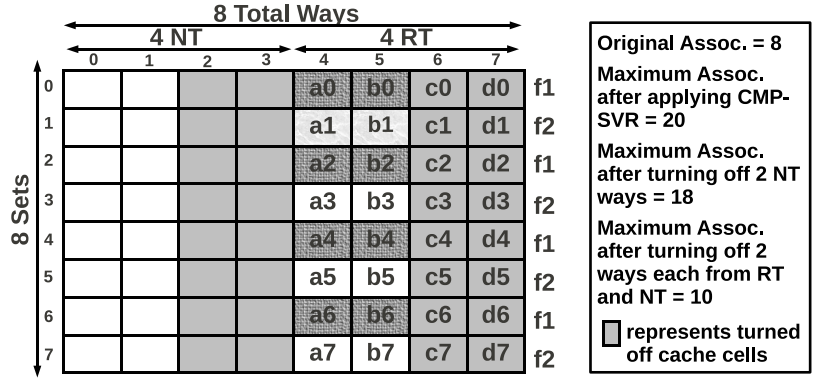


FIGURE 5.9: Way shutdown using CMP-SVR.

transfer only the accesses to the victim bank are stalled, and the other components can continue execution. The last step is to enable CMP-SVR on this target bank.

5.3.2.1 Effects of Way Shutdown on CMP-SVR

Once the limit to the turned off banks is reached: either due to performance degradation or due to maximum bank off limit, one can further save static power by attempting to turn off cache ways from the currently active cache banks. We choose to turn off ways from the NT-partition. This is done, because CMP-SVR helps to increase associativity using the RT partition of the sets. Thus even if the set has lesser ways in NT, a set in high demand can still fulfill its requests by spilling data in RT-partition of its friend sets. One can even become ambitious and shutdown ways from the RT-partition. The proposed policy attempts to even shutdown ways from RT-partition, provided it does not degrade performance beyond limits. The results are shown for both categories: NT-only shutdown and NT as well as RT shutdown. All dynamic reconfiguration overheads along with the remapping have been considered during simulation.

Figure 5.9 illustrates the way shutdown proposal for the sample cache. Here the cache is 8-way associative having 8 sets. There are two fellow groups with 4 sets each. Each set is divided into 4-ways for RT and 4-ways for NT. The basic set associativity is 8. Applying CMP-SVR with the help of fellow group of size 4, we get maximum associativity of $(8 + 4 \times 3 =)$ 20. If we shutdown 2 ways from NT

Cache Parameters	Values
Cache Level	L2
Size of a L2 Bank	256 KB / 128 KB
Block Size	64 Bytes
Technology used	32nm
Associativity	4 / 8
Cache Model	NUCA
Operating Temperature	360 K
Actual Cache Size	4 MB / 2 MB

Table 5.1: CACTI Configurations

Components	Parameters
No. of Tiles	16
Processor	UltraSPARCIII+
L1 I/D Cache	64KB, 4-way
L2 Cache bank	128KB/256KB, 8-way
Memory bank	1GB, 4KB/page
Flit Size, Buffer Size	16 bytes, 4
Pipeline Stage	5-stage
VCs per Virtual Network	4
Number of Virtual Networks	5

Table 5.2: System and Network Parameters

portion, we are still able to maintain performance as the maximum associativity is now 18. Shutting down of 2 ways each from NT and RT results in maximum associativity of 10.

5.4 Experimental Evaluation

To validate our idea, a 16 core TCMP setup has been used for experiment as shown in Figure 5.1. We have implemented our proposed policy in GEMS+Simics simulation setup which have been already discussed in Chapter 3. CACTI 6.5 is used for simulating cache power/energy consumption. Table 5.2 contains configuration details of the processor, memory and NoC configuration used in experiments. The CACTI configuration, used in this experiment is given in Table 5.1. We have used PARSEC benchmark suite for validation which are also discussed in Chapter in 3.

5.4.1 Results and Analysis

The proposed policy turns off cache banks up to a limit and then turns off ways from the remaining active banks. We term the policy as "WS". When the way turn off is implemented in the NT-partition, the policy is termed as "WS_NT". When the policy attempts to turn off ways from both NT and RT partitions, it is termed as "WS_NT_RT". Note that turning off ways from only RT partition will not be able to take advantage of DAM, and hence we do not consider this option.

The proposed policies are compared with baseline tiled CMP and two existing approaches: Drowsy cache [1] and simple bank shutdown. The drowsy cache configurations that have been compared here are DR_C1 and DR_C2 with having 25% and 50% ways, respectively, are in the normal mode and the remaining ways are in the low-power mode. The simple bank shutdown policy turns off banks depending on their workloads up to a given performance degradation threshold, which is termed as "BSP" (ref. Chapter 4). In our experimental setup we have considered cache with size 4MB. We have experimented with associativity 4-way and 8-way.

Gains in EDP Figure 5.10 shows the normalised EDP values for a 4MB 4-way associative L2 cache. The two drowsy cache configurations DR_C1 and DR_C2 obtain average EDP savings of 14% and 11%, respectively with respect to the baseline. BSP has average EDP savings of 18%. The proposed architecture saves 21% and 24% average EDP for WS-NT and WS-NT-RT respectively with respect to the baseline.

Static Energy savings The savings obtained in cache leakage energy are shown in Figure 5.11 for various benchmarks. Drowsy cache has savings of 40% and 27% for DR_C1 and DR_C2, respectively. BSP saves on average 34% static energy. The proposed technique outperforms BSP and Drowsy by saving on average 44% and 47% for WS_NT and WS_NT_RT, respectively. The turned off cache portions not only save energy rather it can be used to control the effective chip temperature.

Effect on Higher Associative Caches A 4MB 8-way L2 cache obtains average EDP gains of 33% and 35% with respect to baseline for WS-NT and WS-NT-RT, respectively. The respective static savings for WS-NT and WS-NT-RT are 68% and 70%. Figure 5.12 shows the EDP gains and Figure 5.13 shows the static energy savings for the various benchmark applications. The results have been summarised in Table 5.3.

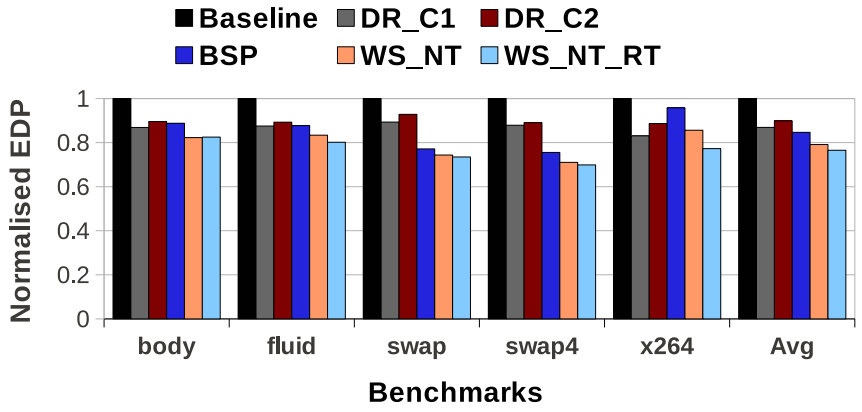


FIGURE 5.10: Savings in EDP in comparison with BSP and Drowsy Cache for 4MB 4-way L2.

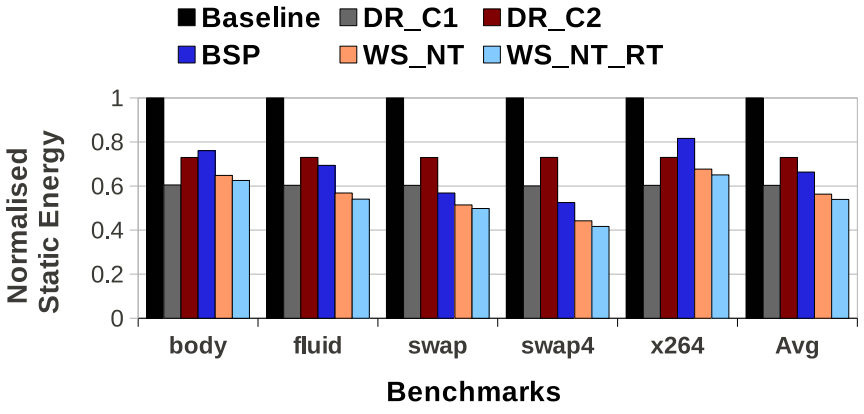


FIGURE 5.11: Savings in Static Energy in comparison with BSP and Drowsy Cache for 4MB 4-way L2.

As can be observed the EDP gains in the proposed architecture are more for an 8-way associative cache as compared to a 4-way associative cache of the same size (note that EDP gains for 4-way are around $21 - 24\%$). This improvement is achieved as in a higher associative cache we can shutdown more portion of the set belonging for the NT-partition, yet maintaining performance due to associativity

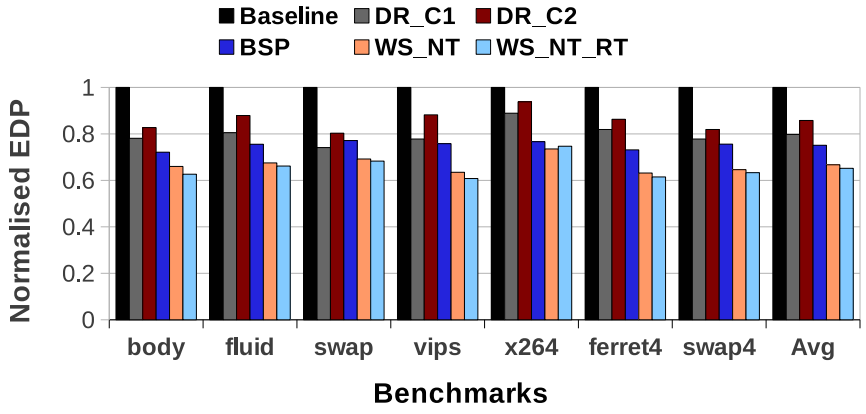


FIGURE 5.12: Savings in EDP in comparison with BSP and Drowsy Cache for 4MB 8-way L2.

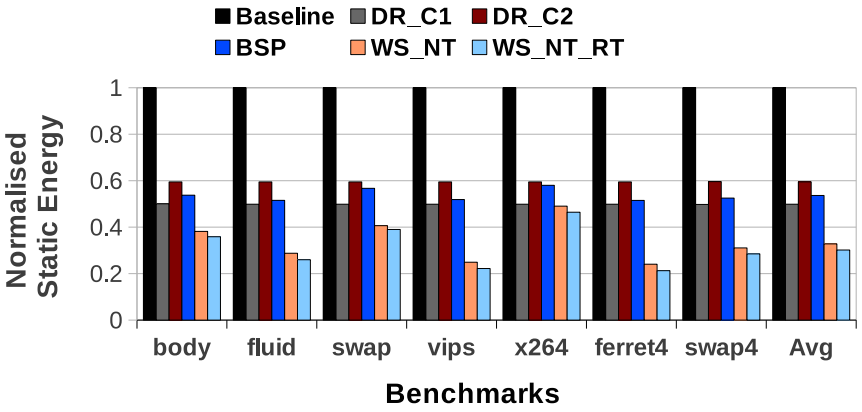


FIGURE 5.13: Savings in Static Energy in comparison with BSP and Drowsy Cache for 4MB 8-way L2.

Parameters	DR_C1	DR_C2	BSP	WS_NT	WS_NT_RT
EDP gains	21%	15%	25%	33%	35%
Static Energy	51%	41%	47%	68%	70%
IPC Degradation	1.15%	1.0%	-0.9%	2.1%	2.0%

TABLE 5.3: Summary of improvement over baseline for a 4MB 8-way set associative L2 cache.

management. Note that these savings are obtained with IPC degradation of merely 2%.

Effect on Smaller Sized Cache Figure 5.14 shows the EDP savings and Figure 5.15 shows the static energy savings for a 2MB 8 way associative L2 cache. The EDP savings are around 15% and 17% and static energy savings are 49% and

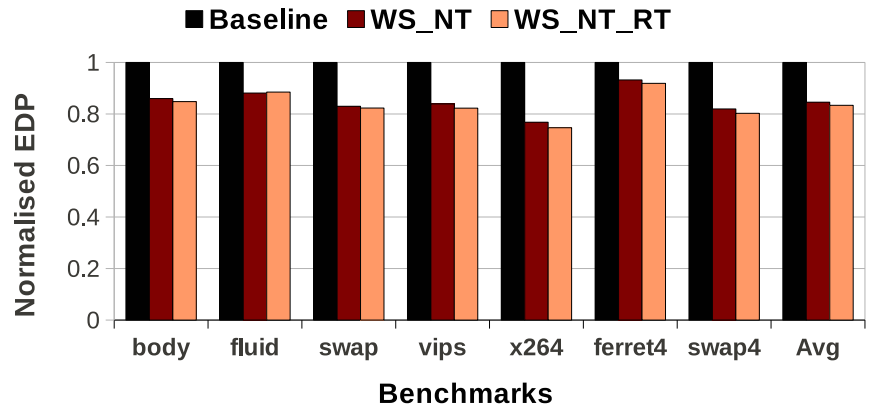


FIGURE 5.14: Normalised EDP gains for 2MB 8-way L2 with respect to baseline.

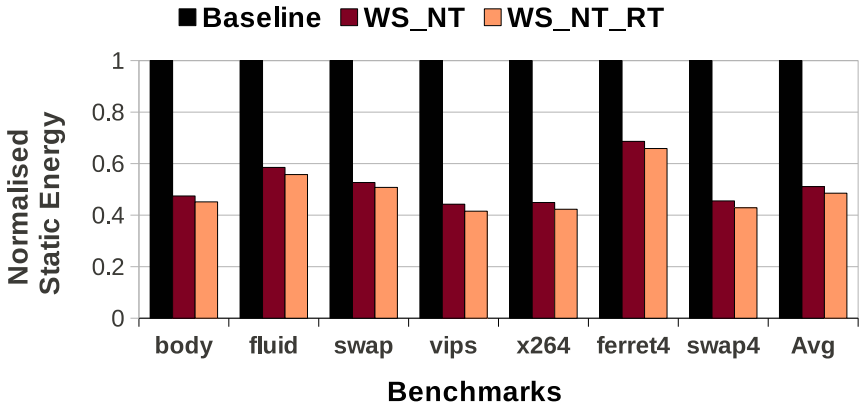


FIGURE 5.15: Normalised Static Energy values for 2MB 8-way L2 with respect to baseline.

52% for WS_NT and WS_NT_RT, respectively over the baseline. If the cache size is small, the scope for only bank shutdown reduces due to capacity issues, hence the combination of bank and way shutdown helps to save energy. These energy savings are achieved with a IPC degradation of 2.6%, on an average.

Controlling IPC Degradation If an application changes its WSS later during execution, then performance degrades, especially for smaller caches. To rectify this, some of the turned off ways are selectively turned on at runtime from RT and/or NT parts of the banks. Here we get 40% savings in static energy (Figure 5.16) which is slightly lesser than earlier value of 52% as some ways are turned-on. As expected, in this case we are able to maintain performance (Figure 5.17)

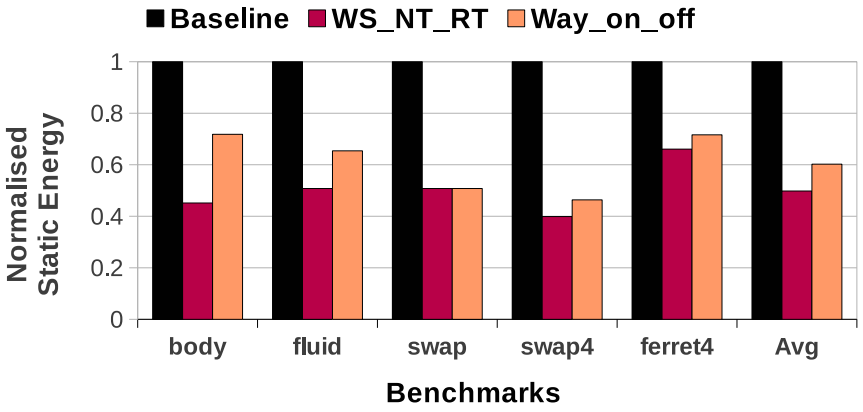


FIGURE 5.16: Normalised Static Energy values for 2MB 8-way L2 for way-off as well as way turn-on policy.

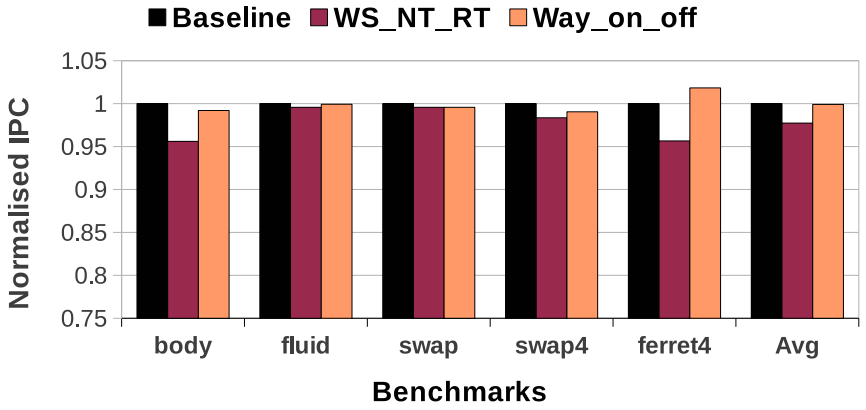


FIGURE 5.17: Normalised IPC values for 2MB 8-way L2 for way-off as well as way turn-on policy.

compared to 2.6% degradation in earlier instance.

Overhead The proposed technique requires an additional mapping table to implement CMP-SVR, which leads to area overhead of 9.5% and 1% for power [34]. Power gating circuitry of cache banks and cache ways, used in proposed work, would need negligible circuit overheads, similar to other existing approaches [65]. The cache usage statistics can be maintained by the cache controllers.

5.5 Conclusion

The DiCeR with DAM dynamically shuts down the least accessed cache banks and attempts to shutdown ways from the remaining active banks. Way shutdown reduces the associativity and can degrade system performance by increasing the miss rate. However, if we are able to use the ways from other sets within the bank, then the turned off ways do not affect performance. This is achieved by dynamically increasing sets' associativity. The proposed policy gives $33-35\%$ gains in EDP and saves static power around $68-70\%$ for 8-way associative cache having a size of 4MB. The policy shows better gains for higher associative caches, as they provide more opportunity for way shutdown, with more fine tuning of the cache sizes. All these gains are obtained by staying within the IPC degradation threshold. In case the IPC degrades beyond threshold, one can resort to selectively turn on the cache-ways. Such a policy demonstrated 40% static energy savings while maintaining IPC. The proposed policy was also shown to perform better than BSP and drowsy cache [1], which keeps cache ways in low-power mode.

Both of our energy efficient policies discussed so far in Chapter 4 and 5 outperform Drowsy cache [1] with significant amount of saving in leakage of the LLC. The policy proposed in this chapter has an extra area overhead due to CMP-SVR (less than 10%), where DiCeR proposed in previous chapter has negligible area overhead. Additionally, complexity related to the attachment of power gating circuitry is also a bit higher in this current work than the earlier one; as in this work, cache is resized at multiple granularity levels. But, the area overhead for power gating circuitry in both of these are considered as negligible [65]. However, shutting down of a larger cache bank more effectively reduces power consumption than its way-shutdown counterparts.

As power consumption plays the pivotal role in excogitation of on-chip thermal issues, hence, reduction in power consumption is the most effective optimisation knob for temperature reduction. Moreover, larger LLCs, that draw significant leakage power, occupy a major area on the wafer real estate. Hence, gating a large chunk of these LLC can potentially reduce the chip temperature by reducing

leakage power and by introducing on-chip thermal buffers. Bank shutdown, in this regard, can be a promising option towards achieving such goals, as shutting down of (larger) banks create (larger) thermal buffers on-chip with remarkably high savings in leakage. On the other hand, way shutdown can also reduce the chip temperature, but, it can create local hotspots at the smaller number of turned on ways in the CMP caches, in case of heavy accesses. Therefore, towards controlling chip temperature, we have chosen the bank shutdown approach for our next couple of contributions.

Chapter 6

DiCeR at LLC: Towards Controlling Temperature in TCMP

Dynamic Thermal Management (DTM) has become a major concern for the chip-designers, as it becomes a challenging task in recent power densed high performance CMPs, due to integration of more on-chip components to meet ever increasing demand of processing power. This increased chip temperature incorporates circuit errors along with significant increment in leakage power consumption. The popular DTM techniques apply DVFS or task migration to reduce core temperature, as cores are considered as the hottest on-chip components. But, large on-chip LLCs attached in modern CMPs are the principal contributors to the on-chip leakage power consumption and occupy the largest on-chip area, out of which a major portions may be unused. As power consumption reduction plays the pivotal role for temperature reduction, therefore, this work exploits DiCeR to create on-chip thermal buffers for reducing average chip temperature without disturbing the computation. Cache resizing decisions are taken based upon the generated cache hotspots and/or access patterns during the process execution.

6.1 Introduction

Traditional DTM methods are the effective techniques for handling thermal issues in the modern CMPs [29]. Usually, the modern DTM techniques control the running behaviour of the processor cores, the hottest on-chip elements, either by: (a) dynamically regulating its supply voltage and frequency, i.e. DVFS or (b) by migrating a hotter core's task-set to a colder one [29], called as task migration. DVFS scales down the voltage/frequency settings (V/F settings) of the running cores and can be performed either in a coarse grained manner or in a fine grained manner. On the other hand, task migration is an effective technique for reducing peak temperature of the chip, and also has wider applications for lowering energy consumption in modern heterogeneous multi-core systems.

DVFS controls voltage and runtime frequency of the processor cores for adjusting heat dissipation of the on-chip circuitry with significant savings in energy consumption. However, regulating the cores' voltage and slowing down its frequency degrades computing performance. A plethora of measures have been taken earlier to address this issue [146, 109, 111, 106]. Task migration, on the other hand, gathers information from thermal sensors' to decide migration of tasks from a hotter zone to a colder on-chip area. Migration of tasks may stall the system for a while, which degrades system performance, although it has been taken care well in some state-of-the-art policies [112, 147].

In addition with the above discussed DTM policies, larger on-chip LLCs can also be used to control the chip temperature while maintaining performance within a certain limit. On chip LLCs are comparatively colder on-chip elements, hence, are often neglected in prior DTM techniques [29]. However, a recent survey shows a spatial temperature variance of 30K in a modern CMP cache [32] (built in 45nm or lesser technology). This spatial difference in temperature clearly claims that, hotspots can also be generated in the cache area. Moreover, available literature is strengthening the claim by illustrating that on-chip caches incur significant leakage power consumption, which in turn increases chip temperature. Shrinking cache size by shutting down some portions of cache can reduce cache leakage and can

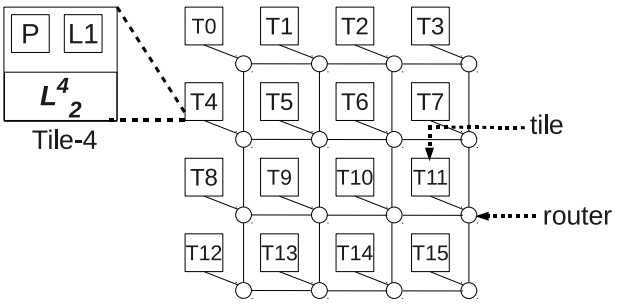


FIGURE 6.1: Tiled CMP architecture

mitigate cache hotspot problem. Furthermore, the powered off cache portions generate on-chip thermal buffer which helps in reducing temperature of adjacent on-chip components. Thereby, it reduces chip temperature without affecting the processor cores.

This work analyses the role of LLCs to control chip temperature in a TCMP architecture (shown in Figure 6.1), by dynamically shutting down certain on-chip LLC portions (while maintaining performance). Towards this, we need to identify the parts of LLC that can be powered-off, as we did in DiCeR. For this, we analyse cache usages and access patterns to make the decision. From our experimental analysis and literature survey, it has been noticed that, larger LLCs are non-uniformly accessed during the execution of an application. By considering non-uniform cache access patterns and locality of reference, cache bank shutdown towards improving thermal efficiency can be done in the following two ways-

1. Dynamic shutdown of heavily accessed cache portions will reduce cache hotspots, reduce leakage power, and create thermal buffer on-chip.

2. Conversely, dynamic shutdown of least used cache parts will reduce cache leakage and will also create on-chip thermal buffer with less performance impact.

The major contributions of this work can be summarised as follows:

1. **Proposal 1 [HB]** The heavily accessed (cache hotspots) L2 banks in a TCMP (as shown in Figure 6.1) will be turned off and contents of these

turned off banks will be remapped to some other colder cache banks, called as target bank. In the latter phase of the execution, if performance degrades beyond a threshold value or the colder target bank becomes a hotter one, the turned off bank will be turned on.

2. **Proposal 2 [CB]** Lightly used cache banks will be turned off after remapping their contents to other moderately used nearby banks. If the performance degrades more than a threshold value, then the turned off banks will be turned on.

3. The turned off cache portions will not only reduce the leakage power consumption, but also create thermal buffers to control chip temperature.

4. Both of these policies are compared with a greedy DVFS technique [2] that reduces chip temperature by applying per core DVFS when the core's temperature violates a predefined threshold value.

6.2 Thermal issues: from LLC perspective

The existing diversities in physical configuration along with the diverse set of applications generate non-uniform power consumption profile of the CMP elements. This uneven power distribution further produces enough spatial difference in temperature of the chip. In our TCMP architecture, a particular tile comprises of a core along with its local/private caches and a chunk of shared L2 as LLC (ref. Figure 6.1). Out of these elements, core's dynamic power is very high that makes it the hottest component among all, whereas comparatively colder L2 bank consumes highest leakage with the largest area occupancy.

6.2.1 LLC: Thermal Characteristics

According to the experimental statistics, thermal potential of large LLC is comparable with other on-chip components. Figure 6.2 shows the change in peak

temperature of LLC for our baseline architecture while executing 4 long running applications. The peak temperature reaches to 75°C or more for the larger caches. The uneven access patterns for the caches show non-uniform thermal status, i.e., heavily accessed cache parts are normally heated more due to their high local power density. Thus large LLCs in modern CMPs have enough potential to become hotspots (Figure 6.2). The raised temperatures will further increase the leakage and thus form a circular dependency between leakage power and temperature (which we have already discussed earlier).

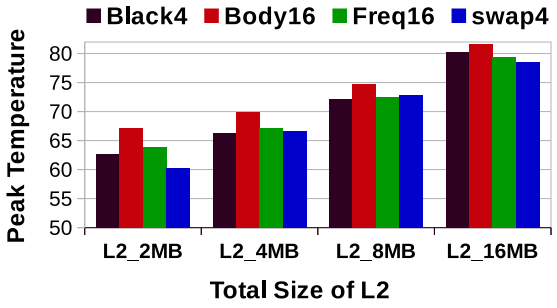


FIGURE 6.2: Peak Temperature of Caches in °C.

6.2.2 Thermal Management: Core vs Cache

The survey given in [29] shows that both DVFS and Task Migration techniques are the promising options to reduce peak and average temperature of the chip, but they may suffer from high system overhead. Migrating tasks from a hotter core to a colder one incurs idle clock cycles. Whereas DVFS increases the execution time by slowing down cores, which may be prone to violate the overall EDP budget. Moreover, task migration is only possible in those scenarios, where number of cores are more than the number of threads currently being executed. In modern era of multi-tasking environment, finding out such situation may be difficult. However, cache based thermal management, on the other hand, may increase memory stalls. But, reducing size of larger caches of modern CMPs may not be expensive enough always. Hence, from a thermal perspective, cache based methods have the following benefits to offer:

1. Core performance is not affected directly.

2. Powered-off cache banks reduce their own as well as their host tile's temperature.

3. Creation of thermal buffer through turned off banks reduces temperature in its vicinity.

4. Can maintain lower average chip temperature for a longer time duration.

These properties motivated us to build up the proposed cache based thermal efficient technique.

6.2.3 Modeling Tile Temperature in TCMP

Dynamic temperature of a tile is driven by the following three factors: (a) the component's own power consumption, (b) heat abduction by the ambient and (c) heat exchange among the peer components, which can be modeled as [33]:

$$\begin{aligned} T_i(t) = T_i(t-1) &+ f_{gen}(P_{dyn}(t) + P_{st}(t)) \\ &- f_{rem}(T_i(t-1) - T_a) \\ &+ \sum_{m=1}^{T_p} f_{tr}(T_i(t-1) - T_m(t-1)) \end{aligned} \tag{6.1}$$

where, at time t, temperature of ith tile is $T_i(t)$. $T_i(t-1)$ is the temperature of the tile i at time $t-1$. $f_{gen}(P_{dyn}(t) + P_{st}(t))$ denotes the generated temperature due to its power consumption. $f_{rem}(T_i(t-1) - T_a)$ is the temperature change due to heat removal by the ambient (where T_a is the current ambient temperature), the most effective way of cooling. Finally, $f_{tr}(T_i(t-1) - T_m(t-1))$ implies the temperature change due to heat transfer among the peer tiles (T_p), which obeys the principle of *superposition and reciprocity* [33].

In case of a powered off cache bank, the power consumption for the bank i.e. $f_{gen}(P_{dyn}(t) + P_{st}(t))$ in equation 6.1 will become zero. Therefore, heat abduction

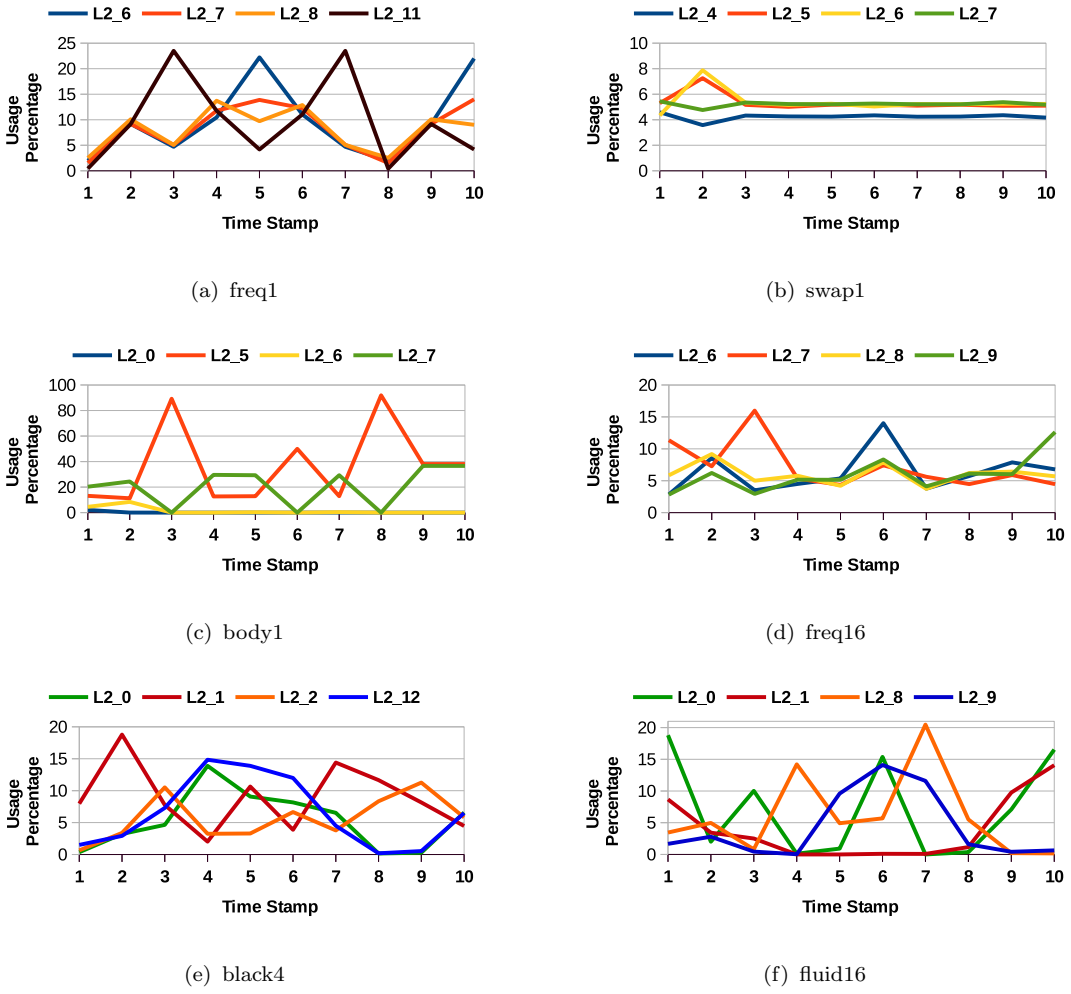
by the ambient will be active with the heat conduction from peers. As, power consumption plays the pivotal role in temperature increment, hence, our hypothesis is that, a powered-off bank will be cooled down eventually, and heat flow will take place to it from its hot neighbours, which will reduce the peers' temperatures. Hence, initially in **HB**, we attempt to reduce cache hotspots by shutting down some heavily used banks to reduce leakage, whereas in **CB**, we turned off the least used banks to significantly reduce leakage power, which will further reduce the effective chip temperature. In both the cases, turned off banks create thermal buffers, and will be able to reduce the chip temperature.

6.3 Performance Linked Thermal Management with DiCeR

As we have seen earlier, the runtime cache accesses across the banks are unevenly distributed in our multi-banked cache architecture. Generally, cache memory exploits *Locality of Reference*, which anticipates the unchanged cache access pattern for an application in future. But, in case of modern multitasking environment, this phenomena is observed for a short interval. Whereas for a large interval (more than $2-5$ million clock cycles for a standard CMP) the principle of *Locality of Reference* is violated (ref. Figure 6.3), hence, we adopt DiCeR towards performance linked temperature control for a TCMP. The performance and request redirection issues for DiCeR can be summarised by the following couple of equations (ref. Section 5.2):

$$MC_i(b) = \sum_{j=1}^{b}(h_j^i.d_j^i + a_o^{ij}.d_o) + \sum_{k=1}^{B-b} r_k^i.n_t^{ik}, b \leq B. \tag{6.2}$$

and



(a) freq1 (b) swap1

(c) body1 (d) freq16

(e) black4 (f) fluid16

FIGURE 6.3: Temporal Change in Bank Usages for 6 different PARSEC applications.

$$
\begin{aligned}
IPC(b) &= \frac{1}{N}\sum_{i=1}^{N} IPC_i(b) \\
&= \frac{1}{N}\sum_{i=1}^{N}\left(\frac{IC_i}{CC_i + MC_i(b)}\right) \\
&= \frac{1}{N}\sum_{i=1}^{N}\left(\frac{IC_i}{CC_i + \sum_{j=1}^{b}(h_j^i.d_j^i + a_o^{ij}.d_o) + \sum_{k=1}^{B-b} r_k^i.n_t^{ik}}\right)
\end{aligned}
\tag{6.3}
$$

The notations written in Equations 6.2 and 6.3 have similar meaning as it is described in Section 5.2.

6.3.1 Target Bank Selection

According to the equation 6.2, target bank selection during cache resizing is to be done by considering two key parameters-(a) present workload of victim bank, and (b) distance between target and victim banks. Exploitation of *Locality of Reference* in addition with shutting down of victim imply that, value of r_k in equation 6.2 is directly proportional to the usage of victim in recent past. Hence, if a heavily used bank is gated, it is most likely to happen that, value of r_k will be higher enough in near future. On the other hand, r_k will experience a smaller value for the least used victims. Remapping of requests will incur more cycles in case the distance between the target and victim i.e. value of n_t^k is high enough. Since, both r_k and n_t^k are positive integers, hence, large values for both, (even for atleast one) will aggravate system performance drastically. So, it is better to turn off banks by assigning target in its close proximity. From thermal perspective, it can be stated that, the heavily used banks cannot be selected as a target bank to avoid further generation of cache hotspots.

6.3.2 Reconfiguring the LLC

The following equation shows that, the consumed leakage power of the SRAM cells has a direct dependency on the running temperature and supply voltage [28, 23]:

$$P_s = K_1 V_{DD} T^2 e^{(\alpha V_{DD}+\beta)/T))} + K_2 e^{(\gamma V_{DD}+\delta)} \tag{6.4}$$

The notations used in Equation 6.4 represent the same parameters/constants as it was discussed in Chapter 1 (ref. Equation 1.3). However, as leakage power has a direct dependency on the supply voltage (Equation 6.4), hence, gating the power supply to the SRAM cells [65] can reduce the leakage consumption. Power gating technique has been used here as a backbone of this energy efficient cache. The data management during reconfiguration is done by migration of data blocks from victim to the target like DiCeR (ref. Chapter 4). This policy also does not support

multiple target banks for a particular bank in order to keep the remapping logic simple and it also uses *Reuse Counter* [18] for LRU selection, if replacement is needed at the target location.

On the other hand, system architecture needs a mechanism to control the power gating circuitry when to activate or deactivate it. Our cache resizing policy can be implemented as a part of cache controller just like our earlier policies. The policy monitors both performance degradation and temperature values while controlling power gating circuitry. Cache access pattern needs to be monitored also while shutting down or turning on the cache banks. The policy details are described later with detailed analysis.

ALGORITHM 5: [**HB**] Reduction in temperature at cache hotspot

1: T_{normal} : Interval that does not allow reconfiguration attempts
2: T : Reconfiguration interval
3: δ : Permissible percentage degradation in IPC
4: m : Maximum limit on bank shutdown
5: $j = 0$: Number of banks turned off. Initially zero.
6: D : Distance threshold, used for target bank selection.
7: Run system for T_{normal} number of clock cycles.
8: **while** $(j \leq m)$ **do**
9: Find out the hottest LLC bank.
10: Compute degradation in IPC compared to original average IPC.
11: **if** hottest bank (B_h) is a non-target bank and IPC degradation $< \delta$ and $j < m$ **then**
12: Select coldest bank (say B_t) as target having NoC $hop\_distance \leq D$ from the hottest bank (say B_h) has to be turned off.
13: Migrate contents of B_h to B_t, and enable remapping.
14: Turn off B_h.
15: j + +.
16: **else if** (hottest bank is a target or IPC degradation $\geq \delta$) and $j \geq 1$ **then**
17: Find out the least recently turned off source bank (say B_h) of this target bank (say B_t).
18: Turn on B_h and migrate remapped blocks of B_h from B_t.
19: Disable remapping at the source, B_h.
20: j - -.
21: **end if**
22: Run for the next T number of clock cycles.
23: **end while**

6.3.3 Algorithmic Design

HB: As leakage power increases quadratically with the temperature (equation 6.4), it is hence better to turn off a "hot" bank and distribute its blocks across a few colder banks, called as target banks. Eventually, it will cool down and later even if it is turned-on, the leakage will be in control (for a while). However, turning off a hot bank and distribution of its blocks across the other colder ones will consume some dynamic energy but, reduction in static energy will compensate this. NoC latency and energy values will also be increased slightly due to remapping of future requests to the target banks.

Algorithm 5 shows the detailed process of HB, which reduces temperature at cache hotspot. Initially, the process runs for T_{normal} clock cycles, after which the system will start monitoring temperatures of different on-chip components. This is a very first time-span during which system starts dissipating heat, hence, no cache reconfiguration takes place. However, at the end of first T_{normal} clock cycles, system does the following operations (line 8 to 23) as follows and loops back after end of each T clock cycles:
System initially finds out the hottest cache bank (line 9) and checks the IPC degradation over the last period [T clock cycles] (line 10). If performance degradation is within a predefined limit (δ), and the hottest bank is not a target bank, the system will turn off the hottest bank after migrating its contents to the target and enable remapping at turned off bank's controller (line 11 to 15). System also maintains the maximum permissible limit of turned off banks. On the other hand, if a target bank is found as a hottest bank or IPC degrades more than δ (line 16), system turns on a powered off bank after relocation of all the remapped data and disables the remapping (line 17 to 20).

CB, the other end of the spectrum, turns off the banks having low temperature profile and maps its contents to another colder bank. This saves the energy of the turned-off bank(s) and its usage was minimal, hence, the target bank is not overloaded, implying lesser temperature increment of the target. Also, turning off least used banks hardly affects performance.

ALGORITHM 6: [**CB**] Reduction in temperature using cache as thermal buffer

1: T_{normal} : Interval, that does not allow reconfiguration
2: T : Reconfiguration interval
3: δ : Permissible percentage degradation in IPC
4: m : Maximum limit on bank shutdown
5: $j = 0$: Number of banks turned off. Initially zero.
6: Run system for T_{normal} number of clock cycles.
7: **while** $(j \leq m)$ **do**
8: Select minimum used cache bank, B_i, as a victim bank.
9: Select another bank, B_t as the target bank, that is closest to B_i and has average usage.
10: Compute degradation in IPC compared to original average IPC.
11: **if** degradation is lesser than δ and $j < m$ **then**
12: Select another moderately accessed bank, B_t as the target bank.
13: Shutdown B_i after enabling remapping to B_t.
14: j + +
15: **else**
16: Turn on most recently turned off bank.
17: Migrate all the blocks of original bank from its target and disable remapping.
18: j - -
19: **end if**
20: Run system for T number of clock cycles.
21: **end while**

Policy CB, described in **Algorithm 6**, starts resizing the cache dynamically after T_{normal} clock cycles (line 7 to 21). From this point onwards the application runs for T clock cycles (line 20) and at the end of which it resizes the cache as follows: The minimum accessed bank B_i will be sorted out and an average accessed bank, B_t will be fixed as target, which is closer to B_i (line 8 to 9). If the IPC degradation over the last period [T clock cycles] is more than the predefined threshold value, δ, the system turns off B_i after transferring all of its contents to B_t and enables remapping at B_i (line 10 to 14). If the IPC degradation violates the constraint at the completion of a period, the system turns off the most recently turned off bank after relocation of its contents (line 16 to 18). At the end of turning on process, system disables the remapping.

Note that, both HB and CB incur same storage overhead like in DiCeR.

Components	Parameters
No. of Tiles	16
Processor	UltraSPARCIII+
Flit Size	16 bytes
Buffer Size	4
#Virtual Networks	5
L1 I/D Cache	64KB, 4-way
L2 Cache bank	512KB, 8-way
Memory bank	1GB, 4KB/page
Pipeline Stage	5-stage
VCs per Virtual Network	4

TABLE 6.1: System and Network Parameters

Cache Parameters	Values	Core Parameters	Values
Cache Level	L2	Clock rate	2400MHz
Size of a L2 Bank	512KB	ALU per core	2
Block Size	64 Bytes	FPU per core	1
Technology used	32nm	MUL per core	1
Associativity	8	Ambient temperature	47°C
Cache Model	NUCA		

TABLE 6.2: McPAT and HotSpot Configurations

6.4 Experimental Analysis

For our hardware platform, we use a 16 core TCMP setup as shown in Figure 6.1. Each of the core tiles consists of an UltraSPARCIII in-order core, designed in 32nm technology. The cores are homogeneous in nature and composed by several units- an Instruction Fetch Unit (IFU), a Load Store Unit (LSU), a private L1 Data and Instruction cache. The reconfiguration interval T for both HB and CB is taken as 5M clock cycles with IPC degradation threshold δ which is set to maximum of 5%. The reduction in cache size to lesser than its 25% degrades performance more than 5% [5]. Hence, out of 16 L2-banks, $m = 10$ and $m = 12$ banks are allowed to shutdown in case of HB and CB, respectively. As in HB, heavily used banks are turned off, hence, too much remapped traffic can increase AMAT, so we allowed $m = 10$ for HB and $m = 12$ for CB. The 8MB L2 cache used in this simulation framework has an associativity of 8 ways. Table 6.1 contains configuration details of processor cores, memory and NoC which are used in our simulation.

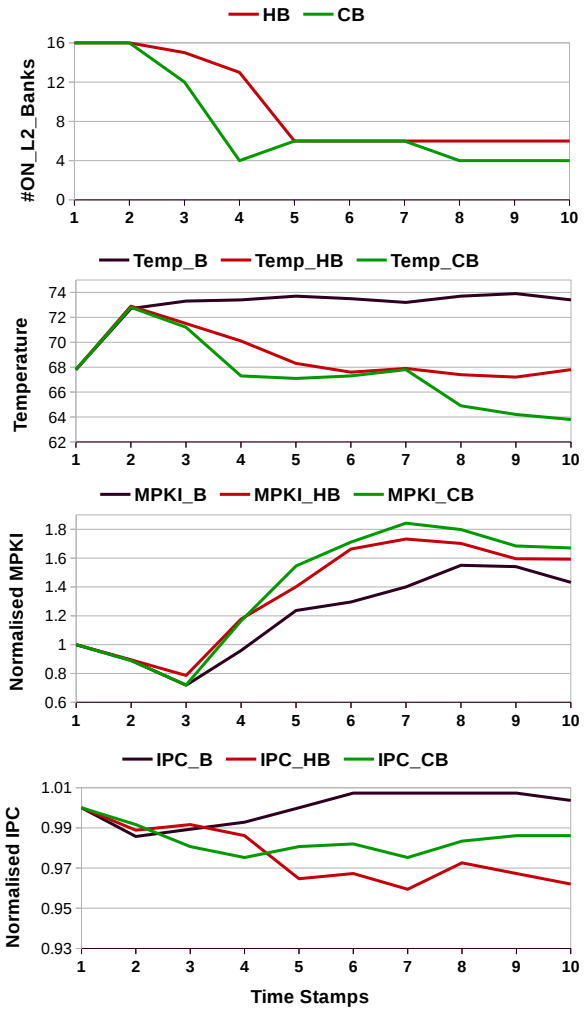


FIGURE 6.4: Temporal effect of cache resizing: body16.

6.4.1 Simulation Setup

We have used our closed loop simulation framework described in Appendix A, which is a collection of GEMS [126], SIMICS [125], McPAT [21] and HotSpot 6.0 [3]. Table 6.2 contains the configuration parameter details used by McPAT and HotSpot 6.0 in our simulation. Multithreaded PARSEC benchmark suite having emerging applications [6] are used to validate the proposed architecture.

6.4.2 Temporal effect of cache resizing

Both of our policies dynamically resize the cache to reduce leakage consumption and temperature. Figure 6.4 shows the (discrete) temporal effect of cache resizing

while applying both HB and CB independently, for Body16. To avoid redundancy, results for Body16 is only shown here. Change in the number of active banks for both HB and CB are shown in the top-most graph in Figure 6.4 at 10 consecutive time-stamps taken at a fixed interval of 10M ruby cycles during execution. As the execution progresses with resizing, the turned-off cache area are eventually cooled down. HB shuts down cache hotspots but the number of turned-off banks are lesser than CB. Temperature decrement is more in CB; but HB still has a noticeable temperature reduction (as shown in second graph from the top in Figure 6.4 with respect to baseline (Temp_B)). However, shrinking in cache size increases MPKI in both the cases, but CB incurs more conflict and capacity misses as, it shrinks cache more than HB. The third graph from the top in Figure 6.4 shows the temporal change in MPKI.

Effect on NoC, IPC and target temperature: The dynamic cache resizing further increases system overhead by incurring migration and remapping of cache blocks at victim and turned-off banks, respectively. The shutting down of heavily used banks in HB increases this overhead by invoking more remap requests, whereas for CB, powering off lightly used banks slightly increases the same. However, this overhead increases NoC latency and the NoC energy consumption, which is more in HB than CB. We show the final change in NoC latency and NoC energy consumption for all of our applications in Figure 6.7 and 6.8, respectively. This increased NoC overhead degrades IPC more in HB than CB, which is shown in Figure 6.9 for all the applications whereas, temporal change in IPC for Body16 is shown in the bottom most graph of Figure 6.4. On an average, 11.38% and 3.2% overall increment in NoC energy have been observed for HB and CB, respectively, with 4.62% and 3.06% overall increment in respective NoC workload than baseline architecture. However, this NoC energy values include migration, remapping and the energy consumption during off-chip accesses. The average performance degradation for HB and CB are 5.0% and 1.1%, respectively.

Moreover, increased workloads further increases temperature at target locations. For Body16 in HB, bank 12, a moderately used bank, has become a target. Remapping of some heavily used bank(s) to bank 12 increases its workload and hence, its

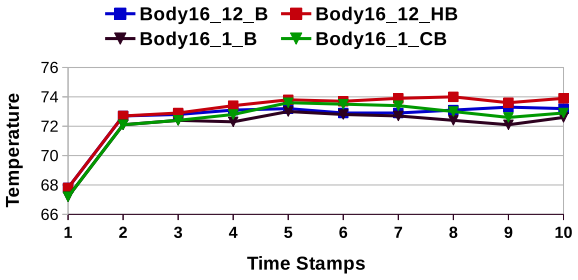


FIGURE 6.5: Thermal status of target banks in HB and CB, for Body16.

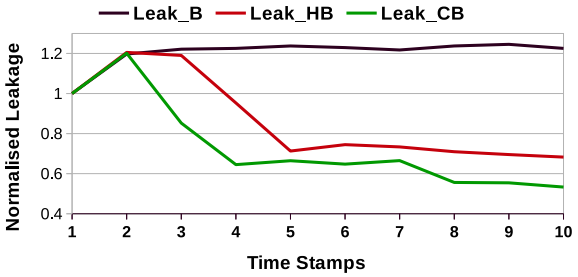


FIGURE 6.6: Change in cache leakage in HB and CB, for Body16.

temperature is raised up. On the other hand, bank 1 is a target in CB for Body16, which handles extra loads of some lightly used bank(s). Hence, the increment in its temperature with respect to baseline is a bit lower than bank 12 in HB. The graphs are shown in Figure 6.5.

Finally, all of these changes in temperature have positive effect on cache leakage consumption. Figure 6.6 shows the temporal change in cache leakage for Body16 while applying HB and CB individually. Leakage consumption is significantly reduced in both the cases. More turned off locations in CB gives more savings than HB, which only turns off cache hotspots. Both policies have good leakage savings, which are shown in Figure 6.10 for all of our applications. On average, HB saves 42% in leakage, whereas CB has a savings of 48%. Note that, the time frames shown in Figures 6.4, 6.5, 6.6 are the same.

6.4.3 EDP gain

Figure 6.11 shows the EDP gains for HB and CB compared to the baseline. The average EDP gains are 21% and 29% over the baseline architecture, for HB and

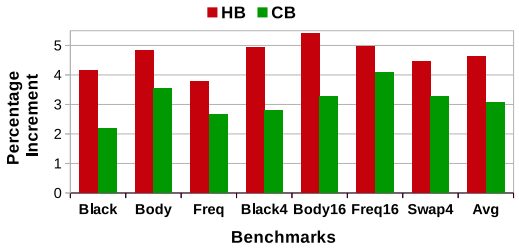


FIGURE 6.7: Increment in NoC latency than Baseline.

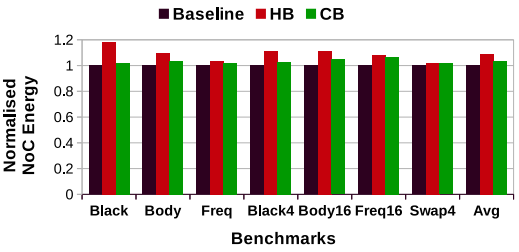


FIGURE 6.8: Increment in NoC Energy than Baseline.

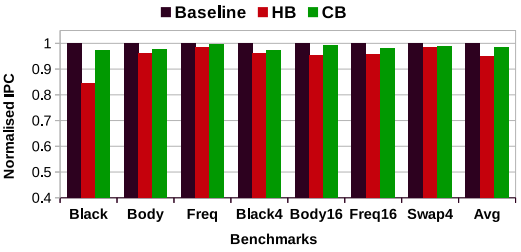


FIGURE 6.9: Change in IPC with respect to baseline.

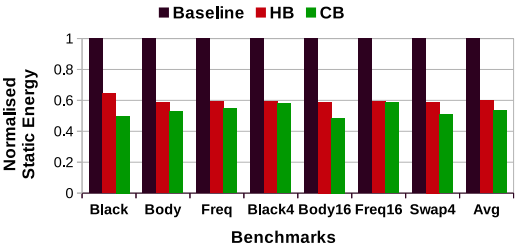


FIGURE 6.10: Reduction in leakage energy for HB and CB than baseline.

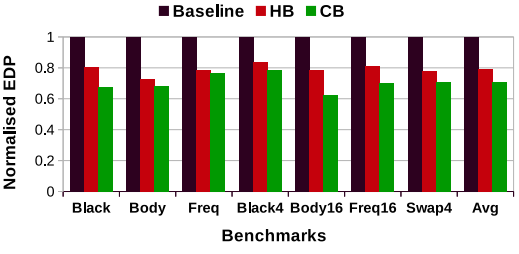


FIGURE 6.11: EDP gain for HB and CB over baseline.

CB, respectively. Increment in NoC energy and more performance degradation in HB, reduce its EDP gain over CB. Still both policies have considerable EDP gains. The EDP includes all of the above mentioned energy consumption during and after cache resizing including the overhead of migration.

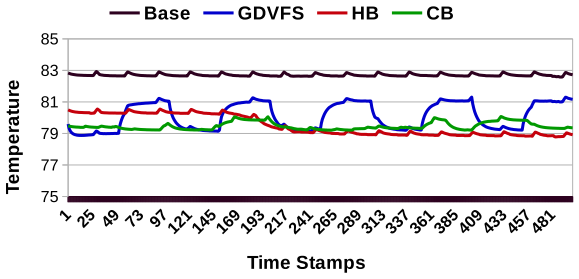
6.4.4 Effect on Tile and Chip Thermal Profile

6.4.4.1 Temporal Variation

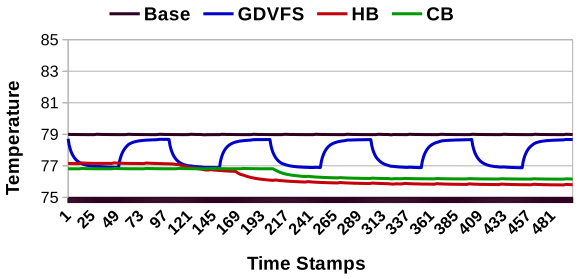
Figure 6.12 shows the temporal change in tile temperature of Tile 7 and 10 for Body16 & Freq, respectively, while applying HB & CB independently. We further compare our policies with Greedy DVFS (GDVFS) [2], a core based DTM technique. GDVFS scales down a core's V/F settings (at 1.5v, 1.8GHz) when its temperature goes beyond the preset value (95°C in our work) and scales it up (at 1.65v, 2.4GHz [baseline values]) to enhance performance, once temperature is below the threshold. GDVFS reduces tile temperature by 4°C atmost whereas HB & CB reduce it by 3.7°C and 4.1°C, respectively for Body16. In case of Freq, these reductions are 2.7°C by GDVFS, and 3.4°C & 3°C for HB & CB, respectively. The memory intensive application, Freq, does not allow much cache shrinking, hence, reduction is lesser than Body16. Figure 6.13 & 6.14 show the change in peak & average chip temperature for Body16 & Freq, respectively. GDVFS reduces peak temperature by 5°C for both the applications, but lesser reduction of 2.8°C & 2.2°C are noticed for average chip temperature for Body16 & Freq, respectively. HB & CB, on the other hand, reduce peak temperature by 3.4°C & 3.7°C for Body16 and 2.1°C & 2.3°C for Freq, respectively. But both HB & CB reduce average chip temperature in the range of $3 - 4$°C for both the applications and perform comparable to GDVFS.

6.4.4.2 Thermal Stability

To ensure circuit reliability, a stable thermal profile is to be maintained in a CMP. Table 6.3 and 6.4 show the standard deviations for thermal changes in

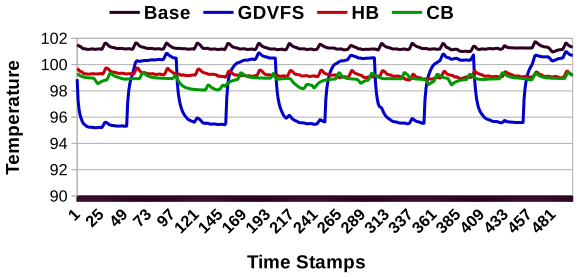


(a) Tile 7 for Body16 (best case)

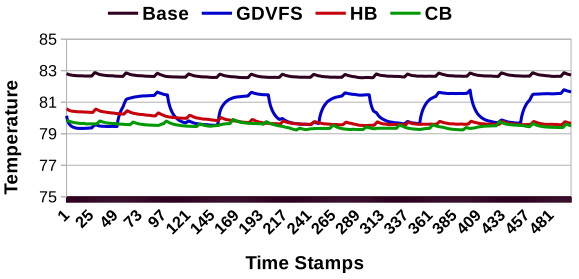


(b) Tile 10 for Freq (worst case)

FIGURE 6.12: Temporal Thermal variation of a tile.

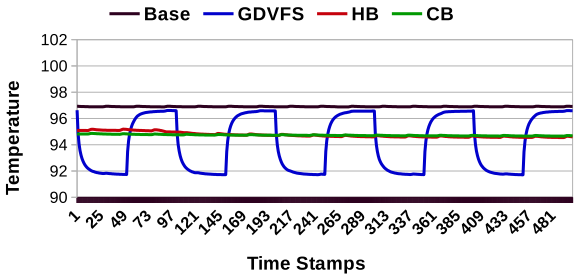


(a) Peak Temperature

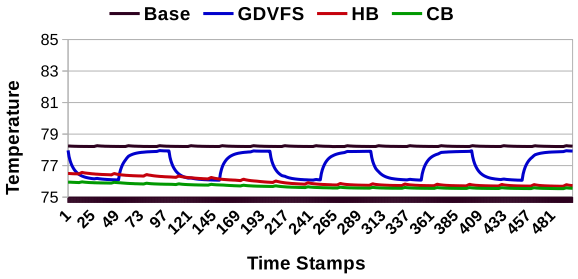


(b) Average Temperature

FIGURE 6.13: Temporal change in Chip Temperature of Body16. (best case)



(a) Peak Temperature



(b) Average Temperature

FIGURE 6.14: Temporal Change in Chip Temperature of Freq. (worst case)

Applications	Baseline	Greedy DVFS	HB	CB
Black	0.05	0.79	0.63	0.28
Body	0.11	0.83	0.62	0.19
Freq	0.14	0.77	0.69	0.22
Black4	0.06	0.80	0.64	0.28
Body16	0.07	0.78	0.68	0.24
Freq16	0.06	0.77	0.61	0.20
Swap4	0.16	0.72	0.58	0.27
Gmean	**0.08**	**0.78**	**0.63**	**0.24**

TABLE 6.3: Standard deviation for temporal changes in Average Chip Temperature.

peak and average chip temperature for the set of applications we used here. The highest standard deviation values for average temperature are 0.69 for HB, and 0.28 for CB; these respective values are 0.45 and 0.21 in case of peak temperature. Shutting down of heavily used banks degrades more performance in case of HB, hence, more dynamic cache resizing (by turn-ON-OFF) takes place, which changes thermal profile more frequently than CB. However, this change is less frequent than GDVFS. Due to more frequent changes in V/F settings, GDVFS maintains a standard deviation in the range of 0.72 to 0.83 for average temperature, and

Applications	Baseline	Greedy DVFS	HB	CB
Black	0.14	1.92	0.29	0.21
Body	0.25	1.85	0.30	0.16
Freq	0.11	2.17	0.45	0.09
Black4	0.15	2.01	0.28	0.21
Body16	0.13	2.29	0.26	0.21
Freq16	0.07	2.17	0.31	0.16
Swap4	0.24	2.03	0.30	0.18
Gmean	**0.14**	**2.06**	**0.31**	**0.17**

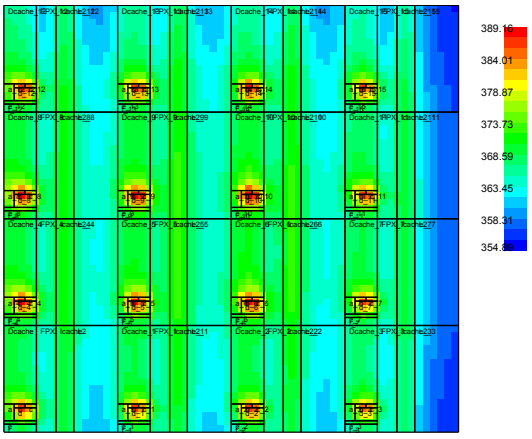
TABLE 6.4: Standard deviation for temporal changes in Peak Temperature of the Chip.

1.85 to 2.29 for peak temperature. These values indicate a comparable thermal stability of HB and CB with GDVFS.

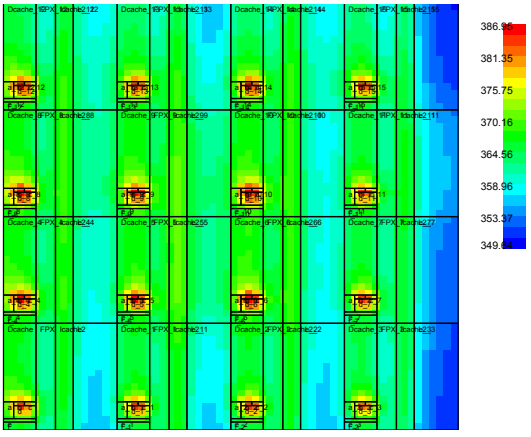
6.4.4.3 Spatial Variation

The existing diversities in power consumption across the on-chip components build up the diverse spatial thermal profile of the chip. Figure 6.15 shows the spatial thermal status of the chip for baseline, HB and CB, generated from Hotspot 6.0 [3], while running Body16 application. In comparison with baseline, both HB and CB show certain changes, when banks are turned off. The legends in the figures show the decrement in peak and lowest temperature. Tiles experiencing a bank shutdown have lesser temperature, whereas target banks' temperature are raised up slightly.

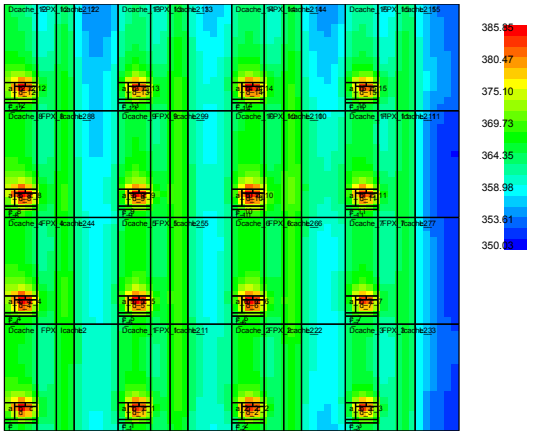
Furthermore, we show the individual tile temperature at some certain point when independently HB, CB and GDVFS are applied. Figure 6.16 shows the change in tile temperature for Body16 while applying the corresponding thermal management technique. The grey coloured box in this figure for HB and CB indicate that, the tile has a turned off bank and hence, trivially having a noticeable reduction in temperature. GDVFS, on the other hand also reduces temperature in the range of $3 - 4$°C, which is around 4.4°C at most in case of CB. The maximum temperature reduction for HB is around 4.1°C. These values claim that, our cache based



(a) Baseline



(b) HB



(c) CB

FIGURE 6.15: Spatial Thermal variation of the Chip for Body16.

policies play a crucial role in reducing average temperature of a tile comparable with the GDVFS policy.

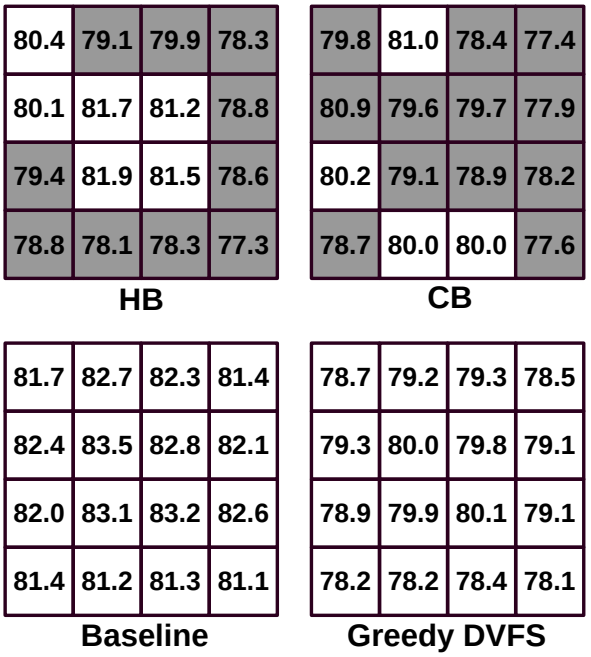


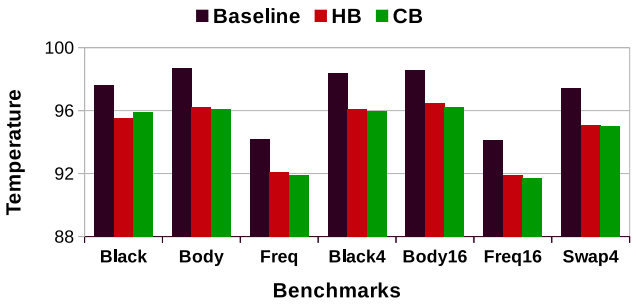
FIGURE 6.16: Change in Average Tile Temperature for Body16.

6.4.4.4 Scalability

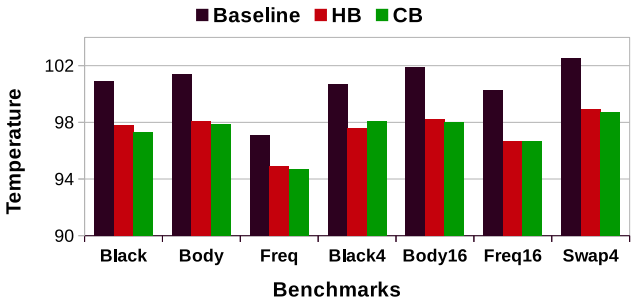
To show the scalability of our proposal, we have simulated our idea for a smaller sized 4MB L2 and a larger sized 16MB L2 cache. The size of thermal buffer varies across the total cache size in terms of following:

1. Too much reduction in cache size will affect performance drastically.
2. Smaller sized banks create smaller thermal buffer, having lesser potential for temperature reduction.

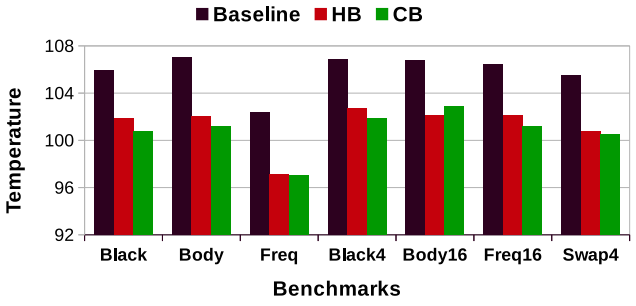
Our simulation results further justify the couple of facts stated above. For a 4MB cache, peak and average temperature are reduced by around 2°C; for 8MB these respective reductions are around 3.5°C and 3.1°C for average and peak temperature in HB and CB. The maximum reduction is shown in case of 16MB cache, which are around 6°C for average and 5°C for peak temperature of the chip, for both HB and CB, respectively. From performance degradation perspective, average IPC degradations for HB and CB are 6.3% and 1.8%, respectively. For 8MB



(a) 4MB



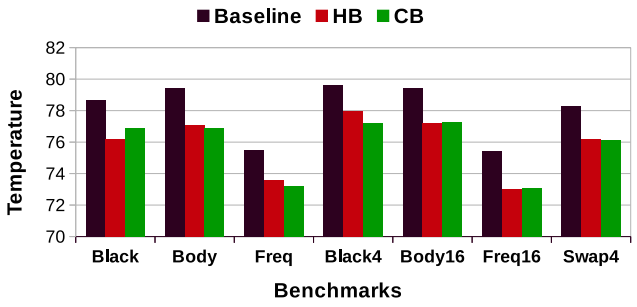
(b) 8MB



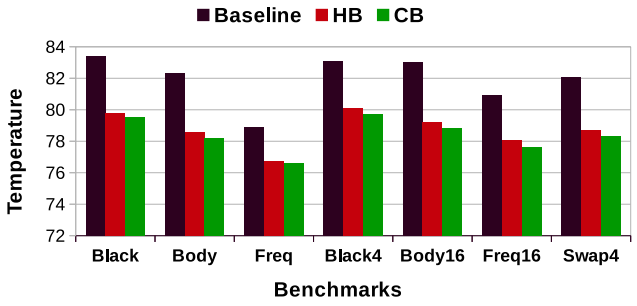
(c) 16MB

FIGURE 6.17: Reduction in Peak Temperature for different cache sizes.

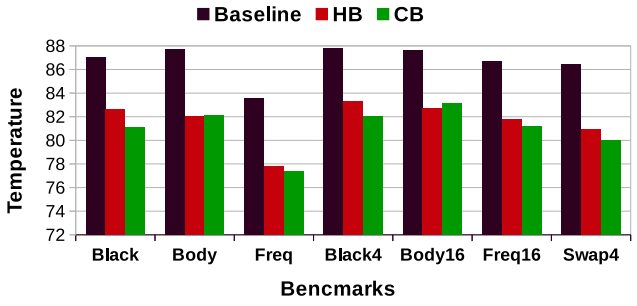
and 16MB caches, these values are lesser than 5% and 4% for HB and negligible for CB. Hence, it can be concluded that larger on-chip caches not only enhance performance, but also can play significant role in on-chip thermal management. Reduction in peak and average temperature for 4MB, 8MB and 16MB are shown in Figures 6.17 and 6.18, respectively.



(a) 4MB



(b) 8MB

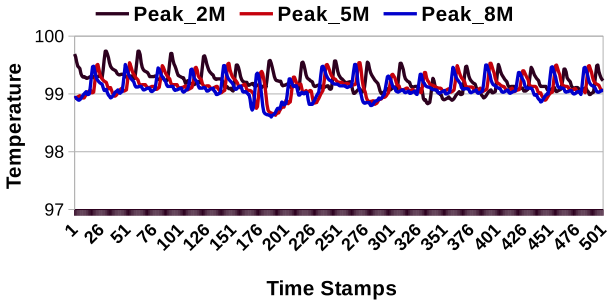


(c) 16MB

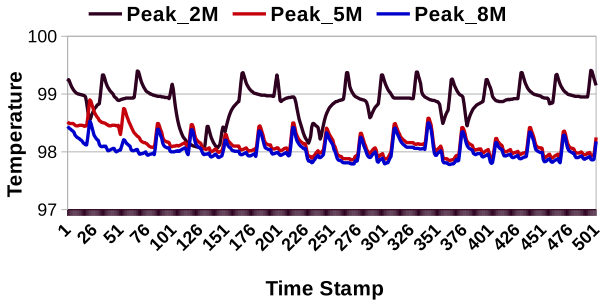
FIGURE 6.18: Reduction in Average Chip Temperature for different cache sizes.

6.4.5 Varying Reconfiguration Interval

We further experimented with due changes in reconfiguration interval (T in Algo. 5 & 6). So far, all of our simulations are performed with 5M span of reconfiguration interval, which we have changed to 2M and 8M intervals. Figures 6.19 and 6.20 depict the temporal changes in peak and average temperature of the chip for Body16 in case of both HB and CB. The graphs in all of these cases clearly show that, fluctuation in chip temperature is more in smaller reconfiguration interval, which has been controlled better for higher values like 5M and 8M. As both 5M

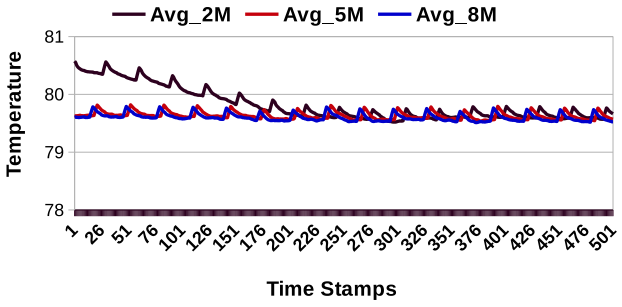


(a) HB

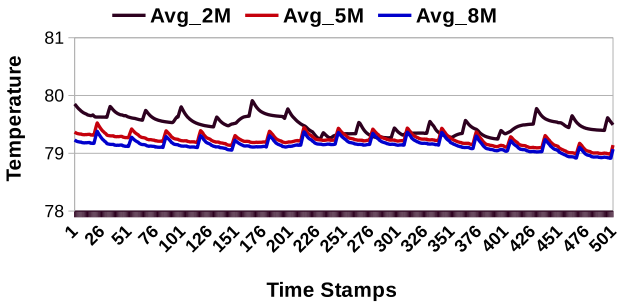


(b) CB

FIGURE 6.19: Temporal Change in Peak Temperature of Body16 while varying Reconfiguration Interval.



(a) HB



(b) CB

FIGURE 6.20: Temporal Change in Average Temperature of Body16 while varying Reconfiguration Interval.

and 8M are showing the same nature of the curves, but, 8M degrades system performance more than smaller ones, hence, 5M is chosen during our experimental evaluation. Actually, for smaller value of T, the reconfiguration overhead will be higher when application's WSS changes abruptly (like Body). But for CPU intensive applications (like Swap4), thermal efficiency is almost same across the values of T. Further, for memory intensive applications (like Freq), smaller values of T incurs high reconfiguration overhead, whereas larger values of T tend to degrade performance as we have to use a fixed lower cache size for a long interval, even when it is in high demand. From both thermal as well as performance perspective, moderate values like 5M, however, shows (sub)-optimal value for all applications we used here.

6.4.6 Summary

Table 6.5, 6.6 and 6.7 report the average reduction in peak and average chip temperature across the applications with HB and CB along with the corresponding degradation in performance, while using 4MB, 8MB and 16MB LLCs. We compare our policies with GDVFS by applying it to all of our above mentioned LLC sizes. As larger caches actively participate in increasing chip temperature (Figure 6.2), hence, cache based policies are showing more benefits with larger caches. On the other hand, GDVFS reduces core's temperature which in turn reduces peak temperature of the chip. But larger area occupancy of LLCs with higher temperature reduce this effect gained from DVFS. Furthermore, due to availability of adequate cache space even after resizing, performance degradation is lesser with larger caches while applying our policies. Therefore, for the CMPs having larger caches, cache based thermal management policies can give more temperature reduction with a controlled performance degradation.

cache banks (not the cache ways like [117]), which is completely done during execution unlike [64].

GDVFS		**HB**		**CB**	
Temperature Reduction (in °C)					
Peak	Average	Peak	Average	Peak	Average
5.2	3.6	2.3	2.2	2.6	2.5
Performance Degradation					
6.1%		6.3%		1.8%	

Table 6.5: 4MB L2 Cache

GDVFS		**HB**		**CB**	
Temperature Reduction (in °C)					
Peak	Average	Peak	Average	Peak	Average
5.0	3.3	3.2	3.1	3.4	3.7
Performance Degradation					
6.2%		4.8%		1.2%	

Table 6.6: 8MB L2 Cache

GDVFS		**HB**		**CB**	
Temperature Reduction (in °C)					
Peak	Average	Peak	Average	Peak	Average
4.7	3.1	4.6	5.1	5.0	5.7
Performance Degradation					
5.6%		3.7%		1.1%	

Table 6.7: 16MB L2 Cache

6.5 Conclusion

LLCs occupy large on-chip area in modern CMPs with a significant amount of leakage power consumption which has a quadratic relationship with chip temperature. In order to control chip temperature, our proposed policy dynamically resizes on-chip LLC to reduce leakage power consumption and to create on-chip thermal buffer which retards the temperature increment of the remaining powered on portions.

Our proposed technique consists of two cache resizing policies: HB, which turns off heavily accessed cache hotspots to cool them down. If performance degrades beyond a threshold, it turns on these cache area later. The other policy, CB, shuts down lightly used cache portions which consume significant leakage energy. The cache turn on decision will be taken once the performance degrades. Both

of our policies partially turn off the cache and create on-chip thermal buffer with significant leakage reduction of 41% and 49%, respectively. The creation of thermal buffer along with the leakage power reduction reduces effective temperature of the tiles in our TCMP model.

Shutting down heavily accessed cache banks in HB reduces temperature of the tiles by 3.6°C. CB, which turns off lightly accessed banks, reduces tile temperature by 4°C. Policy CB shuts down least accessed banks and keeps them turned off for a long duration as performance degradation is negligible. Hence, more reduction in leakage energy is noticed and additionally thermal buffer is active for a long time-span. Therefore, temperature reduction is more in CB while maintaining the performance. Although the first policy HB has a performance degradation of 5%, still both are useful if chip temperature is a priority constraint to control.

Chapter 7

DiCeR in a CCMP Towards Improving Thermal Efficiency

In our previous chapter, we have proposed our cache based thermal efficient technique for TCMP architecture. Basically, the physical structure of a TCMP offers us the following issues while applying DiCeR based technique to reduce its effective temperature:

- The cores/LLC banks located at the circumference of the chip has trivially better cooling opportunity than the others.
- Some set of cores are sandwiched between two L2 banks. These kind of cores are highly benefited when both adjacent banks are turned-off. Conversely, if both banks are heavily heated, then core temperature is likely to be hiked.
- Additionally, a shutdown cache bank may also be sandwiched between two cores. In such cases, cooling rate is very slow for these banks even when they are turned off. Hence, in case they are powered on in future, there is a high probability that, they will resume their operation at some high temperature, which may increase the leakage consumption drastically.

7.1 Introduction

To see the better effectiveness of our DiCeR in temperature control, we have taken a CCMP, where cores are always located at the periphery. Centrally located large LLC area implies that, the banks can never be sandwiched between a couple of cores. Hence, cooling rate for the cache banks are more, especially when they are gated. Furthermore, cores will be benefited more in CCMP if their adjacent banks are turned off. So, this will help to reduce both peak as well as average chip temperature.

This chapter analyses the role of a centralised multi-banked SNUCA LLC (as shown in Figure 7.1) in thermal management while maintaining system performance. By exploiting DiCeR, we dynamically resize the LLC to optimally balance the performance and chip temperature by offering two levels of thermal management-(i) controlling cache temperature, and (ii) reducing temperature of the global hotspots by governing on-chip conductive heat transfer. The turned off cache banks are eventually cooled down, and create on-chip thermal buffers which reduce temperature of the adjacent on-chip components (other cache banks and/or cores). The major contributions of this work can be summarised as follows:

1. Considering performance as a system-wide constraint, we have developed an analytical model for our architecture to determine the optimal cache size. As in this architecture, we have more number of banks, hence, to strengthen the limit of maximum number of turned off banks, we use both analytical and simulation modeling.

2. The analytically determined optimal cache size is used for resizing LLC by following the three thermal efficient patterns-

    - **AltRow.** Shuts down alternate rows of cache banks.
    - **Chess.** Generates a chessboard or checkerboard like pattern in LLC, with two colours. One of the colour will represent prospective shutdown candidates.

- **OptTar.** Cache banks closer to the cores are assigned the highest shutdown priority. Additionally, future requests of the turned off cache banks are optimally handled, as more cache portions are turned off than earlier methods.

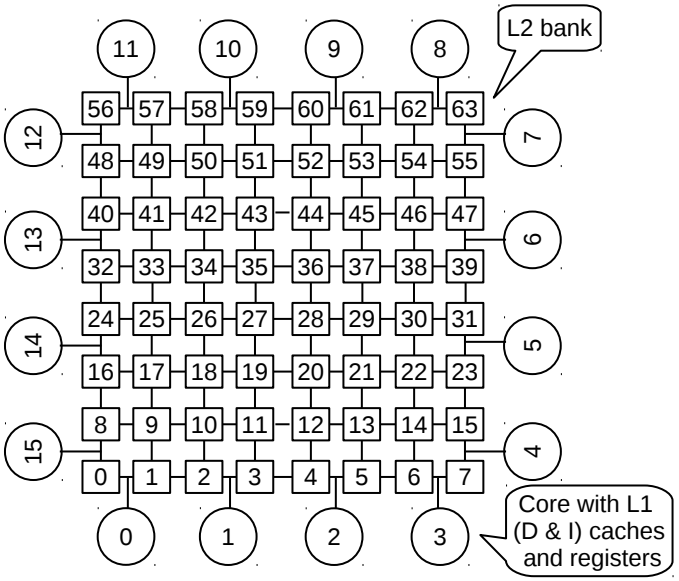


FIGURE 7.1: CMP architecture with Centralised LLC (CCMP).

In case of caches, heavily used blocks can generate cache hotspots, whereas least used cache portions unnecessarily increase the leakage consumption contributing to the chip temperature. Furthermore, in case of some modern applications, cache access patterns do not conform to the classical cache access property, (the *Locality of Reference*), in their long run. These existing diversities in cache access behaviour across the applications show the necessity for dynamic cache resizing.

7.2 Background

In this section, we first discuss the role of a centralised LLC towards the chip temperature and power consumption.

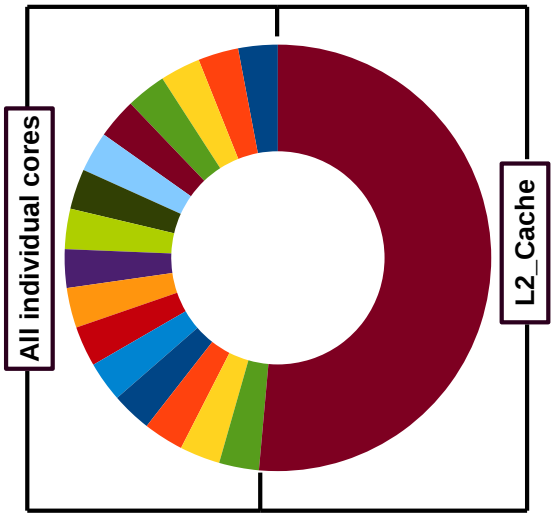


FIGURE 7.2: Percentage contribution for all the on-chip components collected by simulating our baseline CCMP architecture (ref. Figure 7.1) in McPAT having 16 cores (from UltraSPARCIII family) and 64-banked 8MB L2 cache as centralised LLC.

7.2.1 CCMP and its Leakage Hungry LLC

The baseline architecture, used in this work (shown in Figure 7.1), contains 16 homogeneous CPU cores, which are placed along the periphery of the chip. Each circle in the figure, numbered from 0 to 15, represents a single core along with its private Data and Instruction L1 caches. The centrally located shared L2 cache (on-chip LLC in our case) is sliced set-wise into identical 64 banks, numbered from 0 to 63, called as centralised shared cache. This set-wise division implies that, all ways of a set are present in the same bank. Note that, we are using Static NUCA (SNUCA) cache access scheme, where a block is always placed at a fixed cache set on its every allocation. A 2D mesh Network on Chip (NoC) connects all the L2 banks and the Cores.

For our baseline architecture (Figure 7.1), the total power consumption of the chip can be divided into two major components (apart from the interconnect and I/O interface): (I) power consumed by the individual cores that includes processing power along with power consumed by L1 (data & instruction) caches; and (II) power consumed by the LLC (L2 in our case). Figure 7.2 depicts power consumption of the individual on-chip components, out of which L2 consumes highest power among all. Majority of the L2 power consumption is coming from its static

(leakage) component (as shown in Figure 7.3). Independent to the cache accesses, leakage power that has a circular dependency on temperature can be reduced by gating the cache banks. Hence, reducing LLC leakage power by dynamically resizing it, can be a promising option to reduce chip temperature without affecting the computational units as we have already seen earlier.

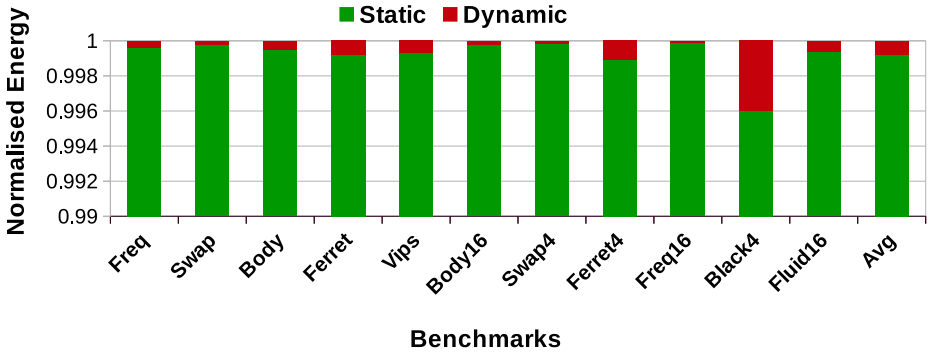


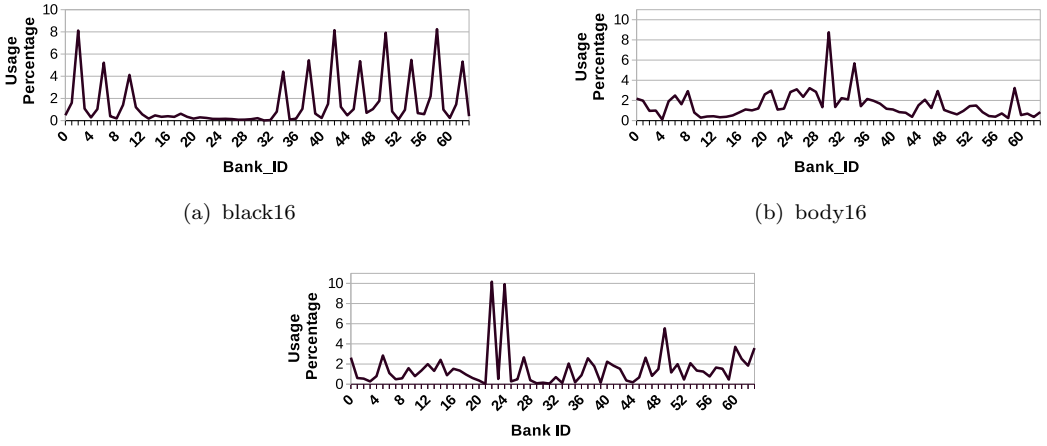
FIGURE 7.3: Distribution of Power Consumption in an 8MB L2 cache.

7.2.2 Thermal Potential of the Centralised LLCs

Table 7.1 shows the changes in peak temperature of LLCs for 4 different sizes in case of a CCMP. The values are derived for three PARSEC applications from our simulation setup discussed earlier. Even, the LLC in a CCMP also follow the same access patterns. However, this peak temperature of LLC gradually increases with its size and even reaches around 80°C for all the cases (like Figure 6.2 for an LLC in a TCMP). These values strongly indicate the existence of cache hotspots in larger LLCs having size 8MB or more, which motivates us to explore the on-chip thermal efficiency while dynamically resizing larger centralised LLCs.

Cache Size	2MB	4MB	8MB	16MB
black16	58.9	60.9	69.9	78.9
body16	60.1	63.4	70.3	80.1
fluid16	59.9	63.7	70.6	79.2

TABLE 7.1: Size vs LLC's Peak Temperature (in °C).



(a) black16

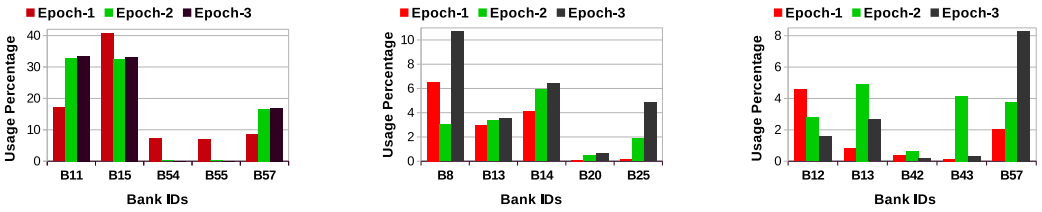
(b) body16

(c) fluid16

FIGURE 7.4: Non uniform distribution of cache bank accesses in a CCMP like Figure 7.1.

7.2.3 Runtime Cache Behaviour

The run-time cache accesses across the banks are unevenly distributed also in a multi-banked LLC of a CCMP architecture. Figure 7.4 shows the distribution of cache accesses for all 64 banks. The results are shown for three long run applications-black16, body16 and fluid16, after running them upto consecutive 100 millions of instructions in our simulation setup. Although cache accesses exploit *Locality of Reference*, but, in the case of long running applications, this property violates with respect to bank mappings. The change in access patterns over long time-span for black16, body16 and fluid16 are shown in Figure 7.5. On the other hand, according to our analysis given in Chapter 6, we have seen that, DiCeR actively participates in reducing chip temperature while maintaining performance degradation within a certain limit. Furthermore, thermal benefits gained through the cache based policy is also comparable with the traditional core based temperature reduction techniques. Hence, it can be stated that, cache based methods can assist in thermal management. They can be used either in isolation or in conjunction with core based methods. Note that, cache based methods are more effective in case of large LLCs.



(a) black16 (b) body16 (c) fluid16

FIGURE 7.5: Change in cache bank access behaviour over time in a CCMP having 64 banks.

7.3 Preliminaries and Analytical Problem Formulation

The TCMP architecture used in our earlier contributions has 16 L2 banks. Applying DiCeR through a heuristic approach to sort and select target among the 16 banks works well enough. But heuristic approach may not be a scalable one for the large scale systems. In this work, we have a CCMP with an L2 having 64 banks, where heuristic approach may aggravate the system performance. Hence, in this section, an analytical model will be introduced with a set of its required preliminaries. The notations, that will be used in this model are given in Table 7.2.

The objective of this work is to increase thermal efficiency of this CCMP architecture by exploiting DiCeR. From thermal efficiency perspective, shutting down of more cache banks will create larger thermal buffers on-chip and eventually the chip temperature will reduce. Therefore, towards enhancing thermal efficiency, gating of more number of cache banks will always be beneficial. But, drastic reduction in cache size will further curtail the performance by increasing off-chip access count. To address these issues, we have to formulate a couple of inter-dependent optimisation problems: (i) minimise chip temperature through DiCeR without violating a predefined performance constraint; (ii) what will be the maximum achievable performance for a certain cache size?

Before handling these problems, we first model the core temperature with its associated power consumptions. In the subsequent parts of this section we will further formulate these problems towards finding out an optimal cache size for a better thermal efficiency.

7.3.1 Core Temperature Modeling

The existing literature claims cores as the hottest on-chip components, but, unlike caches cores are known for their significantly high dynamic power consumption ($P_d^{Ci}(t)$) than their static counterpart ($P_s^{Ci}(t)$). As power consumption constructs the backbone of the on-chip thermal issues, hence, we initially model the core power consumptions.

The dynamic power of core Ci can be written as-

$$P_d^{Ci}(t) = \alpha.C_{Ci}.V_{Ci}^2(t).f_{Ci}(t) \tag{7.1}$$

Descriptions of the parameters/notations used in Equation 7.1 are given in Table 7.2.

The static/leakage power of Ci ($P_s^{Ci}(t)$) at time t, (ref. Equation 1.3), has a direct dependency on running temperature of the circuitry. Additionally, leakage consumption forms a circular dependency with the effective circuit temperature, which strongly indicates that, the theoretical modeling and practical assusmption of leakage as well as running temperature is complicated. Even the well established electro-thermal analogy stated in our literature [3] strengthens this claim. However, authors in [28], decoupled this circular dependency with the help of Piece-Wise Linear Approximation of a curve which has been adopted in our thermal model as follows:

The static power is modeled as-

$$P_s^{Ci}(t) = P_{s-min}^{Ci} + k_T^{Ci}.T_{Ci}(t) + k_v^{Ci}.V_{Ci}(t) \tag{7.2}$$

And the effective temperature $T_{Ci}(t)$ is approximated by-

$$T_{Ci}(t) = (P'^{Ci}_{d}.{V_{Ci}}^2(t).f_{Ci}(t) + \zeta_v^{Ci} + P'^{Ci}_{s}).R \tag{7.3}$$

where, $P'^{Ci}_{d} \triangleq \zeta_T^{Ci}.P_d^{Ci}$ and $P'^{Ci}_{s} \triangleq \zeta_T^{Ci}.P_s^{Ci}$. All notations/parameters used in Equations 7.2 and 7.3 are described in Table 7.2.

Notations	**Descriptions**
Ci	i-th core
t	Time Stamp
$P_d^{Ci}(t)$	Dynamic Power consumption of core Ci at time t
$P_s^{Ci}(t)$	Static Power consumption of core Ci at time t
α	Activity factor
C_{Ci}	Capacitance of core Ci
$V_{Ci}(t)$	Supply voltage of core Ci
$f_{Ci}(t)$	Running frequency of core Ci
k_T^{Ci}	Temperature coefficient of core Ci
k_v^{Ci}	Voltage coefficient of core Ci
P_{s-min}^{Ci}	Minimum leakage power consumption of core Ci
$T_{Ci}(t)$	Effective temperature of core Ci
R	System's constant
ζ_T^{Ci}	Temperature-leakage coefficient
ζ_v^{Ci}	Voltage-leakage coefficient

TABLE 7.2: Descriptions of the notations used in our analytical modeling.

7.3.2 Problem Formulation

This work basically focuses on reducing chip temperature by decaying/reconfiguring its on-chip LLC dynamically. Figure 7.6 shows the internal architectures of our target CCMP. The inner grey blocks numbered from 0 to 63 are indicating L2 banks. Power supply is separated to each bank. Initially, the chip temperature increases as the execution proceeds, until it reaches at its steady state, where the dissipation of the generated heat is same with the heat absorption by the attached cooling mechanism. Applying DiCeR on this system will eventually reduce the average chip temperature by introducing thermal buffers. However, more buffer space although may prohibitively intensify the thermal benefits, but, excessive reduction in cache size will disgrace the system performance.

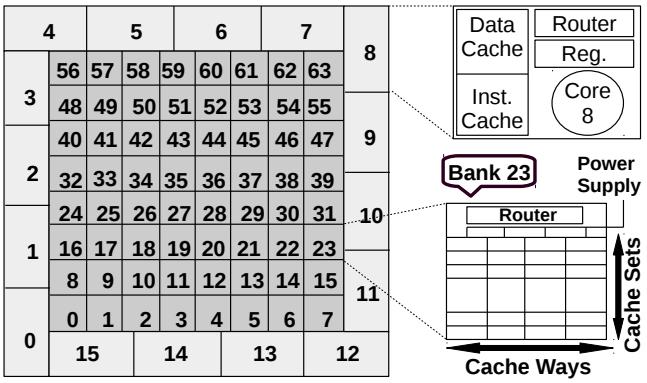


FIGURE 7.6: Internals of our baseline CMP architecture.

Our first problem stated earlier in this section can be analytically defined as follows. Minimise the average chip temperature, T_{mean}, by applying DiCeR, with a predefined performance constraint (C)-

Minimise

$$T_{mean} = Mean(T_1^c, T_2^c, ..., T_N^c, T_1^b, T_2^b, ..., T_B^b) \tag{7.4}$$

subject to,

$$IPC = \frac{1}{N}\sum_{i=1}^{N} IPC_i \geq C. \tag{7.5}$$

where, N is the number of (homogeneous) cores, and B is the total number of cache banks (of same size). T_i^c is the temperature of core i, T_j^b is the temperature of L2 bank j and system IPC should obey the given performance constraint C. Now, our next problem is to find out the optimal number of powered-on banks (b) for which our performance constraint can be met while applying DiCeR. Towards this, we assume a uniform performance constraint for the individual core i.e., $IPC_i \geq C$, where IPC_i is the IPC of core i. Keeping b as system wide shared parameter, now a performance maximisation problem can be formulated as below.

Maximise

$$IPC = \frac{1}{N}\sum_{i=1}^{N} IPC_i \tag{7.6}$$

subject to,

$$IPC_i = \frac{IC_i}{CC_i + MC_i(b)} \geq C. \tag{7.7}$$

&

$$b \leq B \tag{7.8}$$

Here, IPC is the system wide average IPC across the cores with b number of available L2 banks to satisfy the above defined constraints. IPC_i depends upon the total instruction count (IC_i) of itself, the number of cycles required to perform the computation (CC_i) at core i and the number of memory cycles ($MC_i(b)$) required to access L2 (that includes both hit and miss cycles). The DiCeR modifies b and in turn MC_i. The modification details are as follows:

$$MC_i(b) = \sum_{j=1}^{b} (h_j^i.d_j^i + a_o^{ij}.d_o), b \leq B. \tag{7.9}$$

All of these parameters used in the above equation are representing the same like Chapter 5 and 6. h_j^i is the number of hits at bank j, whose requests have been generated at core i. The d_j^i is the delay at bank j to send block to core i. a_o^{ij} implies the number of off-chip accesses due to misses at bank j which have been requested by core i, and (uniform) off-chip access latency is represented by d_o. Initially, we have $b = B$, and after bank shutdown, the target banks are introduced, which incurs a few extra (NoC) cycles for the remapping of the requests to reach at the targets. Hence, the Equation 7.9 can be rewritten as follows:

$$MC_i(b) = \sum_{j=1}^{b} (h_j^i.d_j^i + a_o^{ij}.d_o) + \sum_{k=1}^{B-b} r_k^i.n_t^{ik}, b \leq B. \tag{7.10}$$

r_k^i in Equation 7.10 is the number of remapped requests at $(B - b)$ number of turned off banks, generated by core i. Number of NoC cycles required to reach at target t (from k) is represented by n_t^{ik}. Therefore, after elaboration of $MC_i(b)$ (in Equation 7.10) while considering bank shutdown, Equation 7.6 can be modified as follows:

$$
\begin{aligned}
IPC(b) &= \frac{1}{N}\sum_{i=1}^{N} IPC_i(b) \\
&= \frac{1}{N}\sum_{i=1}^{N}\left(\frac{IC_i}{CC_i + MC_i(b)}\right) \\
&= \frac{1}{N}\sum_{i=1}^{N}\left(\frac{IC_i}{CC_i + \sum_{j=1}^{b}(h_j^i.d_j^i + a_o^{ij}.d_o) + \sum_{k=1}^{B-b} r_k^i.n_t^{ik}}\right)
\end{aligned}
\tag{7.11}
$$

This equation illustrates how IPC depends on b. Now, we will use this analysis to obtain the optimal number of cache banks (i.e. b) needed to get the desired performance.

7.3.3 Performance Modeling with Cache Size

Each core's IPC (IPC_i) is directly proportional with its own instruction count (IC_i) and inversely proportional with the total clock cycles required to execute IC_i at core i. The total number of clock cycles is the summation of its CPU cycles (CC_i) and memory latency ($MC_i(b)$). Now, considering IC_i and CC_i as constants, we can say that, IPC is a function of b, i.e. $IPC_i(b)$. According to the Equation 7.11, reduction in b will reduce the hit count ($h_j^i.d_j^i$) that indicates drastic increment both in cache misses (i.e. increment in $a_o^{ij}.d_o$) and remap cycles (i.e. $\sum_{k=1}^{B-b} r_k^i.n_t^{ik}$), which eventually will degarde the performance ($IPC(b)$), by incurring extra memory cycles. Therefore, Equation 7.6 can exhibit a nature of a concave function if we vary b, and the performance improvement at core i can then be written as-

$$
PI_i = \frac{\partial IPC_i(b)}{\partial b} \tag{7.12}
$$

Practically, the values of PI_i varies with the applications. The estimation of PI_i at t is basically derived from its value at $t-1$, i.e. from performance at the prior phase of the running process. Performance degradation sets value of PI_i as

negative. As L2 is a shared resource to all the cores, we can assume a uniform change in PI_i for all i.

$$PI_1 = PI_2 = PI_3 = ... = PI_N \tag{7.13}$$

By using Lagrange Multiplier [35], the above equation can be proved as:

$$LF = \frac{1}{N}\sum_{i=1}^{N} IPC_i(b) + \sum_{i=1}^{N} \lambda_i(IPC_i - C) \tag{7.14}$$

The first term in the RHS of above equation implies the objective whereas second term implies the constraints to be satisfied. Equation 7.14 is maximised when $(b, \lambda_1, \lambda_2, ...\lambda_N)$ is a stationary point, and first order derivative is zero [35], which mathematically can be written as-

$$\frac{\partial LF(b, \lambda_1, \lambda_2, ..., \lambda_N)}{\partial(b, \lambda_1, \lambda_2, ..., \lambda_N)} = 0 \tag{7.15}$$

This has now formed $N+1$ equations with $N+1$ unknowns, and IPC maximises when Equation 7.13 satisfies and optimal value of b can be derived.

7.3.4 Thermal Model

Before modeling temperature for our CCMP (ref. Figure 7.6) with equation 6.1 [33], we divided the whole CMP into three zones-

- the core area, for which the thermal status depends on the other adjacent core blocks and the neighbouring cache banks;
- the cache banks adjacent to the cores, where heat exchanges between the core blocks and the peer cache banks;
- other cache banks, where heat flows only among the cache banks.

Therefore, the temperature of a core block ($T_C(t)$), the bank ($T_M(t)$) adjacent to core and an inner bank ($T_I(t)$) at time t can be modeled, respectively, as-

$$T_C(t) = T_C(t-1) + f_{gen}(P_{dyn}(t) + P_{st}(t)) - f_{rem}(T_C(t-1) - T_a) + \sum_{m=1}^{p_C+p_B} f_{tr}(T_C(t-1) - T_m(t-1)) \tag{7.16}$$

$$T_M(t) = T_M(t-1) + f_{gen}(P_{dyn}(t) + P_{st}(t)) - f_{rem}(T_M(t-1) - T_a) + \sum_{m=1}^{p_C+p_B} f_{tr}(T_M(t-1) - T_m(t-1)) \tag{7.17}$$

$$T_I(t) = T_I(t-1) + f_{gen}(P_{dyn}(t) + P_{st}(t)) - f_{rem}(T_I(t-1) - T_a) + \sum_{m=1}^{p_B} f_{tr}(T_I(t-1) - T_m(t-1)) \tag{7.18}$$

The notations in the above equations have their usual meaning like Equation 6.1. p_C denotes the number of peer core blocks and p_B is the number of adjacent cache banks for the corresponding element whose temperature is being modeled. We further modify equation 7.16 by replacing f_p with the power consumption parameters for a core of Equation 7.3 as follows:

$$T_C(t) = T_C(t-1) + (P'^C_d.V^2_C(t).f_C(t) + \zeta^C_v + P'^C_s).R - f_a(T_C(t-1) - T_a) + \sum_{m=1}^{p_C+p_B} f_c(T_C(t-1) - T_m(t-1)) \tag{7.19}$$

We use the insight gained from this zone based temperature variation to decide the locations of the cache banks as candidates for shutting down. In particular, this zone based temperature variation motivates us to propose and analyse three different patterns (given in Section 7.3.7) for enhancing thermal efficiency of the CCMP.

7.3.5 Combined Analytics

Shutting down of a cache bank will make $f_{gen}(P_{dyn}(t) + P_{st}(t)) = 0$ in Equations 7.18 and 7.17. When power consumption for a block is zero, its temperature will only depend upon the ambient temperature (T_a) and the temperature of its peers. Zero power consumption over a long time-span will retard the temperature increment rate and gradually the block will cool down. More temperature difference with colder (power gated) peers will increase conductive heat transfer from the hotter powered-on block to the colder peers, and eventually temperature of the powered-on block will also reduce. Therefore, output of f_{tr} in Equations 7.16, 7.17 and 7.18 will produce smaller values and will reduce temperatures of cores along with the cache area. Hence, it can be concluded that, the mean temperature of the chip will be reduced more by gating more number of cache banks at appropriate locations.

On the other hand, drastic reduction in the cache size curtails the performance by incurring more number of cache misses. According to section 7.3.3, performance has been maximised based on the available number of cache banks for a given performance constraint. To show the effectiveness of our theoretical model we assume a CMP having a uniform execution pattern, i.e., same number of instructions are executed by each of the cores and all cache banks are assigned with uniform workload. Now, while dynamically reducing cache size our model must satisfy the Equation 7.13 for maximising performance with the available cache size. Moreover, cache size reduction through power gating increases both the miss rate and the NoC latency while accessing the target banks.

7.3.6 Finding out Optimal b

According to the equations 7.7 and 7.12, IPC of a core will be affected if the cache capacity changes. This is reflected in the component $MC_i(b)$ in the equations that represents the memory access latency. As derived in Equation 7.11, this latency

depends on the number of cache misses, the available number of cache banks, off chip access delay and delay incurred for redirected requests to target banks.

Our aim is to shutdown cache banks to control temperature while keeping IPC under a given constraint. To find one optimal value for all applications is not practically feasible. We therefore obtained the values of IPCs for all benchmark applications and did a linear regression to obtain the value for b. Results are shown in Figure 7.7. For our experiments, we have put a limit of 4% on IPC degradation and hence using results of Figure 7.7 we derive the value of b as 16 for our setup. Thus we can shutdown maximum of $(B - b)$ number of L2 banks.

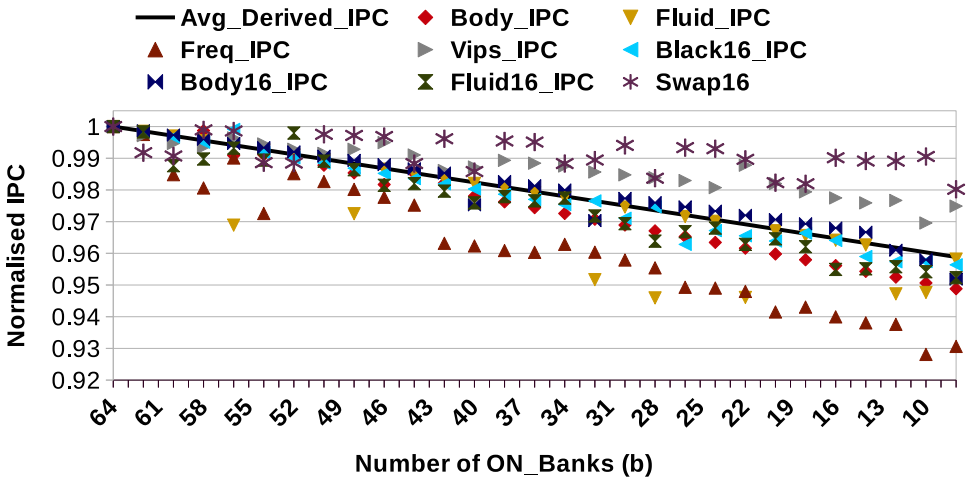
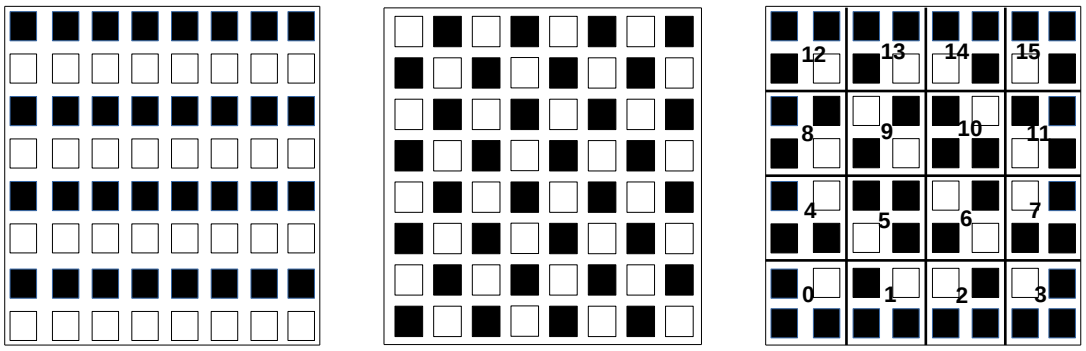


FIGURE 7.7: Relationship between IPC and Cache Size.

7.3.7 Patterns for Cache Resizing

Using the insight gained from Section 7.3.4, we have designed three patterns for cache resizing that indicate which banks are to be shutdown and their corresponding target locations. Figure 7.8 shows three patterns, named as AltRow, Chess and OptTar, respectively.

- *AltRow* (Figure 7.8(a)), the first pattern, turns off cache banks located on alternate rows. The target bank for a shutdown bank is the powered-on bank in the neighbouring row, having NoC hop-distance 1. During execution, the powered-off and target banks switch roles.



(a) AltRow (b) Chess (c) OptTar

FIGURE 7.8: L2 bank Shutdown patterns.

- *Chess* follows a pattern like a chess board (ref. Figure 7.8(b)) where black coloured banks are turned off and others are kept powered-on becoming targets of their gated peers. This pattern is also swapped like the earlier one during execution.

- *OptTar* shuts down banks adjacent to the cores (shown in black in Figure 7.8(c)) for creating more thermal buffers near the chip's hotspots. The banks are clustered into a size of 4. The clusters along the periphery have one powered-on and three gated banks in each cluster. The inner clusters can have 2-ON and 2-OFF banks as per requirements. The ON/OFF banks change roles during execution. However, banks along periphery, i.e. near to the cores are kept turned off. The bank which is ON in a cluster becomes the target for the OFF banks in that cluster.

7.4 DiCeR for Thermal Efficiency

7.4.1 Algorithms and Discussions

The implementation of the proposed policy needs to track core-wise IPCs dynamically, in addition with the current temperature, which is monitored by the on-chip thermal sensors [29] located across the chip wafer.

ALGORITHM 7: Performance Constrained and Thermal Efficient Dynamic Cache Resizing.

Input: i, r, t, b_{opt}, δ, M_c, M_p

Initialize $m_c = 0$, $m_b = 0$ and add $Bank_ID$s to M_c and M_b according to the selected pattern;
Run with the baseline system for initial duration i ;
repeat
    Call Reconf();
    Call Trans();
until *end of execution*;

Function Reconf()
repeat
    if $IPC_{deg} < \delta$ *&* $m < b_{opt}$ **then**
        if $m_p < M_p$ **then**
            Select hottest bank m_{hot} from M_p, assign Target bank m_{tar} ;
            $m_p = m_p + 1$;
        end
        else if $m_c < M_c$ **then**
            Select hottest bank m_{hot} from M_c, assign Target bank m_{tar} ;
            $m_c = m_c + 1$;
        end
        Migrate blocks from m_{hot} to m_{tar};
        Turn Off m_{hot} and enable remapping to target at m_{tar} ;
        $m = m_c + m_p$;
    end
    if $IPC_{deg} \geq \delta$ *&* $m \geq 1$ **then**
        Turn on coldest bank m_{cold} from the list of shutdown banks;
        if $m_{cold} \in M_c$ **then**
            $m_c = m_c - 1$;
        end
        if $m_{cold} \in M_p$ **then**
            $m_p = m_p - 1$;
        end
        $m = m_c + m_p$;
    end
until r;
Return;

Function Trans()
Turn on all the turned off banks;
$m_c = m_p = m = 0$;
repeat
    Run the application;
until t;
Modify the selected pattern by updating M_c & M_p, such that the roles of ON/OFF banks swap;
Return;

The practical implementation of our algorithm divides the whole execution time into several big intervals. The intervals are used either for the reconfigurations or for the transitions, as shown in Figure 7.9. The reconfiguration intervals allow

cache resizing while maintaining performance constraint, whereas transition intervals do not allow any cache resizing and runs the whole system by turning on all shutdown components. After running the application for some initial duration we collect bank usage-statistics for all the banks along with the thermal profile of the chip. It is advantageous to shutdown cache banks along the periphery, i.e. near to the cores before we choose to turn off central banks. The list of banks belonging to each type are maintained by two sets, M_p and M_c, respectively. Using the temperature values, if the number of turned-off banks has not reached its limit we select a bank from either of the lists as a candidate for shutdown. The target bank for this candidate bank is chosen as per the selected pattern (discussed in Section 7.3.7). The data blocks of the candidate are transferred to the target banks and on completion of this process the candidate is powered off. All subsequent future requests are forwarded to the target. The successive bank shutdown process continues until we reach at the maximum limit of turned off banks or the performance degrades beyond the predefined limit. In case the performance constraint is violated, the coldest among the turned off banks is selected for turning on. The remapped data belonging to this bank is relocated from its target before resuming its normal operations. This whole process continues for a long enough interval.

The target banks during the reconfiguration interval are overloaded with the workloads of the turned off banks, resulting in increment in their power density hence, the temperature raises up. Even for some memory intensive applications the hotspots can be generated at the target banks if it is being accessed heavily for a certain time quantum. Hence, in order to reduce the power density (or hotspot) we need to switch their roles with the colder turned off banks. This procedure goes through a transition interval where all the powered off banks are turned on, and cache size remains unchanged until end of the interval. With the onset of the next

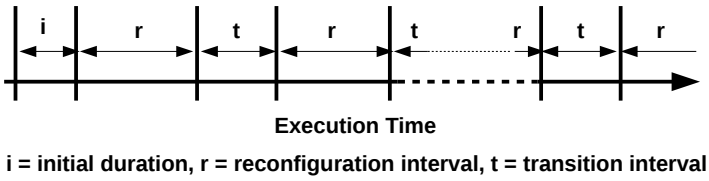


FIGURE 7.9: The division of Execution time while implementing Algorithm 7.

Parameters	Descriptions
i	Length of initial duration
t	Length of transition interval
r	Length of reconfig. interval
δ	Maximum percentage of IPC degradation
M_c	Set of central Bank_IDs to be gated
M_p	Set of peripheral Bank_IDs to be gated
m_c	Counts no. of turned-off banks from M_c
m_p	Counts no. of turned-off banks from M_p
b_{opt}	Maximum no. of banks can be turned-off
m	Counts total no. of turned-off banks

TABLE 7.3: Description of Parameters used in Algorithm 7.

reconfiguration interval, the selected pattern is updated by switching the roles of banks and starts the cache resizing process further. This process continues until the end of the process execution.

Detailed steps of the whole process are given in Algorithm 7 which divides the whole process into three parts. The parameters used in this algorithm are described in Table 7.3.

- *The master part* (line no. 1 to 5)- With the beginning of the execution, it initialises all the required parameters (listed in Table 7.3) and run the system for an initial duration of i clock cycles. While the process is running, system alternatively calls Reconf() and Trans() functions which represent two intervals r and t respectively as shown in Figure 7.9.

- *Function Reconf()* (line no. 8 to 34)- This function checks whether to power on or off the cache banks while maintaining performance constraint δ. The number of shutdown cache banks is also kept within optimal limit, b_{opt}. The algorithm tries to turn off banks from the peripheral parts (line no. 11 to 13). Once this list is exhausted it then attempts to turn off the inner central banks (line no. 15 to 17). On violation of performance degradation constraint, the coldest among the turned off banks is turned on (line no. 23 to 31). The whole process runs in each reconfiguration interval r. The cache resizing follows a particular selected pattern from the three discussed earlier.

- *Function Trans()* (line no. 36 to 43)- The gated banks are turned on at the beginning of the function. During the transition interval t, no cache resizing is allowed. On completion of t, roles are exchanged among the candidates for ON and OFF banks of the last r interval while following the selected pattern.

7.5 Experimental Evaluation

7.5.1 Simulation Setup

For our hardware platform, we use the floorplan as shown in Figure 7.6. The whole chip is divided into 80 tiles, which are of two types-(a) the 16 core tiles, located along the periphery, and (b) 64 central cache tiles. Each of the core tiles consists of an UltraSPARCIII in-order core, in 32nm technology. The core tiles are homogeneous in nature and composed by several units- an Instruction Fetch Unit (IFU), a Load Store Unit (LSU), a private L1 Data and Instruction cache. The shared L2 as on-chip LLC, is divided into 64 homogeneous banks and are distributed uniformly across the 64 cache tiles (ref. Figure 7.6). These on-chip components are connected through a 2D mesh, for which a router is equipped to each of the cache tiles as well as with the cores. Table 7.4 contains configuration

Components	**Parameters**
No. of Banks	64
Processor	UltraSPARCIII+
Flit Size	16 bytes
Buffer Size	4
#Virtual Networks	5
L1 I/D Cache	64KB, 4-way
L2 Cache bank	128KB, 8-way
Memory bank	1GB, 4KB/page
Pipeline Stage	5-stage
VCs per Virtual Network	4

TABLE 7.4: System and Network Parameters

Cache Parameters	Values	Core Parameters	Values
Cache Level	L2	Clock rate	3000MHz
Size of a L2 Bank	128KB	ALU per core	2
Block Size	64 Bytes	FPU per core	1
Technology used	32nm	MUL per core	1
Associativity	8	Ambient temperature	47°C
Cache Model	SNUCA		

TABLE 7.5: McPAT and HotSpot Configurations

details of processor cores, memory and NoC which are used in our simulation. We simulated the whole system in our closed loop simulation framework discussed in Appendix A and PARSEC benchmark suite has been used for validation (ref. Section 3.2).

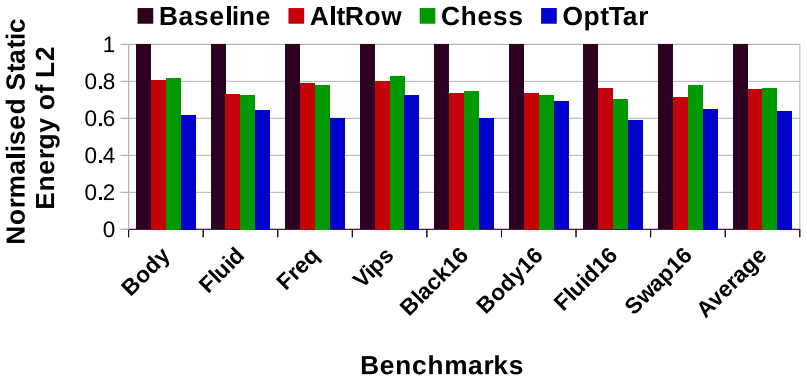


FIGURE 7.10: Static Energy savings at L2 Cache.

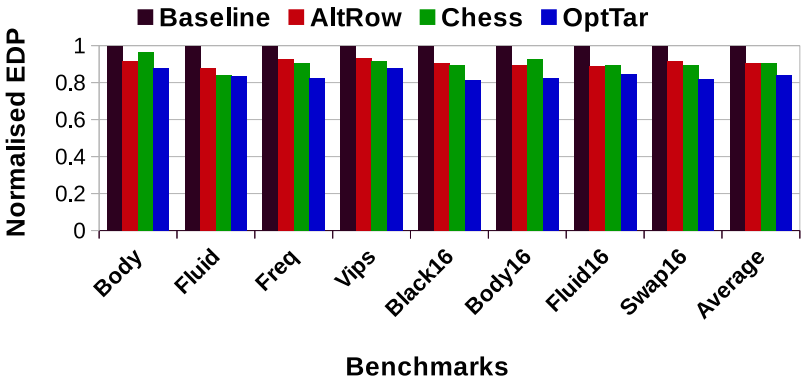


FIGURE 7.11: EDP Savings of the chip.

7.5.2 Leakage Energy and EDP Savings

As power reduction constructs the backbone of any thermal management policy, hence, we first show the leakage energy reduction while implementing the three above mentioned patterns in our closed loop simulation environment. The mix of 8 applications from PARSEC [6] are executed, and their corresponding leakage energy savings are shown in Figure 7.10. The x-axis represents the benchmarks whereas y-axis shows the leakage energy consumptions normalised to the baseline's leakage consumption. The maximum leakage savings of 40.3% are achieved for OptTar. On an average, a static energy savings of 26% and 26.5% have been achieved for AltRow and Chess, respectively.

The effect in static energy savings has sound reflection on total energy consumption of the chip, with an effect on IPC (ref. equation 7.7). Figure 7.11 shows the savings in EDP, which is derived from total energy consumption of the chip (including both cores' and cache-energy) and IPC. The average EDP savings for three patterns are 11%, 11.5% and 18.7%, respectively. The more static energy saving in the OptTar gives more EDP gains than the others.

7.5.3 DiCeR Overhead in CCMP

DiCeR in turn increases NoC overhead due to two major operations-(a) block migration during turn-off process, and (b) remapping of requests after bank shutdown in CCMP as well. Figure 7.12 shows the extra clock cycles required for migration. For most of the applications, this average increment is around 3% for AltRow and Chess, whereas for OptTar, it is close to 5%. However, this extra time-span keeps idle only the victim banks, those that are going to be turned-off just after completion of migration process. Whereas, the operations at the non-victims are executed normally during this period. On the other hand, the remapped blocks to the target locations increases NoC traffic. The maximum average increment in NoC traffic (for OpTar) is 4.86%, which is shown in Figure 7.13. Migration and remapping altogether further increase NoC energy consumption (Figure 7.14)

by 5.18%, (on an average across the applications), for OptTar than our baseline architecture. The distance cognizant target selection for all of our patterns incorporate a maximum hop distance of 2 in OptTar, and 1 for the rests. Hence, OptTar experiences maximum energy overhead than AltRow and Chess, however, this overhead has been compensated by the gained leakage and EDP savings.

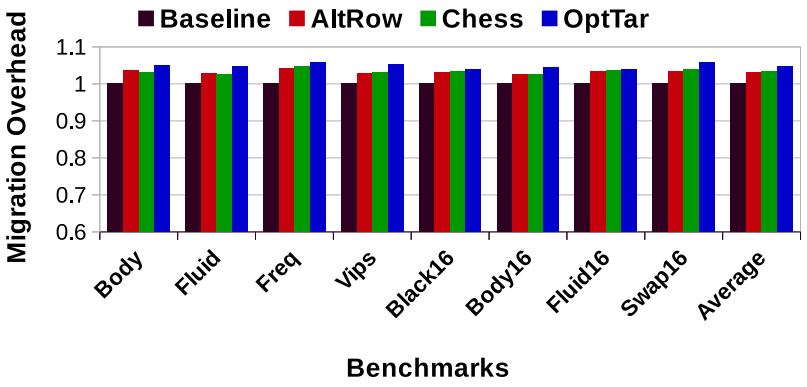


FIGURE 7.12: Migration Overhead.

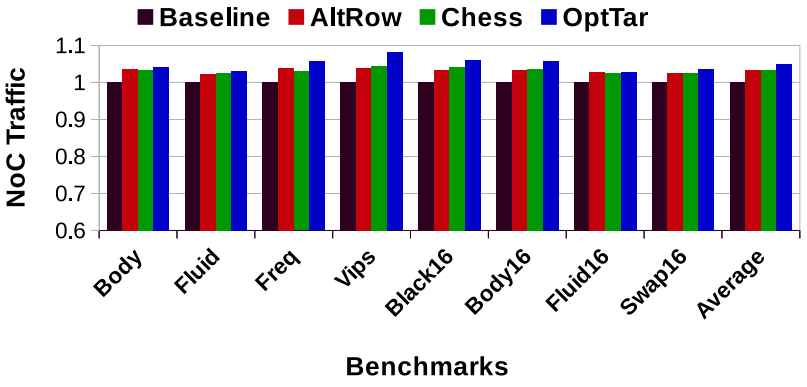


FIGURE 7.13: Increment in NoC Traffic.

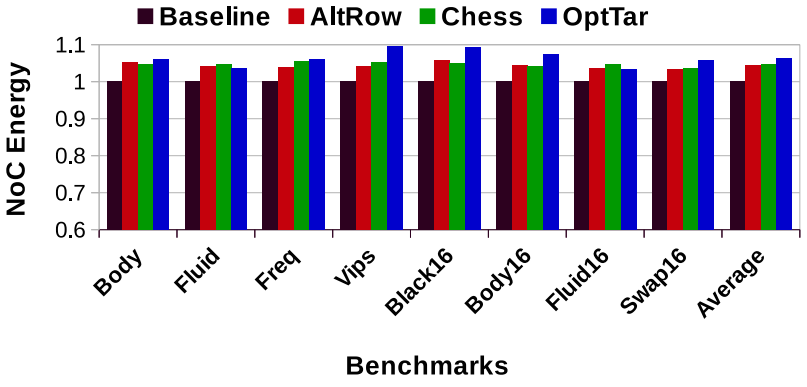
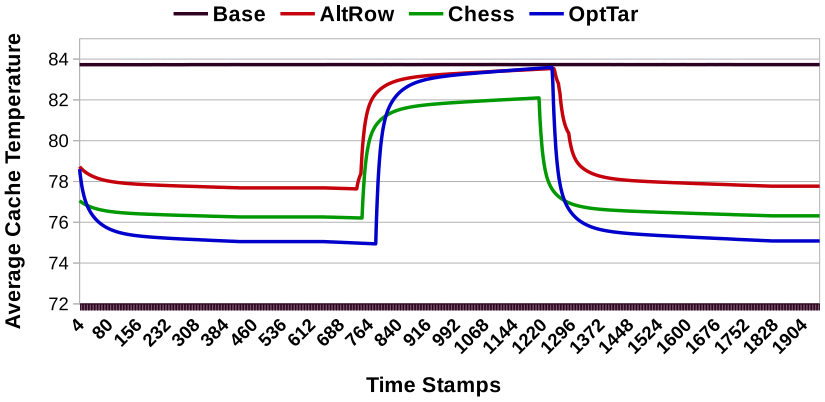
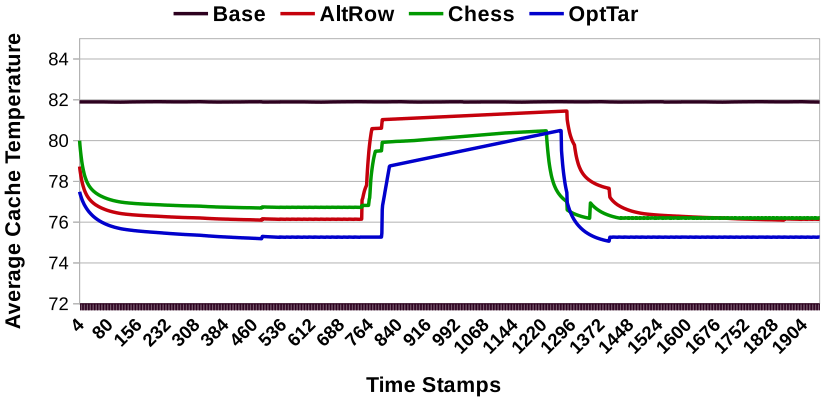


FIGURE 7.14: Increment in NoC Energy.



(a) Fluid16

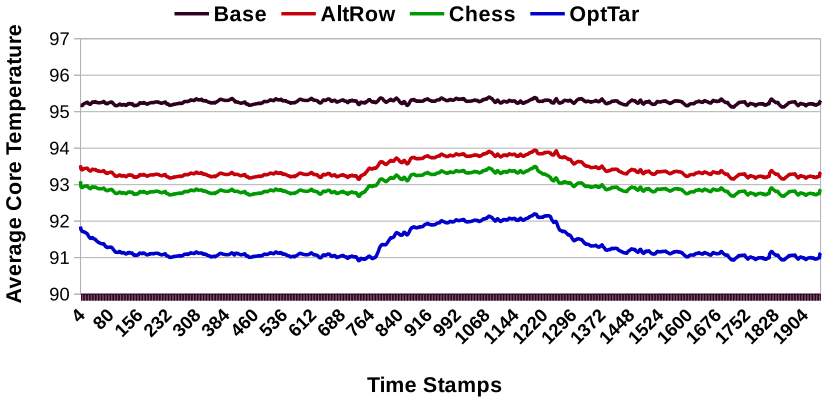


(b) Body16

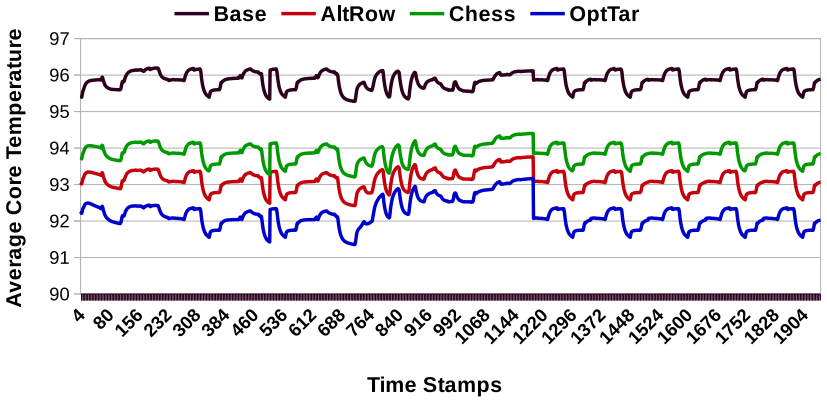
FIGURE 7.15: Snapshot of temporal changes in **average** temperature of L2 during execution.

7.5.4 On-Chip Thermal Profile

The turned off banks consume no power and eventually cooled down due to the heat abduction by the ambient (equation 6.1). Additionally, the on-chip components at vicinity of these shutdown banks will transfer heat towards this cold zone and temperature of the adjacent components will also decrease gradually. To show the effect of our proposal, we compare all three patterns for the best case (for Fluid16) and worst case (for Body16) scenarios. We show the reduction in average temperature of cache area and core blocks, separately. Figure 7.15 and 7.16 show the reduction in average temperatures for cache and core area, respectively. The x-axes in these figures represent a portion of the execution interval which includes two consecutive reconfiguration ($\boldsymbol{r}$) intervals with a transition ($\boldsymbol{t}$) in between. The



(a) Fluid16



(b) Body16

FIGURE 7.16: Snapshot of temporal changes in **average** temperature of the cores during execution.

temporal changes in both caches' and cores' temperatures are noticed due to bank shutdown in both **r** intervals as well as in **t**. Note that, the y-axis denotes the temperature values in °C.

For all the cases, OptTar reduces more temperature than the others due to gating of more and optimally located cache area. The reduction in average temperature for OptTar is around 8.3°C in case of Fluid16. For Body16, OptTar reduces the cache temperature by 5.7°C, during the same interval. Due to their adjacency to the turned off cache banks, average temperature for the core area is reduced by 4.3°C for Fluid16, whereas for Body16 the value is around 3.8°C. The core temperature for Body shows fluctuation due to temporal variation in processing

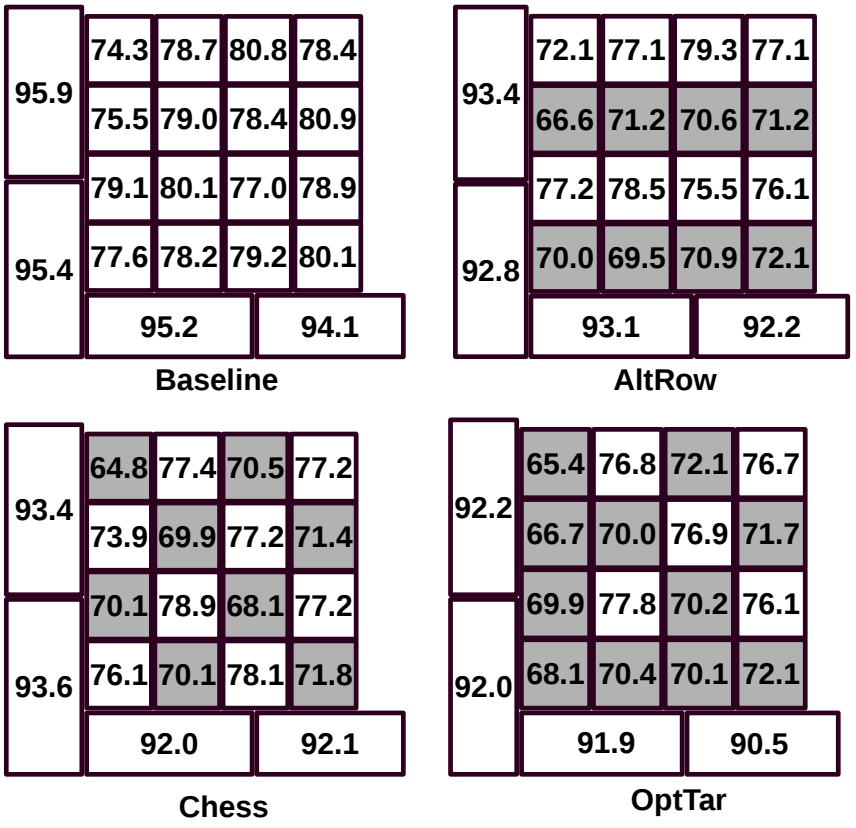


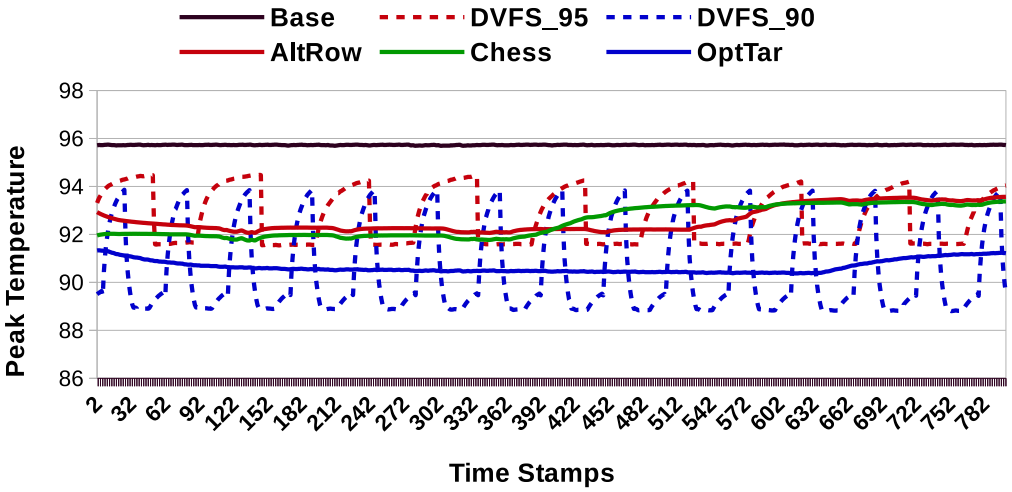
FIGURE 7.17: Temperatures (in °C) of the individual banks during $\boldsymbol{r}$, for Fluid16.

loads across the cores. Furthermore, at the beginning of **t**, the banks are turned-on, hence, the average cache temperature rises. With the beginning of the next **r** after **t**, again the temperature decreases for both L2 cache as well as for cores as the cache banks are gradually shutdown.

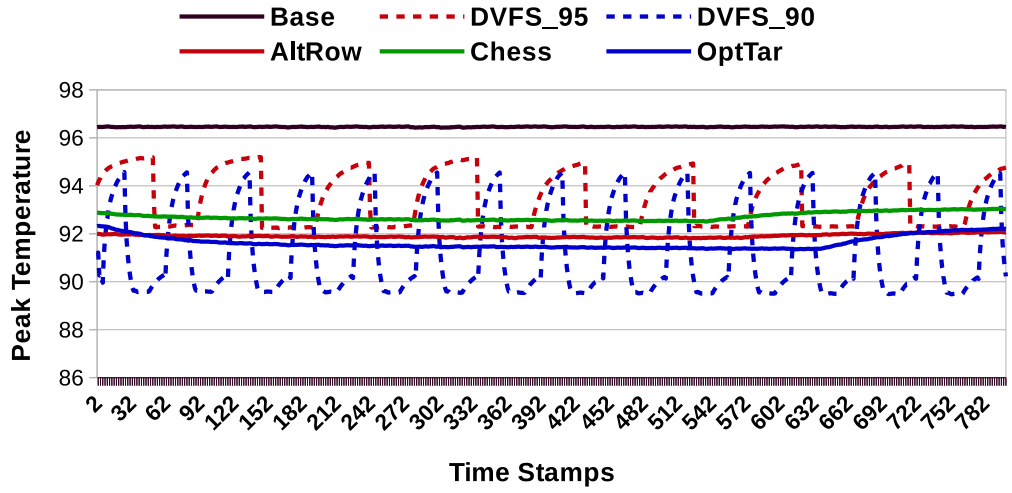
Furthermore, we also show the thermal status for individual L2 banks along with their adjacent cores at an instance during **r**. Instead of showing 64 banks with 16 cores, we have taken the bottom left part of the chip (Core id 0, 1, 14 and 15, with 4 consecutive banks from each row starting with bank id 0, 8, 16 and 24 in Figure 7.6). Figure 7.17 shows the temperatures for these 16 banks with 4 adjacent cores while running Fluid16. The grey blocks in the figure represents shutdown banks. Trivially, remarkable temperature reductions are noticed for the turned off cache area. At the cores, for all of our policies, temperature reduction are in a range of $3 - 4$°C, with maximum reduction of 4.1°C for OptTar.

7.5.5 Comparison with Greedy DVFS [2]

We have implemented a per-core DVFS based thermal optimisation technique (in our simulation framework), called as Greedy DVFS [2] for comparing our cache based thermal efficient policies. Greedy DVFS uses a predefined threshold temperature value, on violation of which V/F setting is scaled down by one step dynamically. Conversely, when temperature of a core is below the threshold, V/F setting will be stepped up for better performance. The change in V/F setting is



(a) Fluid16



(b) Body16

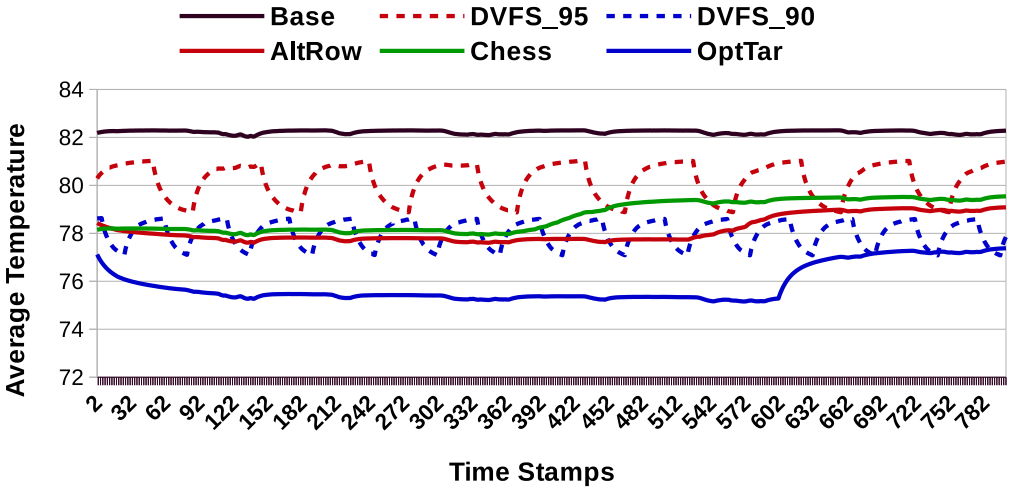
FIGURE 7.18: Snapshot of temporal changes in **peak** temperature of the chip during a sample reconfiguration interval $\boldsymbol{r}$.

Voltage, Frequency	1.8v, 3.0GHz	1.65v, 2.4GHz	1.5v, 1.8GHz	1.35v, 1.2GHz

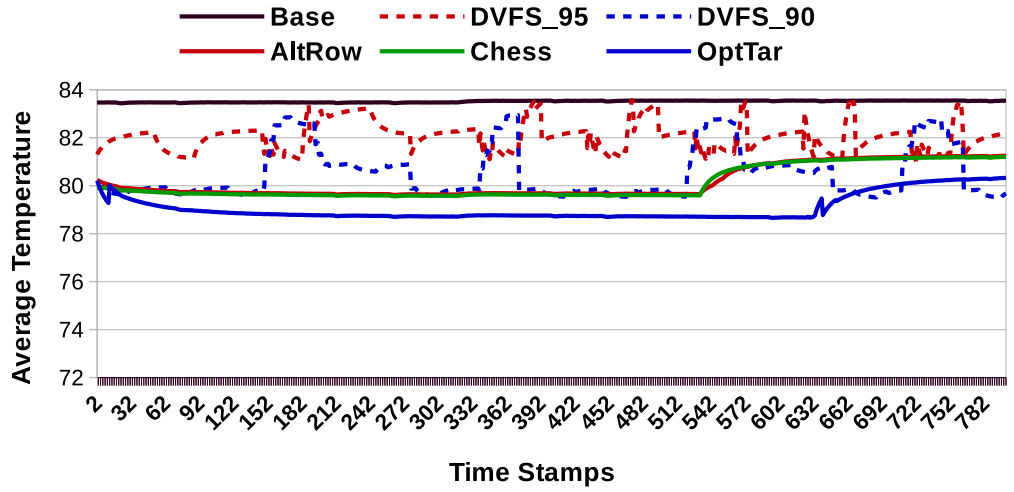
TABLE 7.6: V/F Settings for UltraSPARCIII (used in our simulation)

done periodically per-core at the end of a fixed interval during execution. The detailed V/F settings used in our implementation are given in Table 7.6. We use two threshold values of 95°C and 90°C which are termed as DVFS_95 and DVFS_90, respectively.

Figure 7.18 shows the temporal changes in peak temperature of entire chip for



(a) Fluid16



(b) Body16

FIGURE 7.19: Snapshot of temporal changes in **average** chip temperature during a sample reconfiguration interval $\boldsymbol{r}$.

both Greedy DVFS and our policies during the reconfiguration interval. For both Fluid16 and Body16, our policy shows lesser peak temperature reduction than DVFS_90. As DVFS directly reduces the core's energy consumption, hence, core temperature reduces more than the cache based method. However, in order to compensate the system performance, DVFS shortly scales up the V/F setting that resulting into further increment in temperature within a short time-span. This frequent temperature change can affect the on-chip circuitry due to unstable thermal profile. On the other hand, cache based method ensures stability in reduced temperature range. We achieve around 5.5°C and 4.6°C reduction in peak temperature for Fluid16 and Body16, respectively.

Scaling V/F settings in DVFS generates huge temporal fluctuation in core's temperature incorporating an adverse effect in circuit reliability [148]; whereas cache based techniques offer stable thermal status of the chip. Figure 7.19 shows the temporal change in average chip temperature while using DVFS and our cache based methods. The frequent change in cores' temperature due to Greedy DVFS incurs fluctuation in average chip temperature for both Fluid16 and Body16. The detailed maximum reduction in peak and average chip temperature are given in Table 7.7 and 7.8, respectively. On an average, DVFS_90 and OptTar reduces peak temperature by 6.6°C and 5.5°C, respectively, but cache based policy offers more thermal stability which may help for better durability for the on-chip circuitry. Due to shutting down of larger on-chip area, cache based policy shows more decrement in average chip temperature which is around 4.9°C, whereas DVFS_90 reduces the same by 4.2°C on an average.

A stable thermal profile ensures better reliability for on-chip circuitry. Tables 7.9 and 7.10 show the standard deviations for the temporal changes in peak and average temperature of the chip for different applications. The maximum standard deviation for peak and average temperature are 0.58 and 0.64, respectively, almost for all of our cache based techniques. Whereas, in case of DVFS based policies, least values for standard deviation is 1.01 and 0.67 for peak and average chip temperature, respectively. These values represents better thermal stability of our cache based policies over Greedy DVFS.

Benchmarks	**DVFS_95**	**DVFS_90**	**AltRow**	**Chess**	**OptTar**
Body	3.1	6.1	3.5	3.7	4.9
Fluid	4.4	5.9	3.9	3.6	5.1
Freq	4.2	5.4	4.3	4.5	5.4
Vips	5.7	5.8	4.0	3.9	4.8
Black16	5.5	6.1	3.9	4.5	5.2
Body16	3.8	6.3	4.6	3.9	4.6
Fluid16	4.2	6.6	3.9	4.1	5.5
Swap16	3.2	5.9	3.6	3.8	4.9
Gmean	**4.2**	**6.0**	**3.9**	**4.0**	**5.1**

TABLE 7.7: Maximum Reduction in Peak Temperature (°C) of the Chip with respect to baseline.

Benchmarks	**DVFS_95**	**DVFS_90**	**AltRow**	**Chess**	**OptTar**
Body	2.2	4.0	4.2	3.9	4.8
Fluid	2.1	4.4	4.3	4.2	4.6
Freq	2.8	4.2	4.1	4.0	4.6
Vips	2.4	4.3	4.3	4.4	5.0
Black16	3.2	4.8	4.2	4.1	4.7
Body16	2.7	3.9	3.9	3.8	4.4
Fluid16	3.4	4.8	4.3	4.0	5.8
Swap16	2.0	4.6	4.0	4.2	5.4
Gmean	**2.6**	**4.2**	**4.2**	**4.1**	**4.9**

TABLE 7.8: Maximum Reduction in Average Temperature (°C) of the Chip with respect to baseline.

7.5.5.1 Spatial Thermal Status

Diversities in power consumptions across the on-chip area define the spatial thermal status of the chip. Figure 7.20 shows the thermal status of the chip for baseline,

Benchmarks	**DVFS_95**	**DVFS_90**	**AltRow**	**Chess**	**OptTar**
Body	2.68	2.19	0.46	0.42	0.58
Fluid	1.32	1.01	0.05	0.09	0.21
Freq	1.78	2.13	0.08	0.19	0.34
Vips	1.31	2.67	0.14	0.16	0.42
Black16	1.10	1.95	0.17	0.24	0.28
Body16	2.71	2.51	0.11	0.21	0.32
Fluid16	2.56	2.20	0.16	0.14	0.26
Swap16	2.83	1.36	0.13	0.21	0.23
Gmean	**1.91**	**1.93**	**0.13**	**0.19**	**0.31**

TABLE 7.9: Standard Deviation: **Peak** Temperature of the Chip.

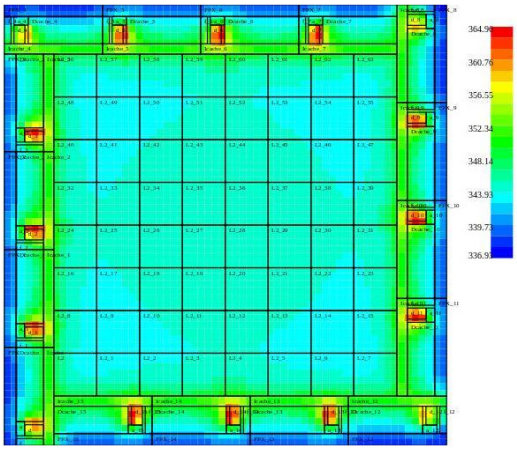
Benchmarks	**DVFS_95**	**DVFS_90**	**AltRow**	**Chess**	**OptTar**
Fluid	0.73	0.81	0.36	0.12	0.61
Body	1.21	0.79	0.51	0.61	0.90
Freq	1.41	0.72	0.23	0.41	0.63
Vips	0.67	0.71	0.55	0.44	0.64
Black16	1.20	1.02	0.51	0.43	0.61
Body16	1.21	1.32	0.71	0.52	0.60
Fluid16	1.11	0.80	0.41	0.17	0.61
Swap16	1.14	0.98	0.34	0.35	0.60
Gmean	**1.05**	**0.85**	**0.43**	**0.34**	**0.64**

TABLE 7.10: Standard Deviation: **Average** Temperature of the Chip.

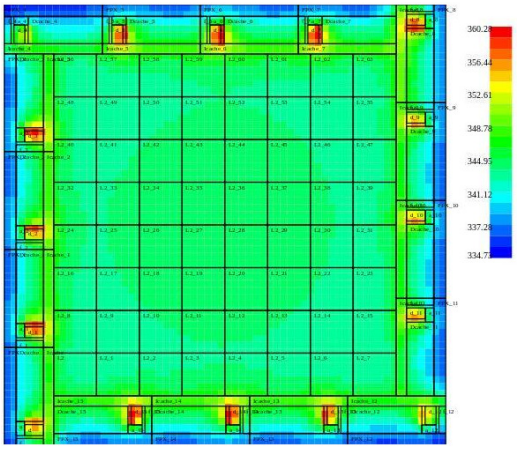
DVFS_90 and OptTar during some $\boldsymbol{r}$, generated from Hotspot 6.0 [3] tool while running Fluid16 application. The corresponding temperature ranges are given in the figures. The peak temperatures have been reduced by 4°C for DVFS_90 and 6°C for OptTar, respectively. In case of OptTar, cache temperature is reduced more as expected and for which average chip temperature shows better reduction than DVFS_90. Moreover, created thermal buffers reduce the peak temperature values of the core area by generating more conductive heat flow towards the cache area from the core. DVFS_90 reduces the core temperature which hardly reduces the cache temperature as reduced cores' temperatures are still a bit higher than the cache temperature. Note that, all the temperature values in Figure 7.20 are in Kelvin(K).

7.5.5.2 Effect on Performance

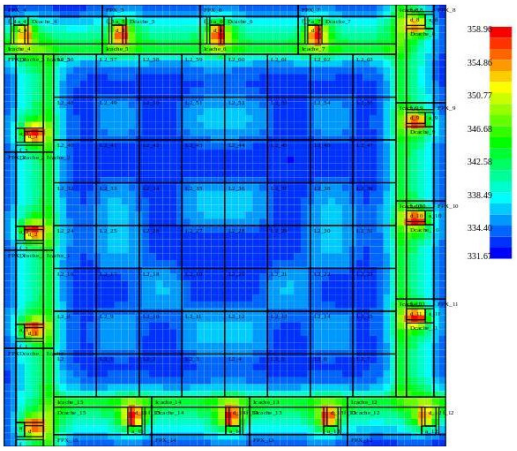
Table 7.11 presents the performance degradation for all the benchmarks with both DVFS policies. DVFS_95, for all the applications, uses maximum frequency and steps it down by one step whereas DVFS_90 uses three levels of frequencies for more reduction in temperature. Hence, overall execution time increases for the applications in DVFS_90 than DVFS_95. The cache based policies does not modify the running frequencies of the cores, hence performance degradation is much lesser than the DVFS policies, although dynamic cache resizing incurs an extra memory latency, which has been taken care in our simulation. However, on an average, DVFS_95 and DVFS_90 degrade performance by 4.38% and 12.51%, respectively.



(a) Baseline



(b) DVFS_90



(c) OptTar

FIGURE 7.20: On-chip spatial thermal behaviour generated by Hotspot 6.0 [3] for Fluid16 at some certain instance during $\boldsymbol{r}$. Temperature Ranges for three configurations-(a) Baseline: $336K$ to $364K$, (b) DVFS_90: $334K$ to $360K$, (c) OptTar: $332K$ to $358K$. Red colour represents peak value where Blue represents the lowest temperature.

The average performance degradation for all the cache based policies are lesser than 4%.

7.5.5.3 Summary

Table 7.12 summarises results for all the policies. DVFS_90 reduces both peak and average temperature at most by 6.6°C and 4.8°C, respectively, whereas OptTar (the best of our proposed policies) reduces peak temperature by 5.5°C and average temperature by 5.8°C. Although, OptTar reduces lesser peak but more average

Benchmarks	**DVFS_95**	**DVFS_90**	**AltRow**	**Chess**	**OptTar**
IPS values (normalised with respect to baseline)					
Body	0.97	0.86	0.98	0.95	0.95
Fluid	0.93	0.90	0.97	0.98	0.97
Freq	0.95	0.85	0.98	0.97	0.97
Vips	0.96	0.87	0.98	0.98	0.97
Black16	0.97	0.88	0.97	0.96	0.96
Body16	0.95	0.86	0.96	0.93	0.96
Fluid16	0.96	0.90	0.96	0.97	0.95
Swap16	0.96	0.88	0.97	0.98	0.98
Average degradation in IPS (with respect to baseline)					
	4.38%	**12.51%**	**2.87%**	**3.51%**	**3.62%**

Table 7.11: Change in IPS with respect to baseline.

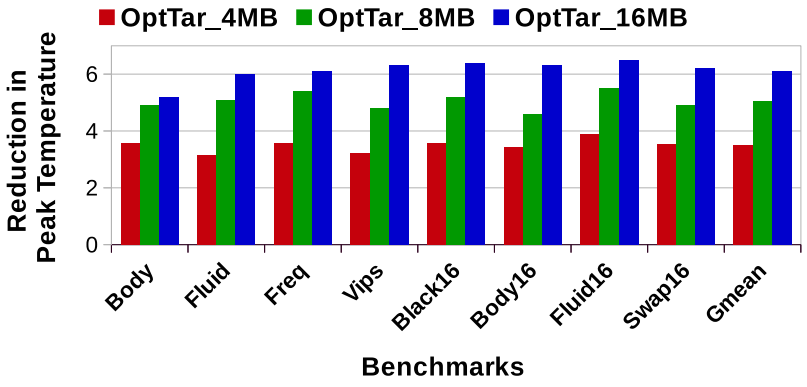
Parameters	**DVFS_95**	**DVFS_90**	**AltRow**	**Chess**	**OptTar**
Max Peak Temp Reduction	5.7°C	6.6°C	4.6°C	4.5°C	5.5°C
Max Avg Temp Reduction	3.4°C	4.8°C	4.3°C	4.4°C	5.8°C
Std Dev of Peak Temp Change	1.91	1.93	0.13	0.19	0.31
Std Dev of Avg Temp change	1.05	0.85	0.43	0.34	0.64
IPS Degradation	4.38%	12.51%	2.87%	3.51%	3.62%

Table 7.12: Summary

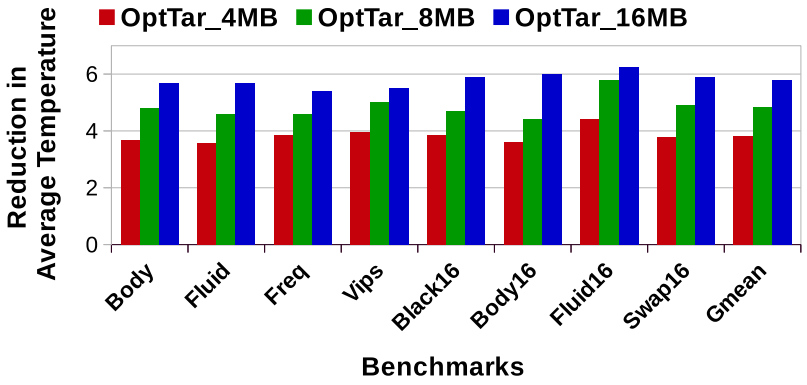
chip temperature than DVFS_90, but, DVFS_90 degrades performance by 12.51% which is 3.62% for OptTar. As LLC occupies a significant portions on-chip and turning off its major parts makes a big chunk of energy free on-chip zone, which reduces more average chip temperature in OptTar than the DVFS based ones. On the other hand, DVFS based policy only handles cores' V/F settings, i.e. it only attacks the on-chip hotspots, hence, peak temperature reduction is more in case of DVFS based policies.

7.5.6 Evaluating the Scalability

To show the scalability, we used OptTar and ran the same set of applications with 4MB and 16MB L2 cache having 64 banks. Figure 7.21 shows the reduction in



(a) Peak Temperature of the Chip



(b) Average Chip Temperature

FIGURE 7.21: Comparing temperature reduction (in °C) with different cache sizes.

peak and average chip temperature over various cache sizes across the applications. We could shutdown 48 banks in the case of 16MB cache and this reduced the peak and average temperatures by 6.1°C and 5.8°C on an average, respectively. Larger turned off banks create larger thermal buffers, hence, more reduction in temperatures. Gating 48 banks in 16MB cache, also maintains the cache size at 4MB which provides lesser cache space to the applications, hence, the IPC degrades around 2%, on an average. Conversely, for 4MB L2 cache, powered off banks create smaller thermal buffers than larger ones, hence, lesser temperature reduction is obtained. The average reduction in peak temperature for 4MB is around 3.5°C, which is around 5.1°C in case of 8MB cache. Moreover, 4MB cache does not allow much resizing while maintaining performance, hence, the thermal benefit is lesser. Our performance cognizant DiCeR maintains the cache size according to the WSS

of the applications. Thus, implementing our policy in larger caches gives more opportunity for reconfiguration and temperature control.

7.6 Conclusion

In this work, we have explored the possibility of using DiCeR towards temperature control in CCMPs. We provide a cache based thermal management technique for modern CMPs equipped with larger centralised LLCs. The cores are placed along the periphery of the chip. Processor based techniques like DVFS are effective in temperature control, however have considerable performance impact.

Considering performance as a system wide constraint, we dynamically resize the LLC to create on-chip thermal buffer which reduces chip temperature. Three different patterns are proposed here to get the maximum reduction in average as well as peak temperature while maintaining the performance within a limit. The simulation results, prepared by running PARSEC workloads, are further compared with the Greedy DVFS, a core based thermal management technique. Both Greedy DVFS and OptTar show maximum reduction in peak temperature by 6.6°C and 5.5°C, respectively, for an 8MB LLC. The temperature reduction is more with the larger LLCs, due to more chances to create larger thermal buffers. DiCeR overheads are taken into account in our simulation, which can be further overlapped with context switching for enhancing performance. Cache based policy shows better thermal stability than DVFS based ones, hence offers more circuit reliability.

In recent CMPs, the cache capacity and area is considerable, and hence, one can use cache reconfiguration to lower overall chip temperature with minimal impact on the performance. The results also show that LLCs can remarkably contribute to safeguard against the thermal breakdown.

Chapter 8

Conclusion and Future Work

Thanks to the manager of DCH, whose staff management skills opened up the avenue for building up this thesis work towards energy and thermal management in modern CMPs.

This research work is motivated towards improving energy and thermal efficiency of the modern CMPs by dynamic cache resizing. Modern CMPs equipped with shrunk transistors, having channel length of 32nm or less, have high power density. This in turn increases the effective chip temperature, that may lead to thermal breakdown of the circuitry. On-chip LLCs in these CMPs consume high leakage power which significantly contributes to the total power consumption of the chip. Also, large LLCs have sufficient potential to create on-chip hotspots when fabricated using shrunk transistors. As power consumption constructs the backbone of on-chip thermal issues, hence, we initially attempt to reduce power consumption of the LLCs by shutting down some least used portions of it, which also helps us for improving thermal efficiency.

Basically, to improve the energy efficiency of a TCMP, we dynamically resized the LLC at two granularity levels: (i) at bank level, and (ii) both at bank and way levels (i.e. hybrid). Later, towards improving thermal efficiency of the CMPs, we exploit the dynamic cache resizing at the LLCs in following ways: (i) for a TCMP

having 16 banked LLC, we turned off least used banks as well as heavily used banks, and (ii) for a CCMP with 64 banked LLC, we turned off cache banks by following some resizing patterns that assists to reduce the cache as well as core temperature.

8.1 Summary of Contributions

Traditional cache accesses follow locality of reference, which anticipates that, currently least used banks will have least usages in future. Hence, to reduce the leakage energy consumption of the LLCs with minimal impact on performance, in our very first attempt, we decided to turn off a set of lightly used cache banks dynamically (DiCeR). But, in modern multi-tasking environment, locality of reference is violated for the long running applications. Therefore, one-time shutting down of cache banks will have an adverse effect on the system performance if cache space cannot be provided dynamically to the applications on demand in future. Hence, in order to put a balance between energy consumption and performance, we proposed a performance constrained dynamic cache resizing technique that turns off least accessed cache banks to save leakage, while maintaining performance degradation within a certain limit by turning on cache banks when needed. For a 4MB 4 way set associative cache, this technique achieves a leakage energy saving by 65% with 30% gain in EDP, while performance penalty is minimal.

The workloads of the turned off banks in DiCeR are handled by the remaining powered on banks, resulting into increased capacity and conflict misses at these banks (called as target banks). Towards mitigating this fact, we incorporated DAM based technique named CMP-SVR at these powered on banks which improves system performance by reducing the number of cache misses. Subsequently, this performance gain is further traded to save more leakage energy by turning off more cache banks as well as cache ways. Moreover, a bank that experiencing 50% of its cache way shutdown, still maintains more associativity than its actual value due to the activation of DAM. However, in association with DAM, this policy saves

leakage by 70% with 35% EDP gains for a 4MB 8 way set associative cache. DAM in combination with DiCeR is also effective for the smaller sized caches, where we also provide the option for turning on of the cache ways on demand. For a 2MB 8 way L2, this technique saves leakage by 52% while performance degradation is lesser than 3%. Note that, to reduce NoC overhead, this policy also selects target banks in a distance cognizant manner.

In our next exploration, we have exploited the DiCeR for increasing thermal efficiency on-chip. The heavily used cache portions consume higher dynamic energy, which in turn creates hotspots at these cache locations. Therefore shutting down of these cache portions will reduce temperature at these hotspots. But, managing subsequent requests at these heavily used powered off banks may create hotspots at their target locations and can also curtail system performance. The controlled used of DiCeR helps to reduce cache hotspots while keeping performance degradation within a certain limit. On the other end of the spectrum, least used cache banks consume heavy leakage which forms a circular dependency with the chip temperature and may prone to thermal breakdown in future. Hence, turning off these banks not only reduces leakage rather they create on-chip thermal buffers which will further reduce the effective chip temperature. However, for a TCMP having an 8MB L2 as LLC, both of these policies reduce average chip temperature by around 4°C with minimal impact on the performance.

The heuristic solution given for DiCeR performs well with the 16 banked cache. But, analytical formulation can stringent the heuristic claim by putting a solution for the optimal cache size while maintaining performance degradation within a certain threshold. In this work, we initially design an analytical solution for optimal cache size and tried to maintain the same throughout the execution of the process, in a CCMP having 64 L2 banks. While maintaining this optimal cache size, we dynamically turned off cache banks in some specific locations on-chip which urgently need an ample amount of thermal buffers for reducing temperature. Towards this, we develop three different cache resizing patterns and used them during process execution for better thermal efficiency. For an 8MB L2 cache having 64 banks,

this policy achieves more than 5°C reduction in peak and average chip temperature, which are comparable with a traditional core based thermal management technique.

Figure 8.1 summarises the contributions of this thesis.

8.2 Scope for Future Work

The contributions of this thesis can be extended in a number of ways. Some of these possible future research directions are listed below:

- Our state destroying caches although saves significant amount of leakage energy but may be a costlier one when WSS is large enough to fit into the available actual cache space. To mitigate this system overhead, we can also exploit state preserving drowsy cache technique in addition with DiCeR. System can alternatively decide and apply either of these techniques for leakage minimisation. Towards implementing this, power gating circuitry has to be attached with the MTCMOS circuitry.

- DiCeR works well enough with our 2D CMPs for a smaller set of cores and LLC banks. But, this technique can also be applied to the modern 3D CMPs where caches are placed in multiple 2D layers. Depending upon the applications, DiCeR can be applied in addition with DAM and way turn off/on techniques. The dynamic change in system performance will further apply the energy optimisation policy during execution.

- DiCeR can further be attached with task migration and/or DVFS to enhance its thermal efficiency. Basically, towards applying this in a large CMP, an analytical model has to be developed for selection of proper optimisation policy for controlling chip temperature.

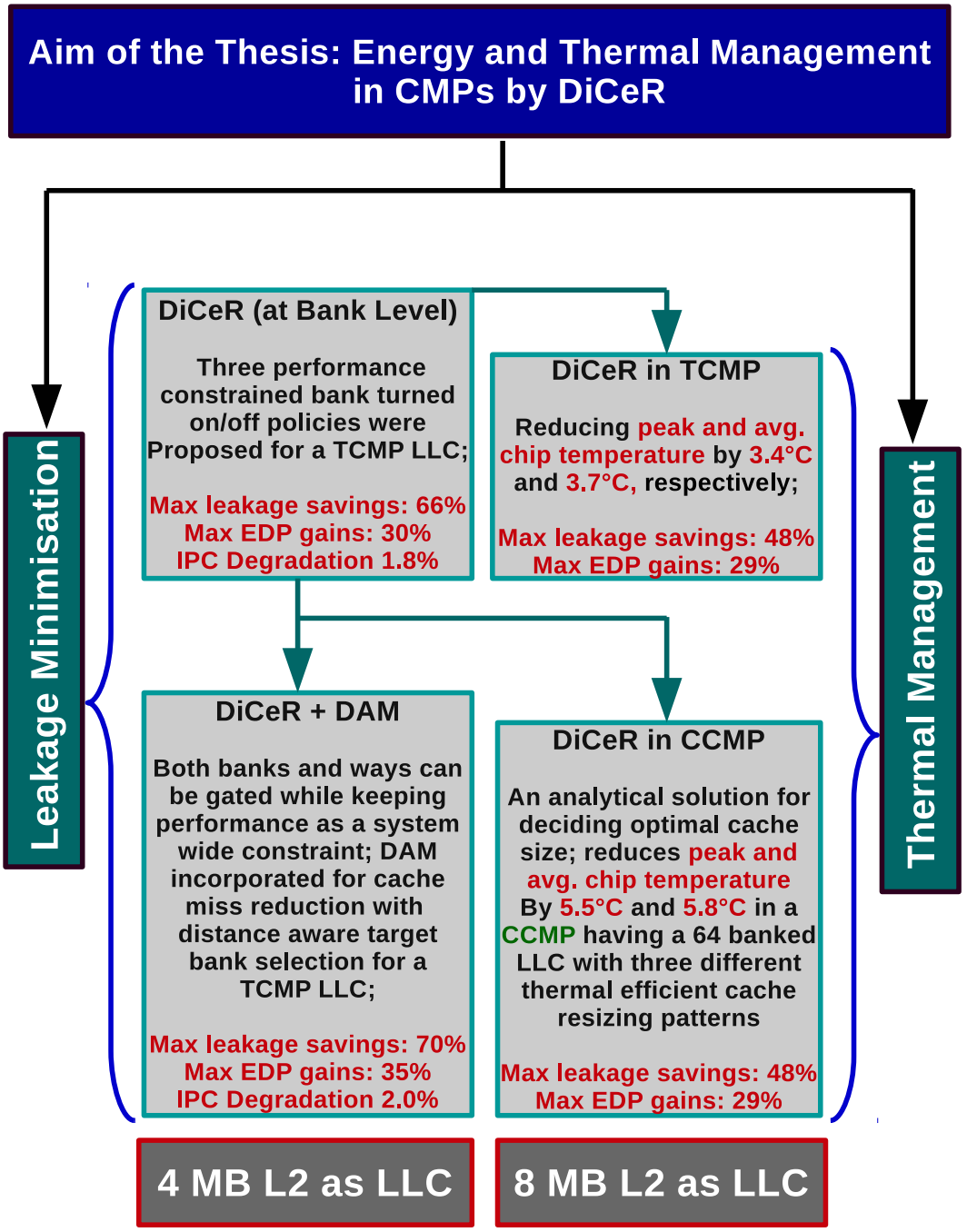


FIGURE 8.1: Summary of the thesis contributions. The results for the leakage savings are shown for 4MB caches and for thermal management the cache size is 8MB as given in the figure. Note that, the proposed architectures are also evaluated with various cache configurations.

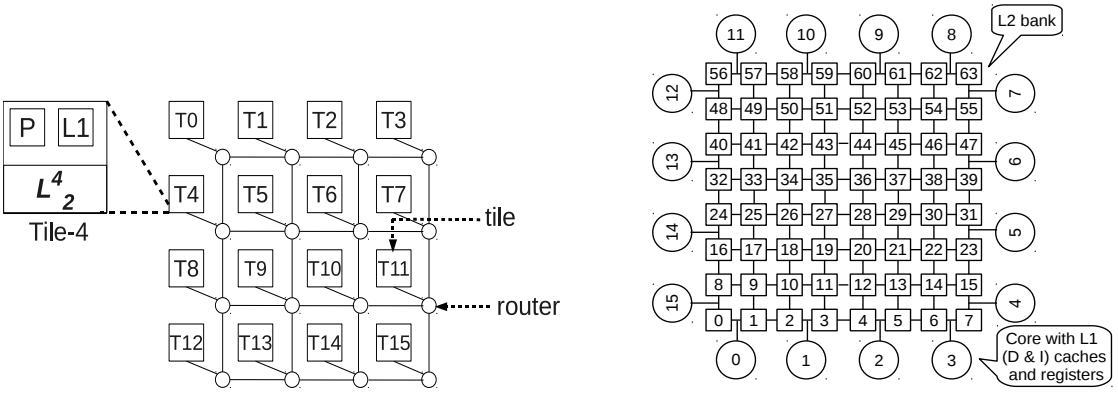
Appendix A

Closed Loop Simulation Framework

In this chapter, we will discuss about the details of our closed loop simulation framework, that we have used for our dynamic thermal simulation. We have done our thermal simulation for the last couple of contributions, where a TCMP and a CCMP architecture are simulated. Before going into the details of these simulation methodologies, we first discuss about our baseline architectures with their respective configurations. Figure A.1 shows our baseline architectures for a TCMP (Fig. A.1(a)) and for a CCMP (Fig. A.1(b)).

TCMP and CCMP Architecture The whole TCMP is divided into 16 identical tiles where each tile contains a processor core from UltraSPARCIII family, a private L1 Data and Instruction caches and a chunk of shared L2, called as L2 bank. A 2D mesh NoC binds up the whole chip by attaching each of the tile to a dedicated router. Before configuring this architecture, we first look into the micro-architectural schema of a processor core. The tile area in a TCMP is sub-divided into two major parts-(a) Core area and (b) L2 bank.

CCMP, on the other hand, contains 16 cores placed along the periphery of the chip. The whole L2 as shared LLC is located at the central portion of the chip



(a) Tiled CMP architecture

(b) CMP with centralised LLC

FIGURE A.1: A Tiled CMP and a CMP with Centralised LLC.

and divided into 64 equal sized cache banks. The core configuration is same in both TCMP and CCMP we used in our works. The core area along the periphery contains the same components like a tile in TCMP except the L2 bank.

A.1 Components at Core area and L2 Cache

While configuring the simulation setup, initially we design the UltraSPARCIII core in Simics and McPAT simulators. In very first step, the physical core components are required to be designed in the simulators. Our simulated core area contains the following set of physical components [21]-

- **Pipelines per Core:** we have two integer pipelines and one floating point pipeline per core.
- **Pipeline Depth:** determines the number of pipeline stages in each pipeline, which is 5 for both types of pipelines.
- **ALU and MUL per Core:** we have two ALU and one MUL (for multiplications and divisions) unit per core used for integer instructions.
- **FPU per Core:** one floating point unit serves the FP instructions.
- **Instruction Buffer Size:** this buffer is able to store 32 entries at a time and is placed between IF and ID stages of the pipeline.

- **Decoded Stream Buffer Size:** this buffer is located between ID and EXE stages of the pipeline and has 16 entries in it.
- **Int and FP Instruction Window Size:** the size of both integer and floating point instruction windows are 16.
- **ROB Size:** as we are using in-order processor, hence, ROB is absent in our design.
- **IRF and FRF Size:** the integer and floating point register files can store 32 data in each of them.
- **Load and Store Buffer Size:** as in-order cores do not need load buffer, hence, only store buffers with 64 entries are kept at the core area.
- **Icache and Dcache Config:** both instruction as well as data caches have size of 64KB with 4 way set associativity.

All of these above parameters are fixed through out the execution and kept same for both TCMP and CCMP core designs. The LLC area is as follows:

- **L2 Config:** an 8MB LLC is used in our simulation. For TCMP, we have 16 banks and each of the bank having a size of 512KB. For CCMP, we have 64 identical banks and each bank can contain 128KB of cache data.

A.2 Initial Simulator setup and Floorplan

During installation of Simics simulator we have provided the core model, i.e. Ultra-SPARCIII in our case, and Simics implicitly prepares the core components except L1 caches. Additionally, to make it a multi-core, we also specify the number of cores. Once the initial virtual hardware is prepared, we install Solaris Operating System in it. Note that, the virtual hardware offered by installed Simics system is known as target machine, and the computer on which we install Simics is called

as host machine. The file transfer between the host machine and target machine is done through some created mount point of Solaris at target machine.

However, once, Simics installation is over with Solaris, GEMS is compiled and attached with it at the Simics interface arena. GEMS has its own configuration file, where we have specified the parameters related to on-chip memories and NoC. This GEMS+Simics integration together forms the complete system. Specifically, we use "rubyconfig.defaults" file from the Ruby memory module of GEMS to develop our simulated architecture. In this file, we first specify the size and associativity of L1 Data and Instruction caches. These caches are private to each core and the values specified here determines a single local L1 cache. Now, in the next, we further specify the shared L2 size and its associativity. We also have fixed the number of available on-chip cache levels along with a fixed cache block size across the cache levels. The main memory size is also specified in the same file. Once, these memory parameters are ready, now we specify the number of L1 and L2 banks. Further, for setting up the 2D mesh NoC, we need to determine the NoC parameters, i.e. Virtual Network, Virtual Channel etc. Garnet network module of GEMS takes care about this. At this point our GEMS+Simics setup is ready to run.

McPAT on the other hand, now has to be modeled for the same kind of architecture. However, McPAT itself has a default xml configuration interface for UltraSPARCIII processors. Now, we specify all of these fixed parameters in this xml interface from where McPAT models the target CMP. The components discussed in Section A.1 are specified in this file and McPAT is executed for the very first time to get the area details for the individual components. Note that, area details directly depends upon the technology, i.e. channel length of the transistors those are kept at the backbone. This technology parameter is also specified in the xml interface. Following command executes the McPAT setup and gives us the area details for components at core area and L1 and L2 caches:

Syntax: ./mcpat -infile <*.xml> -print_level <level of detailed output>

Example: ./mcpat -infile Niagara2.xml -print_level 5

The example shows the actual data/values that we have used in our simulation. Niagara2.xml is an input (processor description) file, provided by the McPAT simulator, contains the same component descriptions like UltraSPARCIII processor [13, 149]. Hence, we use this file for our processor description and we have changed the cache parameters (like associativity, size etc.) as needed. McPAT provides 5 levels of detailed printing of its output parameters. Higher values of this provide more details where lower values produce values at more abstract level. For thermal modelling, we need to have simulated area as well as power consumption details of the individual components, hence, we printed the outputs in details with print level 5.

However, the outputs of McPAT is now ready for our floorplan construction. Towards this, we now use the HotFloorplan simulator from HotSpot 6.0 [3], that prepares the floorplan of the whole chip by executing the following command:

./hotfloorplan -c hotspot.config -f ev6.desc -p avg.p -o output.flp

In the above command, the executable **hotfloorplan** takes four options during its execution. The first option '-c' enables the simulator to take inputs from the HotSpot configuration file. '-f', the second option enables floorplan descriptions as input for the individual on-chip components, whereas the third option '-p' gets average power consumption of the respective components. The average power values are the critical parameters while designing the thermal efficient CMPs. Finally, '-o' prints the developed floorplan into the "output.flp" file, the desired floorplan. Next, we discuss these input/output parameters in details.

HotSpot Configuration [hotspot.config] This file contains two classes of parameters that have to be specified: (a) Thermal model parameters and (b) Floorplanner parameters. The set of thermal model parameters includes the following:

1. **Chip Specification:** this subclass includes chip thickness, thermal conductivity of silicon, specific heat of silicon and temperature threshold for enabling DTM.
2. **Heat Sink Specification:** For more realistic simulation, HotSpot further includes heat sink specifications, that include convection capacitance, convection resistance, length of side of heatsink, heatsink thickness, its thermal conductivity and the specific heat.
3. **Specification of Interface Material:** Here, thickness thermal conductivity and specific heat are mentioned for the interface material.
4. **Others:** In this zone, ambient temperature is to be mentioned. Apart from that, processor clock frequency and the sampling intervals need to be specified.

All of these parameters are used for thermal modeling during calculation of temperature values. The latter one, i.e., floorplanner parameters include the following:

1. **L2 modeling:** The number of L2 banks with their labels are to be mentioned here.
2. **Rim Modelling:** The rim thickness has to be specified with usage of dead space on-chip while making the floorplan.
3. **Annealing Parameters:** While making CMP floorplan, this simulator needs simulated annealing method for preparing the floorplan. Actually, this policy tries several floorplans and selects the best combination as output to the users. However, detailed process regarding this is out of scope of this thesis.

Floorplan Descriptions [ev6.desc] This description file contains the area (in mm^2) of the individual components with its minimum & maximum aspect ratio (min-asp & max-asp). Aspect ratio implies the ratio of height and width for each of the on-chip components. Among the other positional parameters in the 2D plane of the chip, this file also provides whether a component is rotable or not. The format is as follows:

Syntax: <unit-name> <area> <min-asp> <max-asp> <rotable>

Example: FPX $1.17785e-6$ 1 6 1

In this example, we have described the details of FPX, i.e. Floating Point ALU having an area of $1.17785e-6mm^2$, with minimum & maximum aspect ratio of 6 and 1. This component is rotable and hence, we put 1 at the end. Additionally, this file also provides information about the connectivity among the components, i.e. which unit should have direct connectivity with which one. The format for the same is as follows:

Syntax: <unit1-name> <unit2-name> <wire-density>

Example: InX IS 1

This command uses three parameters, where first two parameters represent two units, and wire-density describes about their adjacency information. In this example, InX and IS are Integer ALU and Instruction Scheduler. This two units have to be placed in such a way so that, they can be connected to each other directly, for which wire-density = 1 has to be written.

Average Power Consumption [avg.p] Towards generating average power consumption, we executed three PARSEC applications-Black16, Body16 and Fluid16 upto 100 ruby cycles in GEMS+Simics framework. These traces are further sent

to McPAT for simulation of power values. For collecting the central tendency of the average power consumption values, we extracted geometric mean of the power consumptions of individual elements, across the applications. These power traces are then used in our avg.p file. This file contains the unit-names along with its average power consumption which looks as follows:

avg.p

```
=========
- - - - - - - - - - -
- - - - - - - - - - -
FPX 1.3032335
Reg 0.0014071
InX 2.3281616
ITB 0.0008605
Mul 0.8638675
IS 0.4180698
- - - - - - - - - - -
- - - - - - - - - - -
=========
```

We have shown a smaller view of the avg.p file above, where we have only six on-chip components along with their power consumptions (in Watt). The components are as follows:

- FPX: Floating Point ALU
- Reg: Registers
- InX: Integer ALU
- ITB: Instruction Buffer
- Mul: Multiplication unit

- IS: Instruction Scheduler

Floorplan as Output [output.flp] The generated output is written into output.flp file in which following format is followed:

Syntax: <unit-name> <width> <height> <left-x> <bottom-y>

Example: Dcache 0.0007746983 0.00209321 0.00033104 0.00067272

Each row of this file contains the data corresponding to a particular unit. We have shown only for Data Cache (Dcache), where unit name is placed at the beginning of the row followed by its width, height, and co-ordinate position (left-x and bottom-y) on the 2D plane of the chip. Once **output.flp** is prepared, an embedded Perl script in HotSpot can be used to generate the graphical floorplan of the CMP. The command for the same is given below:

```
./tofig.pl ev6.flp | fig2dev -L ps | ps2pdf - ev6.pdf
```

Note that, to avoid overlapping in our graphical floorplan, we replaced the names of some elements (have large names) with identical alphabets as follows:

- **a** represents Reg.
- **b** represents IS.
- **c** represents ITB.
- **d** represents Mul.
- **e** represents DTB.
- **f** represents InX.

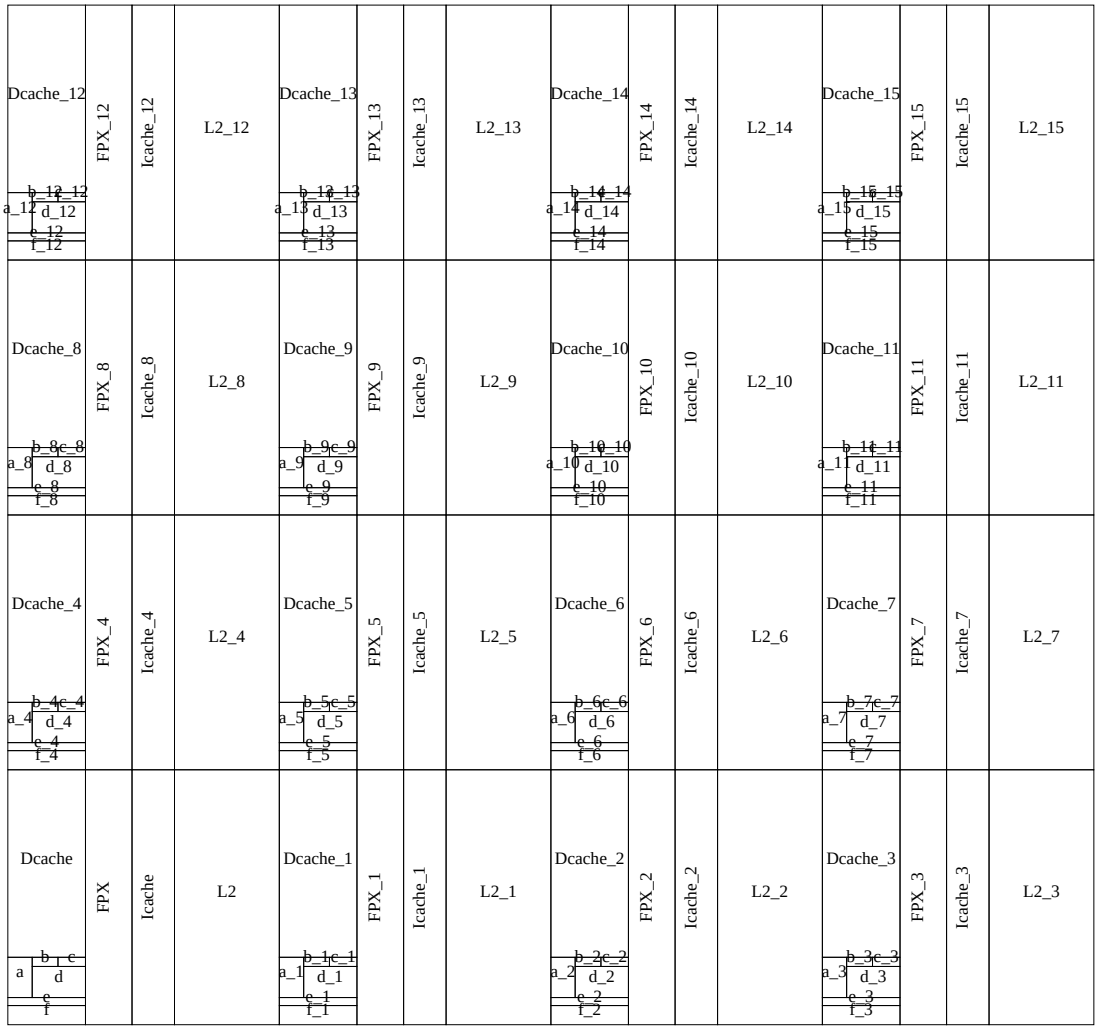


FIGURE A.2: TCMP Floorplan

Components	**Parameters**
Processor	UltraSPARCIII+
Flit Size	16 bytes
Buffer Size	4
#Virtual Networks	5
Memory bank	1GB, 4KB/page
Pipeline Stage	5-stage
VCs per Virtual Network	4

TABLE A.1: System and Network Parameters

And names of the all other remaining elements are kept unchanged. However, the prepared floorplans for both of our architectural frameworks i.e. TCMP and CCMP are depicted in Figure A.2 and A.3, respectively. The L2 banks in these figures are represented as L2_[Bank-ID]. The fixed parameters regarding NoC, Core and Caches, used in our simulation are given in Table A.1 and A.2.

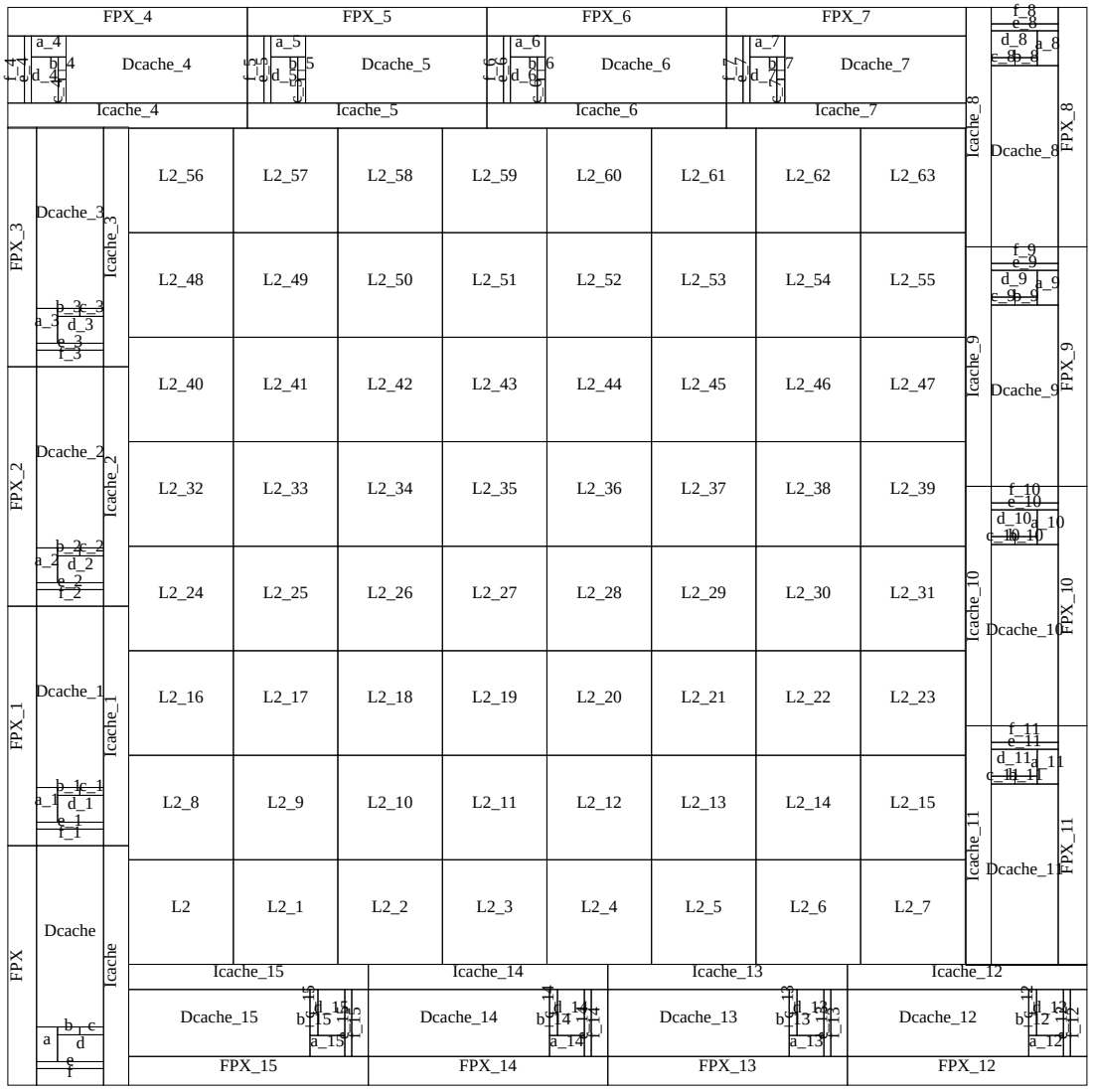


FIGURE A.3: CCMP Floorplan

A.3 A few fixed McPAT Parameters

Apart from our previously mentioned parameters, McPAT also takes the following parameters from its xml based input interface:

- **Instruction Length:** length of an instruction that has to be processed by the cores and this length has to be similar with or multiples of the memory transfer length, else integral number of instructions cannot be provided in a single fetch cycle to the requested core.

Cache Parameters	Values	Core Parameters	Values
L1 I/D Cache	64KB, 4-way	Clock rate	2.4 or 3.0GHz
An L2 Bank	128 or 512KB	ALU per core	2
Block Size	64 Bytes	FPU per core	1
Technology used	32nm	MUL per core	1
L2 Associativity	8	Ambient temperature	47°C
Cache Model	NUCA	Sampling interval	25 or 30μS

TABLE A.2: Cache and Core Parameters for McPAT and HotSpot Configurations

- **Opcode Width:** determines the width of opcode of an instruction, which can have two parts- opcode and operands. Opcodes are supplied by the L1 Instruction cache to the cores.
- **Machine Type:** is used to decide whether this core architecture supports Out-of-Order (OoO) executions or not. If it supports OoO execution, then ROB or Re-Order Buffer has to be attached with this core.
- **Number of Hardware Threads:** indicates the level of parallelism a core can achieve.
- **Fetch Width:** an amount that can be fetched from the memory during fetch cycles. This value is usually similar to the size of L1 cache line.
- **Number of Instruction Fetch Ports:** available number of ports for fetching instructions.
- **Decode Width:** the amount can be decoded at a time. The peak decode rate is changed with the number of cores that can run concurrently.
- **Issue Width:** usually same with the dispatch width of the instruction scheduler.
- **Commit Width:** is used to determine the number of ports available for the register files, so, that much amount can only be sent to the register on commit.
- **FP Issue Width:** same like the *Issue Width* but used for Floating Point instructions.

- **Prediction Width:** this component is related to the branch prediction architecture for mitigating control hazards during execution.

A.4 Closed Loop Simulation

Figure A.4 shows the closed loop simulation framework that we have developed and used in our dynamic thermal simulation once the floorplan is ready. While simulation is running, the sampling period for performance traces has been taken uniformly at 0.1 million Ruby cycles (which is around 30 micro seconds) at a speed of 3.0 GHz. On an average, the simulations last for around 200 million ruby cycles, hence, in whole simulation there will be around 2000 such sampling points. Although the change in thermal status is a continuous process in real hardware, but, in our discrete-event simulation model, it has to be modeled in a discrete manner. However, sampling rates are sufficient enough [29] for such simulations. The whole process can be summarised as follows:

1. Run applications in GEMS+Simics environment to get the traces at the sampling points.
2. At the end of each sampling point collect the traces and send them to McPAT to get the power consumption values.
3. Get power values from the outputs of McPAT and feed them to the HotSpot for obtaining the temperature traces.
4. Use these temperature traces for deciding the next LLC configuration for the next sampling intervals.
5. Repeat the whole process until the end of process execution.

We use shell scripts to construct this closed loop simulation framework by integrating these three simulators.

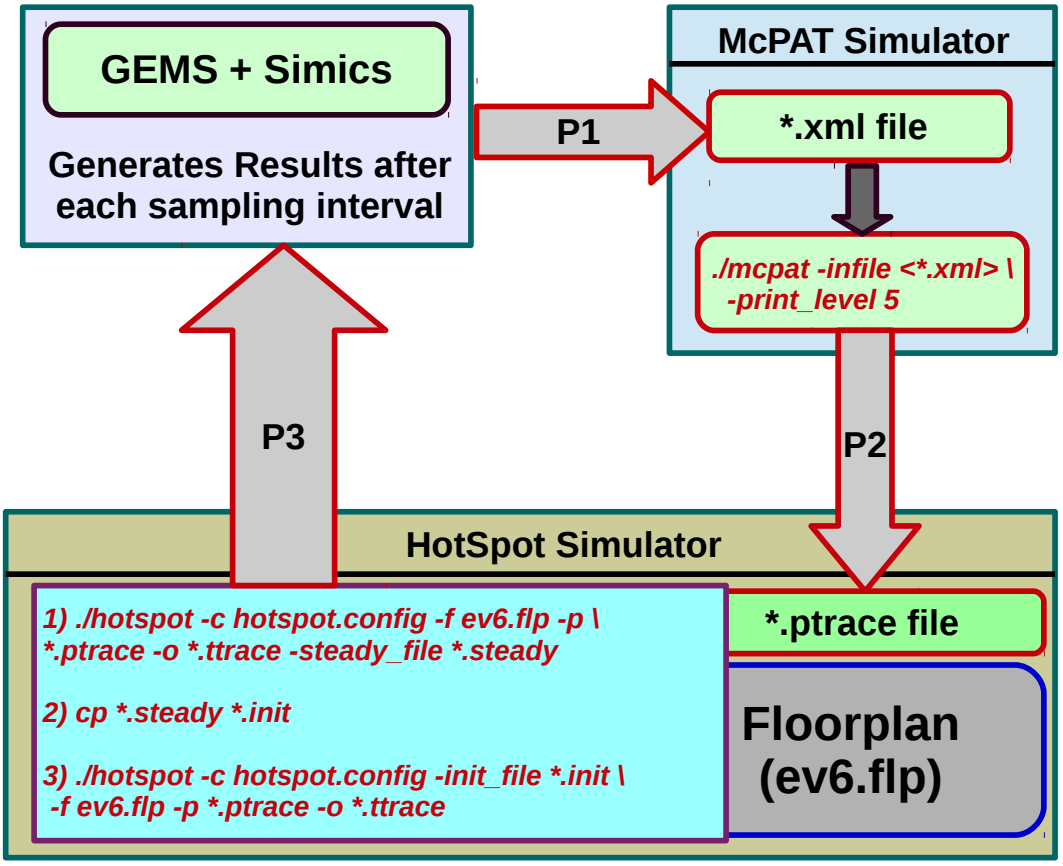


FIGURE A.4: Our Closed Loop Simulation Framework for thermal analysis.

A.4.1 Data Flow in Closed Loop Simulation

The data flow among these simulators play the most significant role during the simulation. We divided this flow of data into three modules, named P1, P2 and P3, as shown in Figure A.4. In this section, we will discuss about these three modules one by one.

P1. This data module flows between GEMS+Simics and McPAT simulators. The outputs of GEMS at each sampling interval is evaluated and from which our shell script prepares the input data to construct the new input file (*.xml) for McPAT. From GEMS output, our script extracts the total number of executed: (a) instructions, (b) integer instructions, (c) floating point instructions, (d) branch instructions, (e) load instructions and (f) store instructions. Additionally, it also extracts: total completed clock cycles, Icache read accesses and read misses, total read and write accesses and misses for each L1 Data and L2 Banks. All of these

extracted values are then placed at their proper locations in *.xml input file of McPAT.

P2. The next data flow takes place between McPAT and HotSpot, where generated power values from McPAT are to be written in the *.ptrace file of HotSpot. The output of McPAT simulator are stored in a file from where our script extracts the total power (Dynamic + Static) consumption of every component discussed earlier. The power values for individual components here indicate the power consumptions over the last sampling interval only. All these values are taken together and appended at the *.ptrace file which provides input to the HotSpot. The power values generated by the McPAT mostly depend upon the following parameters, collected from GEMS output in every iteration: total (and different kinds of) instructions executed, number of cache accesses and misses. The technology parameter along with the current average chip temperature further help McPAT together to generate dynamic and static power consumptions of the individual elements.

P3. On its each and every run, along with the fixed floorplan, HotSpot uses all the power values from the beginning for continuous modeling of temperature values. Once, HotSpot gets power trace, initially it executes to generate the steady state temperature at the last sampling interval (by using 1st command written in the HotSpot block in Figure A.4). This steady-state temperature is further used as the initial temperature at this stage (*.init file). To prepare this a copy command is executed (2nd command in the HotSpot block in Figure A.4). This *.init file is now contains initial temperatures for all of these components and by issuing third command (from HotSpot block in Figure A.4), we get the final transient temperature values for the current sampling interval. This temperature values are written in a *.ttrace file which will be read by GEMS, from P3 interface, for making decisions on future cache configuration.

Bibliography

[1] B. Fitzgerald, S. Lopez, and J. Sahuquillo, "Drowsy cache partitioning for reduced static and dynamic energy in the cache hierarchy," *International Green Computing Conference (IGCC)*, pp. 1–6, June 2013.

[2] A. Mirtar, S. Dey, and A. Raghunathan, "Joint work and voltage/frequency scaling for quality-optimized dynamic thermal management," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 6, pp. 1017–1030, Jun. 2015.

[3] R. Zhang, M. R. Stan, and K. Skadron, "Hotspot 6.0: Validation, acceleration and extension." in *University of Virginia, Tech. Report CS-2015-04*, 2015.

[4] W. Zang and A. Gordon-Ross, "A survey on cache tuning from a power/energy perspective," *ACM Comput. Surv.*, vol. 45, no. 3, pp. 32:1–32:49, Jul. 2013.

[5] S. Mittal, "A survey of architectural techniques for improving cache power efficiency," *Sustainable Computing: Informatics and Systems*, vol. 4, no. 1, pp. 33 – 43, 2014.

[6] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '08, 2008, pp. 72–81.

[7] R. B. Naveen Muralimanohar and N. P. Jouppi, "Cacti 6.0: A tool to model large caches," in *Technical Report HPL-2009-85, HP Laboratories*, 2007.

[8] T. Hayes, O. Palomar, O. Unsal, A. Cristal, and M. Valero, "Future vector microprocessor extensions for data aggregations," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, June 2016, pp. 418–430.

[9] "International Technology Roadmap for Semiconductors - ITRS 2.0," http://www.itrs2.net/, [Online].

[10] [Online]. Available: http://www.intel.com

[11] R. Balasubramonian, N. P. Jouppi, and N. Muralimanohar, *Multi-Core Cache Hierarchies*. Morgan and Claypool Publishers, 2011.

[12] P. Kongetira, K. Aingaran, and K. Olukotun, "Niagara: A 32-Way Multithreaded Sparc Processor," *IEEE Micro*, vol. 25, pp. 21–29, Mar. 2005.

[13] [Online]. Available: https://www.oracle.com/sun/index.html

[14] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Fourth Edition: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., 2006.

[15] J. L. Henning, "SPEC CPU2006 Benchmark Descriptions," *ACM SIGARCH Computer Architecture News*, vol. 34, no. 4, pp. 1–17, sept 2006.

[16] D. E. Culler and J. P. Singh, *Parallel Computer Architecture: A Hardware/-Software Approach*. Morgan Kaufmann Publishers, 1999.

[17] C. Bienia, S. Kumar, and K. Li, "PARSEC vs. SPLASH-2: A quantitative comparison of two multithreaded benchmark suites on Chip-Multiprocessors," in *IEEE International Symposium on Workload Characterization (IISWC)*, sept 2008, pp. 47–56.

[18] S. Das and H. K. Kapoor, "A framework for block placement, migration, and fast searching in tiled-dnuca architecture," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 22, no. 1, pp. 4:1–4:26, May 2016.

[19] M. E. Acacio, A. Ros, and J. M. Garcia, "Scalable Directory Organization for Tiled CMP Architectures," in *Proceedings of the International Confrence on Computer Design*, 2008, pp. 112–118.

[20] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. F. B. III, and A. Agarwal, "On-Chip Interconnection Architecture of the Tile Processor," *IEEE Micro*, vol. 27, no. 5, pp. 15–31, 2007.

[21] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec 2009, pp. 469–480.

[22] S. Kaxiras and M. Martonosi, *COMPUTER ARCHITECTURE TECHNIQUES FOR POWER-EFFICIENCY*. Mark D. Hill, University of Wisconsin, Madison, 2008.

[23] A. P. Chandrakasan and R. W. Brodersen, *Low Power Digital CMOS Design*. Norwell, MA, USA: Kluwer Academic Publishers, 1995.

[24] M. Poremba, S. Mittal, D. Li, J. S. Vetter, and Y. Xie, "Destiny: A tool for modeling emerging 3d nvm and edram caches," in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2015, pp. 1543–1546.

[25] H.-S. Wang, X. Zhu, L.-S. Peh, and S. Malik, "Orion: a power-performance simulator for interconnection networks," in *35th Annual IEEE/ACM International Symposium on Microarchitecture, 2002. (MICRO-35). Proceedings.*, 2002, pp. 294–305.

[26] K. Roy and S. C. Prasad, *Low-Power Cmos Vlsi Circuit Design*. John Wiley and Sons, 2009.

[27] Y.-C. Yeo, T.-J. King, and C. Hu, "Mosfet gate leakage modeling and selection guide for alternative gate dielectrics based on leakage considerations,"

IEEE Transactions on Electron Devices, vol. 50, no. 4, pp. 1027–1035, April 2003.

[28] V. Hanumaiah, S. Vrudhula, and K. S. Chatha, "Maximizing performance of thermally constrained multi-core processors by dynamic voltage and frequency control," in *IEEE/ACM International Conference on Computer-Aided Design - Digest of Technical Papers*, Nov 2009, pp. 310–313.

[29] J. Kong, S. W. Chung, and K. Skadron, "Recent thermal management techniques for microprocessors," *ACM Comput. Surv.*, vol. 44, no. 3, pp. 13:1–13:42, Jun. 2012.

[30] S. Pagani, "Power, energy, and thermal management for clustered many-cores," Ph.D. dissertation, Karlsruhe Institute of Technology (KIT), Karlsruhe, November 2016.

[31] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge, "Drowsy caches: simple techniques for reducing leakage power," in *Proceedings 29th Annual International Symposium on Computer Architecture*, 2002, pp. 148–157.

[32] H. Homayoun, M. Rahmatian, V. Kontorinis, S. Golshan, and D. M. Tullsen, "Hot peripheral thermal management to mitigate cache temperature variation," in *Thirteenth International Symposium on Quality Electronic Design*, 2012, pp. 755–763.

[33] K. Stavrou and P. Trancoso, "Tsic: Thermal scheduling simulator for chip multiprocessors," in *Proceedings of the 10th Panhellenic Conference on Advances in Informatics*. Springer-Verlag, 2005, pp. 589–599.

[34] S. Das and H. K. Kapoor, "Dynamic associativity management using fellow sets," in *ISED*, 2013, pp. 133–137.

[35] H. Everett, "Generalized lagrange multiplier method for solving problems of optimum allocation of resources," *Oper. Res.*, vol. 11, no. 3, pp. 399–417, Jun. 1963.

[36] C. Kim, D. Burger, and S. W. Keckler, "An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches," *ACM SIGOPS Operating Systems Review*, vol. 36, pp. 211–222, Oct. 2002.

[37] S. Das and H. K. Kapoor, "Victim retention for reducing cache misses in tiled chip multiprocessors," *Microprocessors and Microsystems*, vol. 38, no. 4, pp. 263–275, 2014.

[38] M. K. Qureshi, D. Thompson, and Y. N. Patt, "The v-way cache: Demand based associativity via global replacement," *SIGARCH Comput. Archit. News*, vol. 33, no. 2, pp. 544–555, May 2005.

[39] A. Bardine, P. Foglia, G. Gabrielli, and C. A. Prete, "Analysis of static and dynamic energy consumption in nuca caches: Initial results," in *Proceedings of the 2007 Workshop on MEmory Performance: DEaling with Applications, Systems and Architecture*, 2007, pp. 105–112.

[40] "Micron 1 GB DDR2 SDRAM Module Datasheet," http://www.micron.com, [Online].

[41] K. Roy, S. Mukhopadhyay, and H. Mahmoodi-Meimand, "Leakage current mechanisms and leakage reduction techniques in deep-submicrometer cmos circuits," *Proceedings of the IEEE*, vol. 91, no. 2, pp. 305–327, Feb 2003.

[42] S. Murugesan, "Harnessing green it: Principles and practices," *IT Professional*, vol. 10, no. 1, pp. 24–33, Jan 2008.

[43] J. S. Harper, D. J. Kerbyson, and G. R. Nudd, "Analytical modeling of set-associative cache behavior," *IEEE Transactions on Computers*, vol. 48, no. 10, pp. 1009–1024, Oct 1999.

[44] X. Vera, N. Bermudo, J. Llosa, and A. González, "A fast and accurate framework to analyze and optimize cache memory behavior," *ACM Trans. Program. Lang. Syst.*, vol. 26, no. 2, pp. 263–300, Mar. 2004.

[45] S. Chatterjee, E. Parker, P. J. Hanlon, and A. R. Lebeck, "Exact analysis of the cache behavior of nested loops," in *Proceedings of the ACM SIGPLAN*

2001 Conference on Programming Language Design and Implementation, ser. PLDI '01. New York, NY, USA: ACM, 2001, pp. 286–297.

[46] K. Vivekanandarajah, T. Srikanthan, and C. T. Clarke, "Profile directed instruction cache tuning for embedded systems," in *IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures (ISVLSI'06)*, March 2006, pp. 6 pp.–.

[47] S. Ghosh, M. Martonosi, and S. Malik, "Cache miss equations: A compiler framework for analyzing and tuning memory behavior," *ACM Trans. Program. Lang. Syst.*, vol. 21, no. 4, pp. 703–746, Jul. 1999.

[48] C. Xu, X. Chen, R. P. Dick, and Z. M. Mao, "Cache contention and application performance prediction for multi-core systems," in *Performance Analysis of Systems & Software (ISPASS), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 76–86.

[49] X. E. Chen and T. M. Aamodt, "A first-order fine-grained multithreaded throughput model," in *High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on*. IEEE, 2009, pp. 329–340.

[50] I. Gluhovsky and B. O'Krafka, "Comprehensive multiprocessor cache miss rate generation using multivariate models," *ACM Transactions on Computer Systems (TOCS)*, vol. 23, no. 2, pp. 111–145, 2005.

[51] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger, "Evaluation techniques for storage hierarchies," *IBM Systems journal*, vol. 9, no. 2, pp. 78–117, 1970.

[52] X. Xiang, B. Bao, T. Bai, C. Ding, and T. Chilimbi, "All-window profiling and composable models of cache sharing," in *ACM SIGPLAN Notices*, vol. 46, no. 8. ACM, 2011, pp. 91–102.

[53] W. Wang, P. Mishra, and S. Ranka, "Dynamic cache reconfiguration and partitioning for energy optimization in real-time multi-core systems," in

Proceedings of the 48th Design Automation Conference. ACM, 2011, pp. 948–953.

[54] W. Zhang, J. S. Hu, V. Degalahal, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin, "Compiler-directed instruction cache leakage optimization," in *Microarchitecture, 2002.(MICRO-35). Proceedings. 35th Annual IEEE/ACM International Symposium on*. IEEE, 2002, pp. 208–218.

[55] W. Zhang, M. Karakoy, M. Kandemir, and G. Chen, "A compiler approach for reducing data cache energy," in *Proceedings of the 17th annual international conference on Supercomputing*. ACM, 2003, pp. 76–85.

[56] H. S. Kim, N. Vijaykrishnan, M. Kandemir, E. Brockmeyer, F. Catthoor, and M. J. Irwin, "Estimating influence of data layout optimizations on sdram energy consumption," in *Proceedings of the 2003 International Symposium on Low Power Electronics and Design*, ser. ISLPED '03. New York, NY, USA: ACM, 2003, pp. 40–43.

[57] J. Lee and A. Shrivastava, "Pica: Processor idle cycle aggregation for energy-efficient embedded systems," *ACM Trans. Embed. Comput. Syst.*, vol. 11, no. 2, pp. 26:1–26:27, Jul. 2012.

[58] C. Ding and Y. Zhong, "Predicting whole-program locality through reuse distance analysis," in *ACM SIGPLAN Notices*, vol. 38, no. 5. ACM, 2003, pp. 245–257.

[59] D. Chandra, F. Guo, S. Kim, and Y. Solihin, "Predicting inter-thread cache contention on a chip multi-processor architecture," in *High-Performance Computer Architecture, 2005. HPCA-11. 11th International Symposium on*. IEEE, 2005, pp. 340–351.

[60] C. Ding and T. Chilimbi, "A composable model for analyzing locality of multi-threaded programs," Technical Report MSR-TR-2009-107, Microsoft Research, Tech. Rep., 2009.

[61] X. Shi, F. Su, J.-K. Peir, Y. Xia, and Z. Yang, "Modeling and stack simulation of cmp cache capacity and accessibility," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 12, pp. 1752–1763, 2009.

[62] R. Reddy and P. Petrov, "Cache partitioning for energy-efficient and interference-free embedded multitasking," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 9, no. 3, p. 16, 2010.

[63] W. Wang and P. Mishra, "Dynamic reconfiguration of two-level caches in soft real-time embedded systems," in *VLSI, 2009. ISVLSI'09. IEEE Computer Society Annual Symposium on.* IEEE, 2009, pp. 145–150.

[64] H. Noori, M. Goudarzi, K. Inoue, and K. Murakami, "Improving energy efficiency of configurable caches via temperature-aware configuration selection," in *IEEE Computer Society Annual Symposium on VLSI*, April 2008, pp. 363–368.

[65] M. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar, "Gated-vdd: A circuit technique to reduce leakage in deep-submicron cache memories," in *Proceedings of International Symposium on Low Power Electronics and Design*, 2000.

[66] K. Tanaka, "Cache memory architecture for leakage energy reduction," in *Proceedings of the Innovative Architecture for Future Generation High-Performance Processors and Systems*, ser. IWIA '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 73–80.

[67] S. Kaxiras, Z. Hu, and M. Martonosi, "Cache decay: exploiting generational behavior to reduce cache leakage power," in *Proceedings 28th Annual International Symposium on Computer Architecture*, 2001, pp. 240–251.

[68] A. Bardine, M. Comparetti, P. Foglia, G. Gabrielli, C. A. Prete, and P. Stenström, "Leveraging data promotion for low power d-nuca caches," in *Digital System Design Architectures, Methods and Tools, 2008. DSD'08. 11th EUROMICRO Conference on.* IEEE, 2008, pp. 307–316.

[69] S. Mittal and Z. Zhang, "Encache: Improving cache energy efficiency using a software-controlled profiling cache," *IEEE EIT*, 2012.

[70] ——, "Palette: A cache leakage energy saving technique for green computing," *Transition of HPC Towards Exascale Computing*, vol. 24, p. 46, 2013.

[71] S. Mittal, Z. Zhang, and Y. Cao, "Cashier: A cache energy saving technique for qos systems," in *VLSI Design and 2013 12th International Conference on Embedded Systems (VLSID), 2013 26th International Conference on*. IEEE, 2013, pp. 43–48.

[72] H. Zhou, M. C. Toburen, E. Rotenberg, and T. M. Conte, "Adaptive mode control: A static power efficient cache design," *ACM Transactions on Embedded Computing Systems*, vol. 2, no. 3, pp. 347–372, August 2003.

[73] M. A. Z. Alves, K. Khubaib, E. Ebrahimi, V. Narasiman, C. Villavieja, P. O. A. Navaux, and Y. N. Patt, "Energy savings via dead sub-block prediction," in *Computer Architecture and High Performance Computing (SBAC-PAD), 2012 IEEE 24th International Symposium on*. IEEE, 2012, pp. 51–58.

[74] J. Abella, A. González, X. Vera, and M. F. O'Boyle, "Iatac: a smart predictor to turn-off l2 cache lines," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 2, no. 1, pp. 55–77, 2005.

[75] K. T. Sundararajan, T. M. Jones, and N. P. Topham, "The smart cache: An energy-efficient cache architecture through dynamic adaptation," *International Journal of Parallel Programming*, vol. 41, no. 2, pp. 305 – 330, 2013.

[76] D. Benitez, J. C. Moure, D. Rexachs, and E. Luque, "A reconfigurable cache memory with heterogeneous banks," in *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2010, pp. 825–830.

[77] H. Kim and J. Kim, "A leakage-aware l2 cache management technique for producer–consumer sharing in low-power chip multiprocessors," *Journal of Parallel and Distributed Computing*, vol. 71, no. 12, pp. 1545–1557, 2011.

[78] M. Rawlins and A. Gordon-Ross, "On the interplay of loop caching, code compression, and cache configuration," in *Proceedings of the 16th Asia and South Pacific Design Automation Conference*. IEEE Press, 2011, pp. 243–248.

[79] G. Keramidas, P. Xekalakis, and S. Kaxiras, "Applying decay to reduce dynamic power in set-associative caches," *High Performance Embedded Architectures and Compilers*, pp. 38–53, 2007.

[80] K. Inoue, T. Ishihara, and K. Murakami, "Way-predicting set-associative cache for high performance and low energy consumption," in *Proceedings of the 1999 international symposium on Low power electronics and design*. ACM, 1999, pp. 273–275.

[81] S. Dropsho, A. Buyuktosunoglu, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas, G. Semeraro, G. Magklis, and M. Scottt, "Integrating adaptive on-chip storage structures for reduced dynamic power," in *Parallel Architectures and Compilation Techniques, 2002. Proceedings. 2002 International Conference on*. IEEE, 2002, pp. 141–152.

[82] M. Loghi, P. Azzoni, and M. Poncino, "Tag overflow buffering: Reducing total memory energy by reduced-tag matching," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 17, no. 5, pp. 728–732, 2009.

[83] Y. Guo, P. Narayanan, M. A. Bennaser, S. Chheda, and C. A. Moritz, "Energy-efficient hardware data prefetching," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 2, pp. 250–263, 2011.

[84] G. H. Loh, "3d-stacked memory architectures for multi-core processors," in *Computer Architecture, 2008. ISCA'08. 35th International Symposium on*. IEEE, 2008, pp. 453–464.

[85] A. Bardine, M. Comparetti, P. Foglia, G. Gabrielli, and C. A. Prete, "Way Adaptable D-NUCA Caches," *Int. J. High Perform. Syst. Archit.*, vol. 2, no. 3/4, pp. 215–228, Aug. 2010.

[86] Y.-Y. Tsai and C.-H. Chen, "Energy-efficient trace reuse cache for embedded processors," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 9, pp. 1681–1694, 2011.

[87] D. Jevdjic, S. Volos, and B. Falsafi, "Die-stacked dram caches for servers: hit ratio, latency, or bandwidth? have it all with footprint cache," in *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3. ACM, 2013, pp. 404–415.

[88] D. H. Albonesi, "Selective cache ways: On-demand cache resource allocation," in *Microarchitecture, 1999. MICRO-32. Proceedings. 32nd Annual International Symposium on.* IEEE, 1999, pp. 248–259.

[89] X. Wang, K. Ma, and Y. Wang, "Cache latency control for application fairness of differentiation in power-constrained chip multiprocessors," *IEEE Transactions on Computers*, vol. 61, no. 10, pp. 1371–1385, October 2012.

[90] S. Ramaswamy and S. Yalamanchili, "Improving cache efficiency via resizing + remapping," in *International Conference on Computer Design*, Oct 2007, pp. 47–54.

[91] A. M. Dani, B. Amrutur, and Y. N. Srikant, "Adaptive power optimization of on-chip snuca cache on tiled chip multicore architecture using remap policy," *Second Workshop on Architecture and Multi-Core Applications*, pp. 12–17, 2011.

[92] P. Foglia and M. Comparetti, "A workload independent energy reduction strategy for D-NUCA caches," *The Journal of Supercomputing*, vol. 68, pp. 157–182, Oct 2013.

[93] J.-J. Li and Y.-S. Hwang, "Snug set-associative caches: reducing leakage power while improving performance," in *Proceedings of the 2005 international symposium on Low power electronics and design.* ACM, 2005, pp. 345–350.

[94] S. Kaxiras, P. Xekalakis, and G. Keramidas, "A simple mechanism to adapt leakage-control policies to temperature," in *Proceedings of the 2005 international symposium on Low power electronics and design.* ACM, 2005, pp. 54–59.

[95] G. Memik, G. Reinman, and W. H. Mangione-Smith, "Just say no: Benefits of early cache miss determination," in *High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings. The Ninth International Symposium on.* IEEE, 2003, pp. 307–316.

[96] T. Austin, E. Larson, and D. Ernst, "SimpleScalar: an infrastructure for computer system modeling," *Computer*, vol. 35, no. 2, pp. 59–67, Feb. 2002.

[97] F. M. Sleiman, R. G. Dreslinski, and T. F. Wenisch, "Embedded way prediction for last-level caches," in *Computer Design (ICCD), 2012 IEEE 30th International Conference on.* IEEE, 2012, pp. 167–174.

[98] M. D. Powell, A. Agarwal, T. Vijaykumar, B. Falsafi, and K. Roy, "Reducing set-associative cache energy via way-prediction and selective direct-mapping," in *Proceedings of the 34th annual ACM/IEEE international symposium on Microarchitecture.* IEEE Computer Society, 2001, pp. 54–65.

[99] P. Carazo, R. Apolloni, F. Castro, D. Chaver, L. Pinuel, and F. Tirado, "L1 data cache power reduction using a forwarding predictor." in *PATMOS.* Springer, 2010, pp. 116–125.

[100] C. S. Ballapuram, A. Sharif, and H.-H. S. Lee, "Exploiting access semantics and program behavior to reduce snoop power in chip multiprocessors," *ACM SIGARCH Computer Architecture News*, vol. 36, no. 1, pp. 60–69, 2008.

[101] Z. Zhu and X. Zhang, "Access-mode predictions for low-power cache design," *IEEE micro*, vol. 22, no. 2, pp. 58–71, 2002.

[102] A. Bardine, M. Comparetti, P. Foglia, and C. A. Prete, "Evaluation of leakage reduction alternatives for deep submicron dynamic nonuniform cache architecture caches," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 1, pp. 185–190, Jan 2014.

[103] M. Ghosh, E. Ozer, S. Ford, S. Biles, and H.-H. S. Lee, "Way guard: a segmented counting bloom filter approach to reducing energy for set-associative caches," in *Proceedings of the 2009 ACM/IEEE international symposium on Low power electronics and design*. ACM, 2009, pp. 165–170.

[104] F. Xie, M. Martonosi, and S. Malik, "Bounds on power savings using runtime dynamic voltage scaling: an exact algorithm and a linear-time heuristic approximation," in *Proceedings of the 2005 International Symposium on Low Power Electronics and Design, 2005.*, Aug 2005, pp. 287–292.

[105] R. Rao, S. Vrudhula, C. Chakrabarti, and N. Chang, "An optimal analytical solution for processor speed control with thermal constraints," in *Proceedings of the International Symposium on Low Power Electronics and Design*, 2006, pp. 292–297.

[106] V. Hanumaiah, S. Vrudhula, and K. S. Chatha, "Performance optimal online dvfs and task migration techniques for thermally constrained multi-core processors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 11, pp. 1677–1690, Nov 2011.

[107] F. Zanini, D. Atienza, C. N. Jones, L. Benini, and G. De Micheli, "Online thermal control methods for multiprocessor systems," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 18, no. 1, pp. 6:1–6:26, Jan. 2013.

[108] H. Wang, J. Ma, S. X.-D. Tan, C. Zhang, H. Tang, K. Huang, and Z. Zhang, "Hierarchical dynamic thermal management method for high-performance many-core microprocessors," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 22, no. 1, pp. 1:1–1:21, Aug. 2016.

[109] W. Kim, M. S. Gupta, G. Y. Wei, and D. Brooks, "System level analysis of fast, per-core dvfs using on-chip switching regulators," in *IEEE 14th International Symposium on High Performance Computer Architecture*, Feb 2008, pp. 123–134.

[110] J. Lee and N. S. Kim, "Analyzing potential throughput improvement of power- and thermal-constrained multicore processors by exploiting dvfs and pcpg," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 2, pp. 225–235, Feb 2012.

[111] T. Chantem, R. P. Dick, and X. S. Hu, "Temperature-aware scheduling and assignment for hard real-time applications on mpsocs," in *Proceedings of the Conference on Design, Automation and Test in Europe*, 2008, pp. 288–293.

[112] Y. Ge, P. Malani, and Q. Qiu, "Distributed task migration for thermal management in many-core systems," in *Design Automation Conference*, June 2010, pp. 579–584.

[113] Y. Ge, Q. Qiu, and Q. Wu, "A multi-agent framework for thermal aware task migration in many-core systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 10, pp. 1758–1771, 2012.

[114] H. Mizunuma, Y. C. Lu, and C. L. Yang, "Thermal coupling aware task migration using neighboring core search for many-core systems," in *International Symposium on VLSI Design, Automation, and Test (VLSI-DAT)*, April 2013, pp. 1–4.

[115] L. Li, I. Kadayif, Y. F. Tsai, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, and A. Sivasubramaniam, "Leakage energy management in cache hierarchies," in *Proceedings.International Conference on Parallel Architectures and Compilation Techniques*, 2002, pp. 131–140.

[116] J. C. Ku, S. Ozdemir, G. Memik, and Y. Ismail, "Thermal management of on-chip caches through power density minimization," in *Proceedings of the 38th Annual IEEE/ACM International Symposium on Microarchitecture*, 2005.

[117] G. Sun, X. Wu, and Y. Xie, "Exploration of 3d stacked l2 cache design for high performance and efficient thermal control," in *Proceedings of International Symposium on Low Power Electronics and Design*, 2009.

[118] G. Sun, H. Yang, and Y. Xie, "Performance/thermal-aware design of 3d-stacked l2 caches for cmps," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 17, no. 2, Apr. 2012.

[119] H. Noori, M. Goudarzi, K. Inoue, and K. Murakami, "The effect of temperature on cache size tuning for low energy embedded systems," in *Proceedings of the 17th ACM Great Lakes Symposium on VLSI*, 2007.

[120] M. Farahani and A. Baniasadi, "Temperature reduction analysis in sentry tag cache systems," in *Proceedings of the 10th Workshop on MEmory Performance: DEaling with Applications, Systems and Architecture*, 2009, pp. 22–27.

[121] R. Ayoub and A. Orailoglu, "Performance and energy efficient cache migration approach for thermal management in embedded systems," in *Proceedings of the 20th Symposium on Great Lakes Symposium on VLSI*, 2010, pp. 365–368.

[122] Y. Zhang, L. Li, Z. Lu, A. Jantsch, M. Gao, H. Pan, and F. Han, "A survey of memory architecture for 3d chip multi-processors," *Microprocess. Microsyst.*, vol. 38, no. 5, pp. 415–430, Jul. 2014.

[123] S. Lee, K. Kang, and C. M. Kyung, "Runtime thermal management for 3-d chip-multiprocessors with hybrid sram/mram l2 cache," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 3, pp. 520–533, March 2015.

[124] R. Jain, "The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling," *SIGMETRICS Performance Evaluation Review*, vol. 19, pp. 5–11, 1991.

[125] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A full system simulation platform," *Computer*, vol. 35, no. 2, pp. 50–58, Feb 2002.

[126] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet's general execution-driven multiprocessor simulator (gems) toolset," *SIGARCH Comput. Archit. News*, vol. 33, no. 4, pp. 92–99, 2005.

[127] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The Gem5 Simulator," *ACM SIGARCH Computer Architure News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[128] N. Binkert, R. Dreslinski, L. Hsu, K. Lim, A. Saidi, and S. Reinhardt, "The M5 Simulator: Modeling Networked Systems," *IEEE Micro*, vol. 26, no. 4, pp. 52–60, Jul. 2006.

[129] M. Rosenblum and M. Varadarajan, "SimOS: A Fast Operating System Simulation Environment," Stanford, CA, USA, Tech. Rep., 1994.

[130] L. Schaelicke and M. Parker, "The Design and Utility of the ML-RSIM System Simulator," *Journal of Systems Architecture*, vol. 52, no. 5, pp. 283–297, May. 2006.

[131] S. K. Sastry Hari, M.-L. Li, P. Ramachandran, B. Choi, and S. V. Adve, "mSWAT: Low-cost Hardware Fault Detection and Diagnosis for Multicore Systems," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2009, pp. 122–132.

[132] J. An, X. Fan, S. Zhang, and D. Wang, "An Efficient Verification Method for Microprocessors Based on the Virtual Machine," in *Proceedings of the 1st International Conference on Embedded Software and Systems*, 2005, pp. 514–521.

[133] E. Argollo, A. Falcón, P. Faraboschi, M. Monchiero, and D. Ortega, "COTSon: Infrastructure for Full System Simulation," *ACM SIGOPS Operating Systems Review*, vol. 43, no. 1, pp. 52–61, Jan. 2009.

[134] N. Hardavellas, S. Somogyi, T. F. Wenisch, R. E. Wunderlich, S. Chen, J. Kim, B. Falsafi, J. C. Hoe, and A. G. Nowatzyk, "SimFlex: A Fast, Accurate, Flexible Full-system Simulation Framework for Performance Evaluation of Server Architecture," *ACM SIGMETRICS Performance Evaluation Review*, vol. 31, no. 4, pp. 31–34, Mar. 2004.

[135] F. J. Ridruejo, J. Miguel-Alonso, and J. Navaridas, "Full-system Simulation of Distributed Memory Multicomputers," *Cluster Computing*, vol. 12, no. 3, pp. 309–322, Sep. 2009.

[136] J. L. Henning, "Spec cpu2006 benchmark descriptions," *SIGARCH Comput. Archit. News*, vol. 34, no. 4, pp. 1–17, Sep. 2006.

[137] N. Agarwal, T. Krishna, L. S. Peh, and N. K. Jha, "Garnet: A detailed on-chip network model inside a full-system simulator," in *2009 IEEE International Symposium on Performance Analysis of Systems and Software*, April 2009, pp. 33–42.

[138] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," in *Proceedings of the INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE*, 1995, pp. 24–36.

[139] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Ferret: a toolkit for content-based similarity search of feature-rich data," *ACM SIGOPS Operating Systems Review*, vol. 40, no. 4, pp. 317–330, 2006.

[140] M. Müller, D. Charypar, and M. Gross, "Particle-based fluid simulation for interactive applications," in *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*. Eurographics Association, 2003, pp. 154–159.

[141] G. Grahne and J. Zhu, "Fast algorithms for frequent itemset mining using fp-trees," *IEEE transactions on knowledge and data engineering*, vol. 17, no. 10, pp. 1347–1362, 2005.

[142] D. Heath, R. Jarrow, and A. Morton, "Bond pricing and the term structure of interest rates: A discrete time approximation," *Journal of Financial and Quantitative Analysis*, vol. 25, no. 4, pp. 419–440, 1990.

[143] K. Martinez and J. Cupitt, "Vips-a highly tuned image processing software architecture," in *Image Processing, 2005. ICIP 2005. IEEE International Conference on*, vol. 2. IEEE, 2005, pp. II–574.

[144] N. S. Kim, T. M. Austin, D. Blaauw, T. N. Mudge, K. Flautner, J. S. Hu, M. J. Irwin, M. T. Kandemir, and N. Vijaykrishnan, "Leakage Current: Moore's Law Meets Static Power," *IEEE Computer*, vol. 36, no. 12, pp. 68–75, 2003.

[145] L. Li, I. Kadayif, Y.-F. Tsai, N. Vijaykrishnan, M. Kandemir, M. Irwin, and A. Sivasubramaniam, "Leakage energy management in cache hierarchies," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, 2002, pp. 131–140.

[146] M. Martonosi, S. Malik, and F. Xie, "Efficient behavior-driven runtime dynamic voltage scaling policies," in *Third IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS'05)*, Sept 2005, pp. 105–110.

[147] B. Salami, M. Baharani, and H. Noori, "Proactive task migration with a self-adjusting migration threshold for dynamic thermal management of multi-core processors," *J. Supercomput.*, vol. 68, no. 3, pp. 1068–1087, Jun. 2014.

[148] A. K. Coskun, T. S. Rosing, and K. Whisnant, "Temperature aware task scheduling in mpsocs," in *Proceedings of the Conference on Design, Automation and Test in Europe*, 2007, pp. 1659–1664.

[149] [Online]. Available: https://datasheets.chipdb.org/Sun/UltraSparc-IIIi.pdf

Publications Related to Thesis

Journal(s):

- **Shounak Chakraborty**, and Hemangee K. Kapoor, "Performance linked Dynamic Cache Tuning: A Static Energy Reduction Approach in Tiled CMPs," *Journal of Microprocessors and Microsystems (Elsevier)*, Volume 52, July 2017, Pages 221-235.

Under Review:

- "Analysing the Role of Last Level Caches in Controlling Chip Temperature". [under the second phase of revision]
- "Exploring the Role of Centralised Large LLCs in Thermal Efficient Chip Design".

Conferences:

- **S. Chakraborty**, and H. K. Kapoor, "Towards Controlling Chip Temperature by Dynamic Cache Reconfiguration in Multiprocessors"-*30^{th} International Conference on VLSI Design (VLSID)*, 2017, pp. 75-80, Hyderabad, India.
- **S. Chakraborty**, and H. K. Kapoor, "Static Energy Reduction by Performance Linked Dynamic Cache Resizing"-*IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, 2016, pp. 1-6, Tallinn, Estonia.
- **S. Chakraborty**, S. Das and H. K. Kapoor, "Performance constrained static energy reduction using way-sharing target-banks", *IEEE International Parallel and Distributed Processing Symposium Workshop (IPDPSW '15)*, 2015, pp. 444-453, Hyderabad, India.

- H. K. Kapoor, S. Das and **S. Chakraborty**, "Static energy reduction by performance linked cache capacity management in Tiled CMPs"-*Proceedings of the 30th Annual ACM Symposium on Applied Computing (SAC '15)*, 2015, pp. 1913–1918, Salamanca, Spain.

Other Publications of the Author

Conference:

- **S. Chakraborty**, S. Das and H. K. Kapoor, "Static Energy Efficient Cache Reconfiguration for Dynamic NUCA in Tiled CMPs"-*Proceedings of the 31st Annual ACM Symposium on Applied Computing (SAC '16)*, 2016, pp. 1739–1744, Pisa, Italy.

- **S. Chakraborty**, S. Das and H. K. Kapoor, "Power Aware Cache Miss Reduction by Energy Efficient Victim Retention"- *19th International Symposium on VLSI Design and Test (VDAT '15)*, 2015, pp. 1-6, Ahmedabad, India.

- N. K. Meena, H. K. Kapoor, **S. Chakraborty**, "A New Recursive Partitioning Multicast Routing Algorithm for 3D Network-on-Chip". *18th International Symposium on VLSI Design and Test (VDAT '14)*, 2014, pp. 1-6, Coimbatore, India.

- A. Kulkarni, **S. Chakraborty**, S. P. Mahajan, H. K. Kapoor, "Utility Aware Snoozy Caches for Energy Efficient Chip Multi-Processors", *The 28th edition of the ACM Great Lakes Symposium on VLSI (GLSVLSI)*, Chicago, Illinois, USA, May 23-25, 2018. (accepted)