# CoreMemDTM: Integrated Processor Core and 3D Memory Dynamic Thermal Management for Improved Performance

Lokesh Siddhu
*Dept. of Comp. Sc. and Engg.*
*Indian Institute of Technology Delhi*
*Email: siddhulokesh@cse.iitd.ac.in*

Rajesh Kedia
*Khosla School of IT*
*Indian Institute of Technology Delhi*
*Email: rajeshkedia@acm.org*

Preeti Ranjan Panda
*Dept. of Comp. Sc. and Engg.*
*Indian Institute of Technology Delhi*
*Email: panda@cse.iitd.ac.in*

***Abstract*—The growing performance of processors and 3D memories has resulted in higher power densities and temperatures. Dynamic thermal management (DTM) policies for processor cores and memory have received significant research attention, but existing solutions address processors and 3D memories independently, which causes overcompensation, and there is a need to coordinate the DTM of the two subsystems. Further, existing CPU DTM policies slow down heated cores significantly, increasing the overall execution time and performance overheads. We propose *CoreMemDTM*, a technique for integrating processor core and 3D memory DTM policies that attempts to minimize performance overheads. We suggest employing DTM depending on the thermal margin since safe temperature thresholds might differ for the two subsystems. We propose a stall-balanced core DVFS policy for core thermal management that enables distributed cooling, decreasing overheads. We evaluate *CoreMemDTM* using ten different SPEC CPU2017 workloads across various safe temperature thresholds and observe average execution time and energy improvements of 14% and 36% compared to state-of-the-art DTM policies.**

## I. Introduction

With emerging applications requiring higher compute and memory performance, the number of processor cores as well as bandwidth/stacking of memory is increasing [1], [2]. While this has helped achieve faster processing, it has resulted in increased power density [3] and causes the temperature to rise beyond a specified safe operating limit [4], thereby making thermal management an indispensable part of today's computing systems [5]. Thermal management incurs performance and energy overheads, reducing which is a significant problem requiring research attention [5]–[9].

Traditionally, the power dissipation (and temperature) of processors was higher than that of the main memory. Hence many prior works addressed thermal issues only for cores [5]–[12]. With the emergence of 3D DRAMs such as High Bandwidth Memory (HBM) and Hybrid Memory Cube (HMC), power density has increased significantly, causing thermal management in memories to gain research attention [1], [13], [14]. Higher bandwidth in 3D memory enables cores to execute faster, exacerbating thermal issues in cores. Related research has focused exclusively on either core or memory, and typically ignored thermal constraints on the other subsystem. However, in reality, as shown in Figure 1, both processor and memory would have temperature limits for safe operation ($T_P$ and $T_M$, respectively), and considering them separately (neglecting their dependencies) leads to additional overhead (Section III).

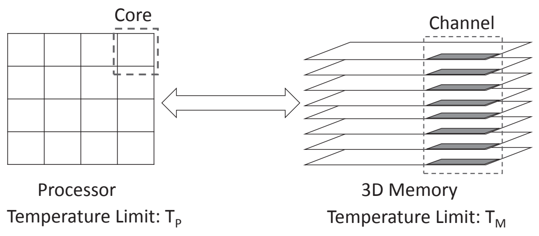


Fig. 1. Processor and 3D memory with thermal limits

We propose *CoreMemDTM*, an approach for joint dynamic thermal management (DTM) of processor cores and memory in a computing system. *CoreMemDTM* is built upon the premise that due to inter-dependence between core and memory, a DTM decision taken for one of them might alleviate the temperature for the other, and therefore, reduce overheads. *CoreMemDTM* uses a multi-level slack-balanced Dynamic Voltage Frequency Scaling (DVFS) technique for DTM of cores (*CoreDTM*) and low power states for DTM of memory (*MemDTM*). When either the core or memory is heated, *CoreMemDTM* invokes the respective DTM policy. In case both core and memory components get heated, *CoreMemDTM* invokes DTM for the component with a lower thermal slack (Section IV).

We evaluate *CoreMemDTM* using ten different workloads consisting of different mixes of SPEC CPU2017 [15] benchmarks. We use four different combinations of safe temperature limits for core and memory. Our experiments suggest that compared to state-of-the-art DTM policies for core [5] and memory [14], *CoreMemDTM* reduces the application execution time by 14% and system energy consumption by 36% on an average across different workloads and temperature thresholds.

This paper makes the following key contributions:

1) We study, for the first time, the overcompensation and overheads resulting from independently running thermal management policies for core and memory.
2) We propose an integrated core and 3D memory DTM policy, *CoreMemDTM*, which uses a stall-balanced core DVFS (*CoreDTM*) and low power memory states-based DTM (*MemDTM*) to improve performance.

## II. Related Work

For the last two decades, thermal management of processor cores, and lately, thermal management of memories, has been an active research topic [1], [4]–[14], [16]. DTM of cores includes techniques such as DVFS [4], [5] and temperature-aware mapping of tasks on cores [6], [11]. Iranfar et al. [7] manage core temperature using a reinforcement learning-based approach to control fan speed, active cores, and frequency. Hameed et al. [10] manage 3D core temperature by dynamically adapting core resources and using DVFS. For 3D multiprocessors, Kumar et al. [16] present a detailed survey on various thermal management techniques. Kim et al. [4] use DVFS to adjust core frequency based on the temperature. Noltsis et al. [5] improve DTM performance through a Proportional-Integral-Derivative (PID) controller to identify DVFS knobs. These works focus on thermal management only for the core and do not consider thermal limits for memory.

With 3D memories such as HMC and HBM becoming popular, researchers have identified thermal issues in memory and proposed DTM techniques [1], [13], [14]. Lo et al. [1] manage HMC temperature by allocating new page requests to a cold memory region. Siddhu et al. [14] propose a channel-level DTM strategy which migrates data from a hot 3D memory channel to a 2D memory and turns off power to the channel; this saves both leakage and dynamic power, enabling faster cooling of 3D memory. However, these works consider only memory DTM without accounting for the core thermal limits.

While prior works [17] focus on reducing the total energy consumption of core and memory, joint thermal management of core and memory has not been considered earlier.

## III. Motivation

Consider a multi-core system with a 3D main memory executing various compute-intensive and memory-intensive applications, which cause significant power dissipation in cores and memory, respectively, leading to heating and frequent activation of the corresponding DTM policies independently. DTM policy for core slows down the heated (compute-intensive) cores while memory DTM policy reduces the memory temperature by slowing accesses to heated (memory-intensive) channels. This, in turn, slows down the cores executing memory-intensive applications, reducing core temperature further and causing overcompensation as the DTM for core did not account for the additional core stalling (and cooling) due to memory DTM.

Figure 2 shows an example of a quad-core processor interacting with a 4-channel 3D memory. In order to reduce interference at the memory, we consider that each core accesses data of its corresponding 3D memory channel, similar to prior works [14], [18]. Compute-intensive applications execute on Cores 0 and 3 and memory-intensive applications on Cores 1 and 2, causing Cores 0 and 3 and memory Channels 1 and 2 to get heated and require cooling. The DTM for core slows down Cores 0 and 3, while DTM for memory slows down Channels 1 and 2. Since cores fetch data from memory, slowing down Channels 1 and 2 is effected essentially by slowing down Cores 1 and 2, resulting in all four cores being slowed down. This leads to inefficient thermal management as the core DTM was unaware of the actions of the memory DTM. The above observation motivates us to reduce DTM overheads with joint thermal management of core and memory.

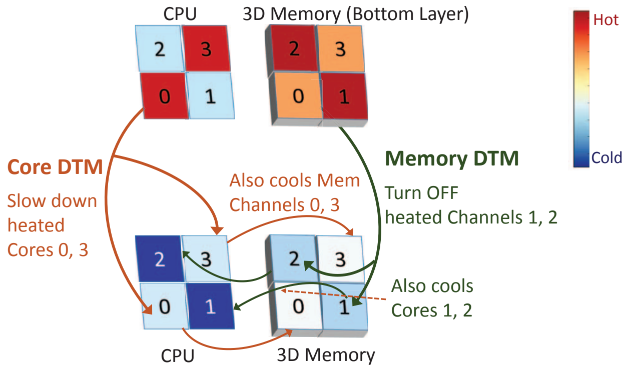


Fig. 2. Thermal map for baseline DTM policy. Applying core and memory DTM independently causes overcompensation, as it neglects dependencies.

## IV. *CoreMemDTM*: Proposed DTM Approach

*CoreMemDTM* combines a core DTM policy (*CoreDTM*, Section IV-A) and a memory DTM policy (*MemDTM*, Section IV-B) to manage core and memory temperatures and reduce performance overhead under thermal constraints.

### A. *CoreDTM: Multi-level Stall-Balanced DTM for Core*

DTM policies for core slow down/stall the heated cores using Dynamic Voltage Frequency Scaling (DVFS), incurring significant execution time overhead. Our policy is invoked at regular intervals (epochs). Every $K^{\text{th}}$ epoch, the coarse-grained multi-level DVFS strategy is invoked which helps quickly reduce the temperature by appropriately setting the frequency (and voltage) of heated cores. At every epoch, if the temperature crosses the DTM threshold, stall balancing is invoked which reduces the frequency (and voltage) of $S$ cores by one level in a rotating manner. The DVFS frequency for the core is decided based on its maximum temperature in the current $K$-epoch interval, while the number of cores ($S$) for stall balancing is determined based on the maximum and average temperatures of all cores in the current epoch. The above strategy, referred to as *CoreDTM*, maintains the core temperature below the safe temperature limit of the processor ($T_P$) by appropriately setting the core frequency ($f$) and power state ($CoreState$).

Algorithm 1 describes the *CoreDTM* policy in more detail. We first obtain the maximum temperature for each core ($T_{DVFS}$) in the current $K$-epoch interval. We also obtain the maximum temperature ($T_{max}$) among different cores for the current epoch (Line 3). If the maximum temperature at any epoch exceeds the safe temperature limit ($T_P$), we immediately place all cores in low power state and stall them. Otherwise, after every $K$ epochs, the multi-level DVFS policy is executed (Lines 12-16) which sets the frequency/voltage of each core based on $T_{DVFS}$ and $T_P$. The $get\_freq$ function first computes the difference between $T_{DVFS}$ and $T_P$ and accordingly determines the core frequency using a table, with a lower frequency being used if the difference is smaller. The *CoreDTM* policy performs stall balancing at every epoch (Lines 17-27). We first set the frequency of cores as determined by the DVFS (Lines 17-18). We also determine the number of cores ($S$) for stall balancing using the maximum temperature of the epoch ($T_{max}$), average

**Algorithm 1:** *CoreDTM* policy

```
Input: $N_c$: total number of cores
Input: $T[0 : N_c - 1]$: current temperatures of cores
Input: $T_P$: safe temperature limit for processor
Input: $t$: current epoch
Input: $K$: DVFS interval (in terms of epochs)
Output: $CoreState[0 : N_c - 1]$: state of cores
Output: $f[0 : N_c - 1]$: frequency of cores
Globals: $T_{DVFS}, f_{DVFS}, last$

// Max. temp. of current $K$-epoch interval
1  for $i = 0;\ i < N_c;\ i = i + 1$ do
2      $T_{DVFS}[i] = max(T_{DVFS}[i], T[i])$
3  $T_{max} = max(T)$ // Current max. temp.
4  if $T_{max} >= T_P$ then // Safety limit reached
5      for $i = 0;\ i < N_c;\ i = i + 1$ do
6          $CoreState[i] = LOW\_POWER$
7          $f[i] = 0$ // All cores in Low Power
8      return $CoreState$, $f$
9  else // Below safety limit.
10     for $i = 0;\ i < N_c;\ i = i + 1$ do
11         $CoreState[i] = ACTIVE$
12 if $t\%K == 0$ then
13     // Multi-level DVFS every K epochs
14     for $i = 0;\ i < N_c;\ i = i + 1$ do
15         $f_{DVFS}[i] = get\_freq(T_{DVFS}[i], T_P)$
16         $T_{DVFS}[i] = 0$
// Stall balancing (SB) DVFS every epoch
17 for $i = 0;\ i < N_c;\ i = i + 1$ do
18     $f[i] = f_{DVFS}[i]$ // Init $f$
19 $T_{avg} = avg(T)$ // Current Avg. temp.
20 $S = get\_SB\_count(T_{max}, T_{avg}, T_P)$
21 for $i = 1;\ i \leq N_c$ and $S > 0;\ i = i + 1$ do
22     $j = (last + i) \mod N_c$
23     $downgrade\_freq\_by\_1(f[j])$
24     $cnt = cnt + 1$
25     if $cnt == S$ then
26         $last = j$ // Global
27         return $CoreState$, $f$
// Determine number of cores ($S$) for stall
   balancing
Function $get\_SB\_count\,(T_{max}, T_{avg}, T_P)$
28     if $T_P - T_{max} < \theta_2^C$ then
29         $S = S2$ // $S2 > S1 > S0$ are constants
30     else if $T_P - T_{max} < \theta_1^C$ then
31         $S = S1$ // $\theta_2^C < \theta_1^C$
32     else
33         $S = S0$
34     if $T_{max} - T_{avg} > \alpha$ then
           $S = S - \beta$
       return $S$
```

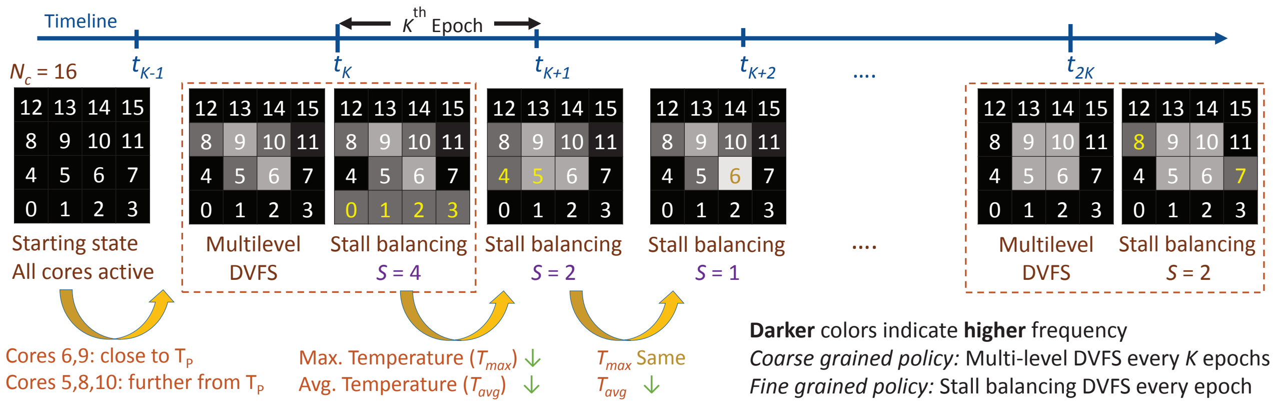
temperature of the epoch ($T_{avg}$), and the safe temperature limit ($T_P$), inside the $get\_SB\_count$ function (Line 28). $S$ is larger if $T_{max}$ is closer to $T_P$. As shown in Lines 28-33, $S$ is assigned one of the three values (i.e., $S0, S1$, or $S2$) depending upon the difference between $T_P$ and $T_{max}$ and pre-defined constants $\theta_1^C$ and $\theta_2^C$. Here $S2 > S1 > S0$ and $\theta_2^C < \theta_1^C$. $T_{max}$ can be high even if one or a few cores have high temperatures. In such cases, since many cores are at a lower temperature, we could reduce the aggressiveness of stall balancing. If $T_{max}$ is larger than $T_{avg}$ by $\alpha$ (indicating only a few cores are heated), the stall balancing core count is decreased by $\beta$ (Line 34). Once $S$ is identified, we reduce the frequency of $S$ cores by one level, in a rotating manner, starting from the core that was last considered for stall balancing (Lines 21-27). The parameters $S0$–$S2$, $\theta_1^C$, $\theta_2^C$, $\alpha$, and $\beta$ are determined empirically (Section V-A).

Figure 3 illustrates the operation of *CoreDTM* for a 16-core processor at various time steps. The timeline is indicated on the top, followed by the core frequency states (darker colors indicate higher frequency). At $t_{K-1}$, all the cores are running at the maximum frequency. At $t_K$ and $t_{2K}$ (i.e., every K epochs), the multi-level DVFS is invoked for coarse-grain adjustments. At $t_K$, it reduces the frequency slightly for Cores 5, 8, and 10 (as their temperature difference from $T_P$ is larger) and significantly for Cores 6 and 9 (as their temperatures are close to $T_P$). The stall balancing is invoked every epoch. At $t_K$, $S$ is determined to be 4, and therefore, the frequency of Cores 0, 1, 2, and 3 (indicated in yellow) is downgraded by 1. At $t_{K+1}$, $T_{max}$ and $T_{avg}$ reduce, resulting in $S$ = 2. Therefore, two cores, Cores 4 and 5 are downgraded in their frequency (while Cores 0, 1, 2, and 3 are restored to their corresponding frequencies determined during the recent DVFS). At $t_{K+2}$, $T_{avg}$ falls further, thereby reducing $S$ value to 1 (even if $T_{max}$ remains the same). The policy places Core 6 at lower frequency. At $t_{2K}$, multi-level DVFS is invoked again, the frequency of Cores 5, 6, 9, and 10 is lowered, and $S$ is determined to be 2. In the stall balancing step at $t_{2K}$, the frequency of Cores 7 and 8 is downgraded by 1. Such an operation with DVFS and stall balancing continues until the completion of the workload.

### B. MemDTM: Low Power State based DTM for Memory

DTM for 3D memories requires reducing both dynamic and leakage power [14]. We propose *MemDTM*, which places heated memory channels (and their neighbours) in low power state to reduce leakage power dissipation and manage the memory temperature, unlike prior works [14] which turn off heated memory channels incurring data migration overheads. During low power state, the memory data is retained, eliminating data migration overheads and the need to have a backup memory. Data is inaccessible during low power state, and the respective cores need to stall until the channels return to the active state.

*MemDTM* considers $T_M$ as the safe temperature limit for memory and runs every epoch, placing a certain number of channels in low power state. $L_1$, $L_2$, $L_3$, and $L_4$ ($L_i < L_{i+1}$) number of channels are placed in low power state when the memory temperature crosses thresholds $\theta_1^M$, $\theta_2^M$, $\theta_3^M$, and $\theta_4^M$ ($\theta_i^M < \theta_{i+1}^M$ and $\theta_4^M = T_M$), respectively. When the temperature exceeds $\theta_i^M$, we select $L_i$ channels for placing in low power



Timeline
$K^{th}$ Epoch
$N_c$ = 16
$t_{K-1}$
$t_K$
$t_{K+1}$
$t_{K+2}$
$t_{2K}$
Starting state
All cores active
Multilevel DVFS
Stall balancing S = 4
Stall balancing S = 2
Stall balancing S = 1
Multilevel DVFS
Stall balancing S = 2
Cores 6,9: close to $T_P$
Cores 5,8,10: further from $T_P$
Max. Temperature ($T_{max}$) ↓
Avg. Temperature ($T_{avg}$) ↓
$T_{max}$ Same
$T_{avg}$ ↓
Darker colors indicate higher frequency
Coarse grained policy: Multi-level DVFS every K epochs
Fine grained policy: Stall balancing DVFS every epoch

Fig. 3. An example *CoreDTM* illustration, showing core frequencies at different epochs

TABLE I
*CoreMemDTM* POLICY

| Core Heated?[1] | Memory Heated?[1] | DTM Action |
|---|---|---|
| No | No | No DTM. Normal Operation. |
| Yes | No | Apply *CoreDTM* (Section IV-A). For cores in low power state, place corresponding memory channels in low power state. |
| No | Yes | Apply *MemDTM* (Section IV-B). Additionally, for memory channels in low power state, place corresponding cores in low power state. |
| Yes | Yes | Compute temperature slack based on current temperature and safe temperature limit. If 3D memory has lower slack (or margin), then apply *MemDTM*, else apply *CoreDTM*. |

[1] Heated implies the current maximum temperature is greater than the defined DTM threshold.

state as follows. We first obtain a list of channels in decreasing order of temperature. We place the first channel from this list and its hottest adjacent channel. Then we select the next channel and continue the process for a total of $L_i$ channels. The rest of the channels are maintained in active state.

### C. *CoreMemDTM: Integrated DTM for Core and Memory*

*CoreMemDTM* integrates *CoreDTM* and *MemDTM* to formulate an integrated DTM policy that considers the inter-dependence between core and memory. Table I describes the *CoreMemDTM* policy. Operation proceeds normally if both core and memory temperatures are below the DTM threshold temperature. If either of them is heated, the respective DTM policy is activated. Alongside, if the DTM strategy places memory channels in low power state, the corresponding cores are also placed in the low power state (and vice-versa). If both cores and memory components are heated, we first compute the temperature slack (difference of current temperature and safe limit) and apply DTM only for the component with lower slack and expect that inter-dependence between core and memory would provide cooling to the other component. From our experiments and as reported in prior works [4], [19], both core DVFS and memory low power state-based policies are fast-acting (compared to the DTM epoch time, which is in milliseconds). Thus, if needed, *CoreMemDTM* could even be alternating between the policies at every epoch to control the temperature effectively (see Section V-B3).

## V. EXPERIMENTS AND RESULTS

### A. *Experimental Setup*

We evaluate *CoreMemDTM* using benchmarks from SPEC CPU2017 suite and create ten different workloads (Table II). These workloads contain different mixes of compute-intensive, memory-intensive, and mixed-type benchmarks, which cause varying core/memory temperature profiles. We use core and memory architecture configuration (Table III) similar to [14]. Using CoMeT toolchain [20], which internally uses the Sniper performance simulator, we simulate these workloads and obtain epoch-wise (1 ms) core power traces for four different frequencies (Table III). We also obtain a memory access trace from CoMeT and use the energy per access value obtained from CACTI-3DD to convert the memory access trace to memory power trace, similar to prior works [14]. For faster simulation of workloads, we use SPEC CPU2017 pinballs [15] for representative simulation regions (PinPoints). We use the core and memory power traces and execute two instances of HotSpot thermal simulator to separately obtain core and memory temperatures at every epoch and feed them to a DTM block which adjusts the frequency/state of cores and memory channels as per the chosen DTM policy. HotSpot parameters are reused from prior works [1], [14].

As the safe temperature limit could vary for different processors (even from the same series) [21] and typically range from 70 °C to 80 °C, we evaluate *CoreMemDTM* for four pairs of core ($T_P$) and memory ($T_M$) thermal limits – ($T_P$, $T_M$) = {(80, 80), (80, 70), (70, 80), (70, 70)} (in °C). We refer to these threshold combinations as C80_M80, C80_M70, C70_M80, and C70_M70, respectively, when describing the results. In our evaluation, we account for the reconfiguration overheads for the core DVFS (2000 CPU cycles, similar to [22]) and memory low power states (24 ns, similar to [19]). The reconfiguration overheads are minimal compared to the DTM epoch time (1 ms). For the *CoreDTM* policy, we use $\{S0, S1, S2\} = \{1, 2, 3\}$, $\{\theta_1^C, \theta_2^C\} = \{2, 1\}$, and $\alpha = \beta = 1$. We use the frequencies 3.6 (default), 3.0, 2.4, and 1.8 (in GHz) corresponding to temperature ranges $(0, T_P - 4.0)$, $(T_P - 4.0, T_P - 1.5)$, $(T_P - 1.5, T_P - 1.0)$, and $(T_P - 1.0, T_P)$ respectively. For the *MemDTM* policy, $\{\theta_1^M, \theta_2^M, \theta_3^M, \theta_4^M\} = \{T_M - 2.0,$

TABLE II
WORKLOADS AND BENCHMARKS

| Workload Name | Benchmark Details, Type |
|---|---|
| *x264* | x264(x16) – Compute |
| *exch* | exchange(x16) – Compute |
| *gcc* | gcc(x16) – Mixed |
| *nab* | nab(x16) – Mixed |
| *mcf* | mcf(x16) – Memory |
| *lbm* | lbm(x16) – Memory |
| *xxee* | x264(x8), exchange(x8) |
| *egnm* | exchange(x4), gcc(x4), nab(x4), mcf(x4) |
| *gnml* | gcc(x4), nab(x4), mcf(x4), lbm(x4) |
| *mmll* | mcf(x8), lbm(x8) |

TABLE III
ARCHITECTURE PARAMETERS

| Parameter | Value |
|---|---|
| Core Model | 16 cores, 22 nm, 3.6/3.0/2.4/1.8 GHz, 1.3/1.2/1.1/1.0 Volts, 84 entry ROB, 32 entry LSQ |
| L1 I/D Cache | 64 KB@2 ns, 2-way/64B-block |
| L2 Cache | Private, 512 KB@6 ns, 16-way/64B-block |
| 3D Memory Configuration | 1 GB, 16 channels, 8 ranks, 1 bank (per rank), closed page policy, 29 ns (latency), 7.6 GBps (per channel bandwidth) |

$T_M - 1.0, T_M - 0.5, T_M\}$ and $\{L_1, L_2, L_3, L_4\} = \{1, 2, 5, 16\}$. The above parameters are obtained empirically by varying the parameter and selecting the value corresponding to the lowest execution time on an average.

### B. Results and Discussion

Since there is no prior work implementing joint DTM of core and memory, we compare *CoreMemDTM* against a baseline that uses state-of-the-art DTM policies for core [5] and memory [14] executing independently. For DTM of cores, the baseline uses a Proportional-Integral-Derivative (PID) controller to adjust core frequency to closely track the safe temperature limit, similar to [5]. For memory DTM, the baseline turns off heated 3D memory channels and migrates their data to a 2D memory [14]. Both these policies operate independently every 1 ms.

*1) Execution Time Comparison:* Figure 4 shows the execution time of different workloads when using *CoreMemDTM*, normalized to that of the baseline. The workloads are shown in their increasing order of memory intensity. We observe that in most cases, *CoreMemDTM* leads to a lower execution time than the baseline. When the temperature thresholds are more stringent (C70_M70), *CoreMemDTM* provides higher performance benefits and results in an average performance improvement of 25% across workloads. Overall, across different temperature thresholds, *CoreMemDTM* results in a performance improvement of 14% (average) over the baseline.

The execution time improvement observed with *CoreMemDTM* can be attributed to various factors. First, we exploit the inter-dependence between core and memory, preventing overcompensation. Thus, *CoreMemDTM* yields higher benefits when memory and core have tighter thermal constraints (C70_M70). Second, we balance the slow down for different cores, due to which the overall workload completes faster. Stall balancing also leads to using a less aggressive DVFS, improving performance. Third, unlike the baseline, using low power state-based memory DTM, we avoid the data migration from 3D to 2D memory, thereby saving migration time (and energy consumption) overheads. We observe a slightly degraded performance for *x264* (Figure 4) when the core temperature constraint is 70 °C (C70_M80 and C70_M70) because *x264*

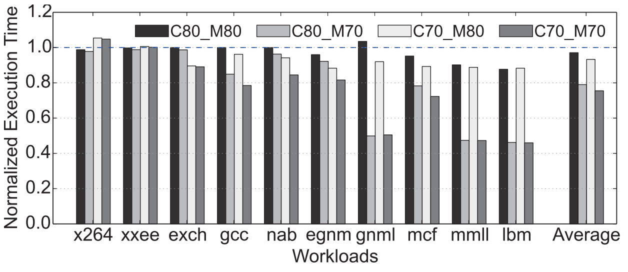


Fig. 4. Execution time normalized to baseline policy

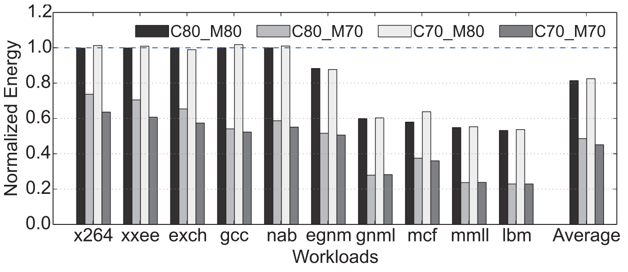


Fig. 5. Energy consumption normalized to baseline policy

causes a fast rise in the core temperature and requires more frequent DVFS rather than stall balancing.

*2) Energy Comparison:* Figure 5 shows the total (core + memory) energy consumption of different workloads and temperature limits when using *CoreMemDTM*, normalized to that of the baseline. *CoreMemDTM* results in lower energy consumption than the baseline. More memory-intensive workloads (shown towards the right in Figure 5) experience considerably higher benefits compared to others. Also, a tighter limit (70 °C) on memory temperature results in significant energy benefits. *CoreMemDTM* yields such energy benefits because it does not incur the large overheads of data migration for memory-intensive workloads or when the memory thermal limit is tighter. Further, *CoreMemDTM* places a core in low power state if its corresponding memory channel is placed in low power state for cooling, thus resulting in additional savings. We observe a slightly higher energy consumption for C70_M80 for a few workloads (*gcc*, *nab*). This is because *CoreMemDTM* avoids overcompensation and hence operates cores at a higher frequency (voltage) than the baseline, thereby consuming higher energy due to energy being a quadratic function of voltage ($E \propto C.V^2$). However, for these cases, *CoreMemDTM* reduces the execution time as it uses higher frequencies. Overall, across different temperature thresholds, *CoreMemDTM* yields an energy reduction of 36%, on average.

*3) Temperature Comparison:* Figure 6 shows the maximum core and memory temperatures at different time instants for *mcf* workload for scenarios when no DTM, baseline policy, or *CoreMemDTM* is applied, corresponding to a stringent thermal limit of C70_M70. Since *mcf* is a memory-intensive workload, the memory temperature rises significantly without DTM. Both the baseline policy and *CoreMemDTM* manage memory temperature below the limit. However, the baseline undergoes frequent heating and cooling cycles, whereas *CoreMemDTM* operates just below the limit. Due to the memory-intensive nature of the

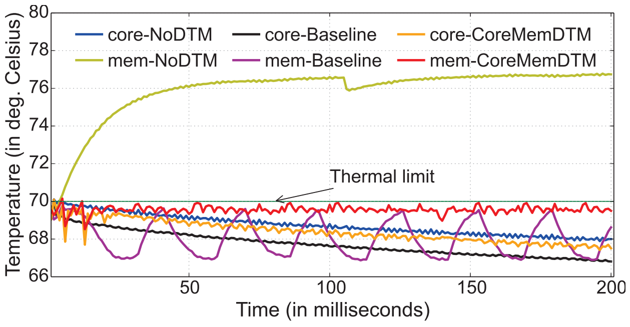


Fig. 6. Temperature-time trace for *mcf* for C70_M70

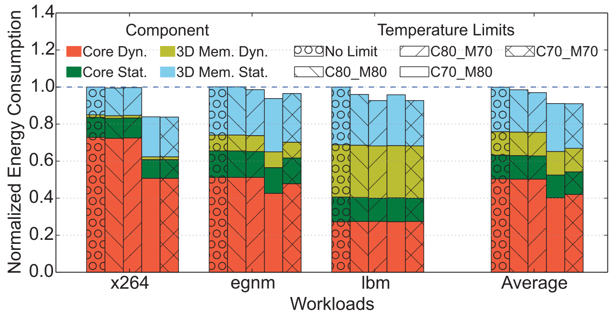


Fig. 7. Breakup of energy consumption for memory and core

workload, the core does not undergo excess heating, avoiding the need for core thermal management. The core temperature with *CoreMemDTM* is close to that of no DTM, but the core temperature with baseline policy is much lower as the PID controller-based DTM cannot stabilize due to inter-dependence between core and memory, and overcompensates.

*4) Performance and Energy Overheads of DTM:* We observe that *CoreMemDTM* causes an execution time overhead of 27% (on average) compared to an ideal case without any thermal constraints (details omitted due to lack of space). Figure 7 shows the split of dynamic and static energy for core and memory (for one workload of each type), normalized to the energy consumed when no thermal limit is imposed. The total energy consumption with DTM is always less than or equal to that without DTM. While this might seem counter-intuitive as DTM should add overheads, we identify that DTM causes cores to operate at a lower frequency and voltage, due to DVFS, compared to no DTM. This reduces energy consumption as energy is a quadratic function of voltage. For memory-intensive workloads (e.g., *lbm*), the 3D memory static energy reduces (other components are similar) with the use of DTM. This occurs because DTM causes the memory to operate at a lower temperature, reducing the temperature-dependent leakage power compared to no DTM. Overall, this study highlights that while DTM increases the execution time of workloads, it reduces the overall energy consumption.

## VI. Conclusion and Future Work

We identify that independent policies for core and memory thermal management incur higher overheads when both cores and memory operate under thermal constraints. To address such overheads, we propose *CoreMemDTM*, a joint DTM approach for processor cores and memory, which adjusts core state based on memory DTM and vice-versa. Our experiments, using ten different SPEC CPU2017 workloads, indicate that *CoreMemDTM* can reduce workload execution time and memory energy consumption by an average of 14% and 36%, respectively. In the future, we plan to extend this work for 2.5D/3D architectures where core and 3D memory are integrated on the same die and more closely influence each other's temperature.

## Acknowledgments

We thank Aritra Bagchi and Hameedah Sultan for useful suggestions on the manuscript. Cadence Design Systems provided financial support to Lokesh Siddhu.

## References

[1] W.-H. Lo, K.-z. Liang, and T. Hwang, "Thermal-aware dynamic page allocation policy by future access patterns for Hybrid Memory Cube (HMC)," in *DATE*, 2016.

[2] E. Azarkhish, D. Rossi, I. Loi, and L. Benini, "Neurostream: Scalable and energy efficient deep learning with smart memory cubes," *TPDS'17*.

[3] C.-H. Chou, D. Wong, and L. N. Bhuyan, "Dynsleep: Fine-grained power management for a latency-critical data center application," in *ISLPED'16*.

[4] J. M. Kim, Y. G. Kim, and S. W. Chung, "Stabilizing CPU frequency and voltage for temperature-aware DVFS in mobile devices," *TC*, 2015.

[5] M. Noltsis, N. Zambelis, F. Catthoor, and D. Soudris, "A closed-loop controller to ensure performance and temperature constraints for dynamic applications," *TECS*, 2019.

[6] Y. G. Kim, J. I. Kim, S. H. Choi, S. Young Kim, and S. W. Chung, "Temperature-aware adaptive VM allocation in heterogeneous data centers," in *ISLPED*, 2019.

[7] A. Iranfar *et al.*, "Dynamic thermal management with proactive fan speed control through reinforcement learning," in *DATE*, 2020.

[8] Y. Zhu *et al.*, "Countering variations and thermal effects for accurate optical neural networks," in *ICCAD*, 2020.

[9] Y. Cao, T. Shen, L. Zhang, X. Yin, and C. Zhuo, "An efficient and flexible learning framework for dynamic power and thermal co-management," in *MLCAD*, 2020.

[10] F. Hameed, M. A. A. Faruque, and J. Henkel, "Dynamic thermal management in 3D multi-core architecture through run-time adaptation," in *DATE*, 2011.

[11] A. K. Coskun, T. S. Rosing, and K. Whisnant, "Temperature aware task scheduling in MPSoCs," in *DATE*, 2007.

[12] W. Liu *et al.*, "Layout-driven post-placement techniques for temperature reduction and thermal gradient minimization," *TCAD*, 2013.

[13] M. H. Hajkazemi, M. K. Tavana, T. Mohsenin, and H. Homayoun, "Heterogeneous HMC+DDRx memory management for performance-temperature tradeoffs," *JETCS*, 2017.

[14] L. Siddhu, R. Kedia, and P. R. Panda, "Leakage-aware dynamic thermal management of 3D memories," *TODAES*, 2020.

[15] R. Panda, S. Song, J. Dean, and L. K. John, "Wait of a decade: Did SPEC CPU 2017 broaden the performance horizon?" in *HPCA*, 2018.

[16] S. S. Kumar, A. Zjajo, and R. van Leuken, "Fighting dark silicon: Toward realizing efficient thermal-aware 3-D stacked multiprocessors," *TVLSI'17*.

[17] J. Yahya *et al.*, "SysScale: Exploiting multi-domain dynamic voltage and frequency scaling for energy efficient mobile processors," in *ISCA*, 2020.

[18] S. P. Muralidhara, L. Subramanian, O. Mutlu, M. Kandemir, and T. Moscibroda, "Reducing memory interference in multicore systems via application-aware memory channel partitioning," in *MICRO*, 2011.

[19] Y. Lu *et al.*, "Rank-aware dynamic migrations and adaptive demotions for DRAM power management," *TC*, 2016.

[20] L. Siddhu *et al.*, "CoMeT: An integrated interval thermal simulation toolchain for 2D, 2.5D, and 3D processor-memory systems," *arXiv*, pp. 1–23, 9 2021. [Online]. Available: http://arxiv.org/abs/2109.12405

[21] Intel, "Thermal management for systems using Intel® Xeon® processors." [Online]. Available: https://www.intel.com/content/www/us/en/support/articles/000006710/processors/intel-xeon-processors.html

[22] R. Jain, P. R. Panda, and S. Subramoney, "Machine learned machines: Adaptive co-optimization of caches, cores, and on-chip network," in *DATE*, 2016.