Islands of Heaters: A Novel Thermal Management Framework for Photonic NoCs

*Dharanidhar Dang‡, *Sai Vineel Reddy Chittamuru†, Rabi Mahapatra‡, and Sudeep Pasricha†
‡Department of Computer Science and Engineering, Texas A&M University, College Station, TX, U.S.A.
†Department of Electrical and Computer Engineering, Colorado State University, Fort Collins, CO, U.S.A.
{d.dharanidhar, rabi}@tamu.edu,{sai.chittamuru, sudeep}@colostate.edu

***Abstract*— Silicon photonics has become a promising candidate for future networks-on-chip (NoCs) as it can enable high bandwidth density and lower latency with traversal of data at the speed of light. But the operation of photonic NoCs (PNoCs) is very sensitive to temperature variations that frequently occur on a chip. These variations can create significant reliability issues for PNoCs. For example, microring resonators (MRRs) which are the building blocks of PNoCs, may resonate at another wavelength instead of their designated wavelength due to thermal variations, which can lead to bandwidth wastage and data corruption in PNoCs. This paper proposes a novel run-time framework to overcome temperature-induced issues in PNoCs. The framework consists of (i) a PID controlled heater mechanism to nullify the thermal gradient across PNoCs, (ii) a device-level thermal island framework to distribute MRRs across regions of temperatures; and (iii) a system-level proactive thread migration technique to avoid on-chip thermal threshold violations and to reduce MRR tuning/trimming power by migrating threads between cores. Our experimental results with 64-core Corona and Flexishare PNoCs indicate that the proposed approach reliably satisfies on-chip thermal thresholds and maintains high network bandwidth while reducing total power by up to 64.1%.**

I. Introduction

The increasing core-density of chip-multiprocessors (CMP) requires high bandwidth to support extensive inter-core communication. Electrical NoCs cannot offer such a large bandwidth while maintaining an acceptable level of power dissipation [1]. On-chip photonic links provide several advantages over traditional metallic counterparts, such as near light speed data transfer, higher bandwidth density, and low power dissipation [2]. Moreover, photonic links have several times lower data-dependent energy consumption compared to electrical wires, enabling the design of high-radix networks that are easier to program [3]. Silicon photonics is thus becoming an exciting new option for on-chip communication, and has catalyzed much research in the area of photonic NoCs (PNoCs) for manycore systems [4]. However, photonic interconnects suffer from susceptibility to thermal fluctuations, which impacts correctness and performance. Hence, PNoCs are yet to be widely adopted.

Microring resonators (MRRs) and waveguides are the basic building blocks of a PNoC. MRRs are used as modulators/demodulators at the source/destination node. MRRs also perform photonic switching operations to route an optical signal in PNoCs. However, photonic components and especially MRRs are extremely susceptible to thermal fluctuations. Fig.1 depicts the impact of thermal variation on MRRs. MRRs R_1-R_n have been designed to resonate on wavelengths λ_1-λ_n respectively at temperature T_1. As the temperature increases, due to the resulting variations in refractive index, each MRR now resonates with a different wavelength towards the red end of the visible spectrum (i.e., red-shift). This red-shift is shown in the figure where, at temperature T_2, MRR R_i will now be in resonance with λ_{i-1}. This phenomenon reduces transmission reliability and results in wastage of available bandwidth, e.g., MRRs are unable to read or write to wavelength λ_n at temperature T_2.

Maintaining a uniform temperature across all the MRRs is a must for reliable data transmission in PNoCs. But thermal fluctuations and gradients are common in CMPs. 3D-ICE [5] simulations of PARSEC [6] and SPLASH-2 [7] benchmarks indicate a 15-20K peak thermal gradient in a 64-core CMP as shown in Fig. 2. Such a huge gradient causes a mismatch of resonant wavelengths of MRRs, leading to unreliable data transmission and PNoC performance degradation.

Recently, few techniques have been proposed to address thermal issues in PNoCs. At the *device-level*, a trimming mechanism is proposed in [8] that induces a blue shift (decrease) in the resonance wavelengths of MRRs using carrier injection. A tuning technique was demonstrated in [9] where a red-shift (increase) in the resonance wavelengths is induced by using a localized heater. Further several athermal photonic devices have been presented to reduce the localized tuning/trimming power in MRRs. These design time solutions include using cladding to reduce thermal sensitivity [10] and using heaters as well as temperature sensors for thermal control. While these device-level techniques are promising, they either possess a high power overhead or require costly changes in the manufacturing process (e.g., much larger device areas) that would decrease network bandwidth density and area efficiency. At the *system-level*, a thread migration framework was presented in [11] to avoid on-chip thermal threshold violations and also reduce trimming/tuning power for MRs. In [12], a ring aware thread scheduling policy was proposed to reduce on-chip thermal gradients in a PNoC. A proportional-integral-derivative (PID) heater mechanism was proposed in [13] that minimizes the effect of thermal variation on PNoC's performance and power. However, all these system-level techniques do not consider the impact of run-time workload variations and also result in considerable power performance overheads.

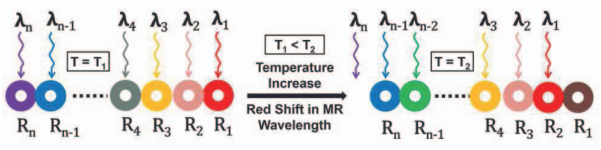


Fig. 1: Impact of thermal variations on MRRs.

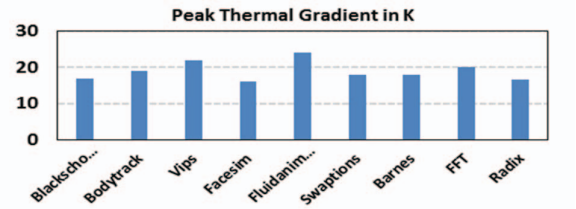


Fig. 2: Peak thermal gradient (in Kelvin) across a 64-core chip running 48-threaded PARSEC [6] and SPLASH-2 [7] benchmarks.

Our goal in this paper is to minimize thermal variations with reduced localized thermal tuning and trimming in PNoCs, thereby reducing key overheads and ultimately easing the adoption of PNoCs for future CMP systems. We propose a novel low-power thermal management framework that integrates an adaptive heater mechanism at the device-level and a dynamic thread migration scheme at the system-level. This paper makes the following contributions:

- A novel temperature island framework with adaptive heater based MRR to handle thermal gradients across PNoC;
- An islands of heaters based dynamic thread migration (IHDTM) scheme in conjunction with a support vector regression based temperature prediction mechanism. Such a scheme nullifies on-chip thermal threshold violations and also reduces trimming/tuning power for MRRs;
- The evaluation of the proposed framework on a 64-core CMP with a system-level simulator shows: (a) 70% improvement in trimming

* Dharanidhar Dang, Sai Vineel Reddy Chittamuru contributed equally to this work.

This research is supported in part by grants from SRC, NSF (CCF-1252500), and AFOSR (FA9550-13-1-0110). The authors are thankful to Luceda Photonics for their device-level experimental support.

978-1-5090-1558-0/17/$31.00 ©2017 IEEE

power dissipation over the most recent prior work, (b) 64.1% improvement in total power dissipation compared to a state-of-the-art thermal management technique, (c) 13.72K improvement in peak temperature, and (d) these improvements are achieved while maintaining full network-bandwidth.

The rest of the paper is organized as follows. Section II explains the proposed thermal management framework in detail. Experiments, results, and comparative analysis are demonstrated in Section III followed by conclusions in Section IV.

II. Islands Of Heaters Based Dynamic Thermal Management (IHDTM)

The proposed IHDTM framework enables variation-aware thermal management by integrating device-level and system-level enhancements. A high-level overview of the framework is shown in Fig. 3. At the device-level, the entire PNoC layer is divided into '*k*' regions or islands, namely: T_{IS1}*-island*, T_{IS2}*-island*, T_{IS3}*-island*, and so on. All MRRs in the T_{ISi}*-island* ($i \leq k$) are designed to operate at T_{ISi}; similarly, MRRs in the other islands are designed to operate at their respective temperatures. We use our device-level technique to overcome small deviations (±10K) in T_{ISi} whereas the system-level technique is used to adapt to larger variations (> ±10K). The device-level technique aims to adapt to the changing on-chip thermal profile, maintaining maximum bandwidth and correct MRR operation while minimizing trimming and tuning power in the PNoC. At the system-level, the dynamic thread migration scheme maintains acceptable core-temperatures for each island. The following sections explain the proposed (i) device-level island framework and (ii) system-level thread migration scheme in detail.

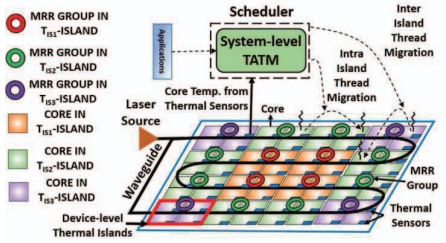
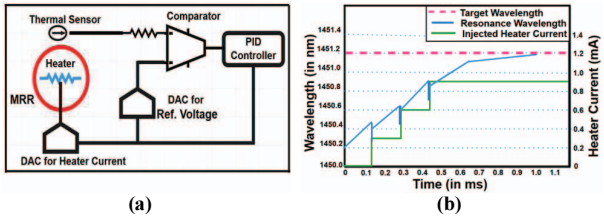


Fig. 3: IHDTM framework with device-level thermal islands and system-level temperature-aware thread migration mechanism (TATM)

A. Thermal Islands

The thermal distribution across a 64-core PNoC chip running PARSEC and SPLASH-2 benchmarks (using 3D-ICE simulation) shows three major zones of temperature: 363K, 343K, and 323K. Also, the average thermal gradient in the PNoC chip is found out to be approximately 15-20K. To reduce this gradient, the proposed device-level framework adopts three islands (as shown in Fig. 3) each of which are maintained at a unique temperature by assigning T_{IS1}, T_{IS2}, and T_{IS3} to 363K, 343K, and 323K respectively. As mentioned in the previous section, MRRs in the 363K-island (T_{IS1}*-island*) are designed to operate at 363K with a variation range of ±10K. MRRs employ thermal tuning and electrical trimming when they are operated below and above their designed temperatures respectively. Similarly, MRRs in other islands are designed to operate at the respective temperatures. For PNoCs of other sizes (e.g. 16-core, 25-core, 36-core, 128-core, 256-core), there can be slight variations in the number of islands and their respective temperature zones. Accordingly, the numbers of islands and their temperatures can be fixed at design time.



(a) (b)

Fig. 4: (a) MRR with adaptive heater (b) Thermal tuning of MRR

To manage localized temperature variation below designed temperature, each MRR is integrated with a PID controller [15] based heater as shown in Fig. 4(a). The PID controller is tuned with proportional band K_p*=50*, integral cycle-time K_i*=1* millisecond (ms), and derivative coefficient K_d*=0*. An open source PID tuning software [15] is used to determine optimal values of K_p, K_i, and K_d.

Algorithm 1: Thermal management of MRR

Input: Temperature (T) around the MRR detected by thermal sensor

//Controller converts T to appropriate heater current as follows:

1: $dT = |T_{island} - T|$

2: $P_{Heat} = \frac{dT}{\rho} \times H_{eff}$

3: ***if*** $(T \leq T_{island})$ ***then*** $i_{Heat} = i_{Max} - \sqrt{\frac{P_{Heat}}{R_{Heat}}}$

4: ***else*** $i_{Heat} = i_{Max} + \sqrt{\frac{P_{Heat}}{R_{Heat}}}$

Output: current (i_{Heat}) to be fed to heater

Algorithm 1 depicts the control algorithm for the heater in each MRR to stabilize thermal variations. In the algorithm, T represents the temperature across an MRR as detected by the corresponding thermal sensor, T_{island} is the fixed temperature of the island in which the MRR resides ($T_{island} = T_{ISi}$), P_{Heat} is the heater power, i_{Heat} represents heater current, and H_{eff} stands for the transfer function of the heater. With any local temperature change *dT*, there is an equivalent shift in resonance for the MRR. To undo this resonance shift in an MRR, an equivalent amount of heat must be radiated by the heater integrated with that MRR. As per the algorithm, the controller collects temperature data *T* from the local thermal sensor as input. In step 1, the absolute value of the difference between *T* and T_{island} is calculated followed by determining the required heater power P_{Heat} in step 2. *T* is compared with T_{island} in step 3 and accordingly the required heater current i_{Heat} is computed either in steps 3-4. The evaluated value of i_{Heat} is fed to the heater coil. This amount of current is needed by the heater to maintain the fixed temperature T_{island} around the MRR. Our analysis shows that a maximum of 1 ms of time is needed for the heater element to bring the surrounding temperature to the desired value of T_{island}. We account for this time delay in our simulations. Fig. 4(b) shows the tuning process of an MRR with injected heater current as explained in the algorithm. The control algorithm is invoked after every 1ms for each MRR.

This heater-based technique helps to stabilize thermal fluctuations in each temperature island with reduced tuning power. However, if the power footprint of a workload on a core associated with a 363K-island is very low, its core temperature may fall below the lower thermal limit (i.e. smaller than 353K). This thermal gradient can significantly increase tuning power consumption of an associated MRR. Similarly, if the power footprint of a workload on a core associated with a 323K-island keeps increasing beyond a threshold, then its core temperature might reach beyond the control of the MRR-trimmer (i.e. greater than 333K). This will in turn permanently shift the resonance of the MRR, inducing errors during communication. To address these issues, we propose a system-level temperature-aware thread migration (TATM) technique that performs thread migration to idle cores to maintain temperatures of corresponding MRRs close to the design-temperatures of their respective islands. By intelligently migrating threads, this technique reduces device-level tuning/trimming power in MRRs. TATM also aims to proactively reduce thermal hotspots, which in turn will reduce instances of irrecoverable drift in MRRs.

B. Temperature-Aware Thread Migration Scheme (TATM)

1) Objective: The primary goal with TATM is to maintain the temperature of all the cores in an island on a die below a specified thermal threshold (T_t) and above a thermal limit (T_l), i.e., for a core i in the T_{ISj}*-island*, $T_{lj} \leq T_i \leq T_{tj}$ where T_i is the temperature of core *i*, T_{tj} is threshold temperature of T_{ISj}*-island*, and T_{lj} is thermal limit of T_{ISj}*-island*. TATM maintains the core temperatures such that the temperature of all the MRRs within an island is close to their design temperature, to reduce tuning power consumption in adaptive heaters as explained in the previous section.

We utilize support vector based regression (SVR) to predict the future temperature of a core. This predicted temperature of a core is compared with the corresponding island's thermal threshold (upper limit) and thermal limit (lower limit) to determine the potential for a thermal emergency. If such a potential exists, then TATM initiates thread migration. Inter-island thread migration (Inter-island cores (*IEIC*)) is preferred over intra-island thread migration (Intra-island cores (*IAIC*)). This step has a twofold benefit. Firstly, by moving the thread away from a core that could suffer a thermal emergency, we avoid instances of irrecoverable drift in the MRR groups of that core. Secondly, by moving the thread to a core in different island, we ensure that the temperature of the island and its corresponding ring blocks remains between the island's thermal threshold (T_{t1}, T_{t2}, and T_{t3}) and thermal limit (T_{l1}, T_{l2}, and T_{l3}) to conserve trimming/tuning power. If a thermal emergency occurs due to exceeding the thermal threshold, then it is preferred that the thread is migrated to a core in an island whose MRR design temperature is higher. If a thermal emergency occurs due to temperature falling below the thermal limit then it is preferred that the thread is migrated to a core in an island whose MRR design temperature is lower. The parameters used to describe TATM in this section are shown in Table I.

Table I: List of TATM parameters and their definitions

Symbol	Definition
IPC_i	Instructions per cycle of i^{th} core
T_i	Current temperature of i^{th} core
TN_i	Average temperature of immediate neighboring cores of i^{th} core; if this core is on chip periphery and missing neighbors, then we consider virtual neighbor cores at ambient temperature in lieu of the missing cores
PT_i	Predicted temperature of ith core
T_{tj}	Thermal threshold of T_{ISj}-island
T_{lj}	Thermal limit of T_{ISj}-island
$IEIC_{tj}$	Inter-island cores for T_{ISj}-island whose island MRRs design temperature is greater than T_{ISj}
$IEIC_{lj}$	Inter-island cores for T_{ISj}-island whose island MRRs design temperature is smaller than T_{ISj}
$IAIC_j$	Intra-island cores for T_{ISj}-island
C	Regularization parameter
W	Weight vector for regression
x_i *and* y_i	Input and outputs in training and test data
ξ_i	Slack variables
ε	Error function
b	Bias for cost function

2) *Temperature Prediction Model:* We designed a support vector regression (SVR) based temperature predictor that accepts input parameters reflecting the workload for a core *i*, in terms of instructions per cycle (IPC_i), temperature (T_i), and surrounding core temperatures (T_{Ni}), and predicts the future temperature for core *i*.

Architecture: A typical SVR [21] relies on defining a prediction model that ignores errors that are situated within the ε range of the true value. This type of a prediction model is called an ε-insensitive prediction model. The variables (ξ and ε) measure the cost of the errors on the training points. These are zero for all points that are inside the ε-insensitive band. SVR is primarily designed to perform linear regression using a cost function (CF) as depicted in equation (1).

$$CF = min\ \frac{1}{2}\ W^T.W + C\ \sum_{i=1}^{n}(\xi_i + \xi_i^*) \quad (1)$$

Subject to:

$$y_i - W^T\Phi(x_i) - b \le \varepsilon + \xi_i\ (\xi_i \ge 0, i = 1, 2, \dots, n) \quad (2)$$

$$W^T\Phi(x_i) + b - y_i \le \varepsilon + \xi_i^*\ (\xi_i^* \ge 0, i = 1, 2, \dots, n) \quad (3)$$

$$\kappa(x_i, x_j) = \Phi(x_i)^T\Phi(x_j) \quad (4)$$

SVR performs linear regression in this high-dimension space using ε-insensitive loss and, at the same time, tries to reduce model complexity by minimizing $W^T.W$. This can be described by introducing (non-negative) slack variables ξ_i and ξ_i^* (i = 1 to n), to measure the deviation of training samples outside the ε-insensitive band. Thus SVR is formulated as minimization of the cost function (CF) in equation (1) with constraints shown in equations (2) and (3).

To handle non-linearity in data, SVR first maps the input x_i onto an m-dimensional space using some fixed (non-linear) mapping denoted as Φ, and then a linear model is constructed in this high-dimensional space as shown in equations (2) and (3). This allows it to overcome drawbacks of linear and logistic regression towards handling non-linearity in data. This class of SVRs is called kernel based SVRs which use kernel κ as shown in equation (4) for implicit mapping of non-linear training data into a higher dimensional space.

As on-chip temperature variation data is non-linear in the original space, our SVR model employs a kernel based regression which uses a Radial Basis Function (RBF) [14] as shown in equation (5):

$$\kappa(x_i, x_j) = exp\left(-\gamma \left| x_i - x_j \right|^2\right) \quad (5)$$

The RBF kernel improves the accuracy of SVR when data has non-linearity in the original space. We performed a sensitivity analysis (SA) to determine regularization parameter (C) and gamma (γ) values of the kernel based SVR (see Section III.A for chosen values). This SA overcomes the possibility of over fitting of training data and improves accuracy further. The definition of each of the variables used in equations (1) to (5) are mentioned in Table I.

Training and Accuracy: We trained our SVR model using a set of multi-threaded applications from the PARSEC [6] and SPLASH-2 [7] benchmark suites, specifically: blackscholes (BS), bodytrack (BT), vips (VI), facesim (FS), fluidanimate (FA), swaptions (SW), barnes (BA), fft (FFT), radix (RX), radiosity (RD), and raytrace (RT) with different thread counts: 2, 4 and 8. We considered different combinations of thread mappings on a 9-core (3×3) floorplan, to train our predictor to determine the temperature of the center (target) core. The threads mapped to a 9-core floorplan represents a generic mapping and can be applied to 64-core, 128-core, and 256-core floorplans.

As the future temperature of a target core is dependent on the average temperature of its immediate neighboring cores, we trained our SVR model with temperature inputs from the target core running a single thread, as well as its surrounding cores running a variable number of threads. Simulations with various mappings of these threads allowed us to obtain data to train our SVR model. This data included temperature for the target core and its neighboring core temperatures, as well as instructions per cycle (*IPC*) for the target core. IPC is very useful to determine if there is a phase change in an application and plays a crucial role in maintaining future temperature prediction accuracy especially when temperatures of a target core and its neighbors are similar at a given time. Our training algorithm involved an iterative process that adjusts the weights and bias values in the SVR (equations (1)-(3)) to fit the training set.

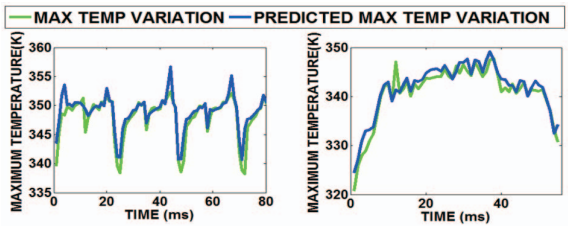


Fig. 5: Actual and predicted maximum temperature variation with execution time for (a) fluidanimate (FA) and (b) radiosity (RD) benchmarks run on a 64-core platform executing 32-threads

We verified the accuracy of our SVR model for multi-threaded benchmark workloads (we considered 6000 floorplans, with 70% of input data for training and 30% for testing) and found that it has an accuracy of over 95%. Fig. 5(a) and (b) show actual and predicted on-chip temperature variations for a 64-core platform executing 32 threads of the FA and RD benchmarks. From these figures it can be seen that our temperature predictor tracks temperature quite accurately. When predicted temperature is beneath thermal limit or exceeds the thermal threshold our thread migration mechanism (which is discussed next) migrates threads between cores to reduce

tuning/trimming power and keep overall maximum temperature below the threshold.

3) *Thermal Management Algorithm:* Fig. 6 illustrates the entire TATM technique. For each core, we periodically monitor the IPC value from performance counters and temperature from thermal sensors. If a thermal emergency is predicted for a core by the SVR predictor, then TATM initiates a thread migration procedure, otherwise no action is taken. In this work we have considered the thermal threshold of an island to be equal to maximum allowable temperature in that island i.e. $T_{tj} = T_{ISj} + 10K$ to avoid instances of irrevocable drift in MRs and thermal limit of an island is minimum allowable temperature in that island i.e. $T_{lj} = T_{ISj} - 10K$ to reduce tuning power.

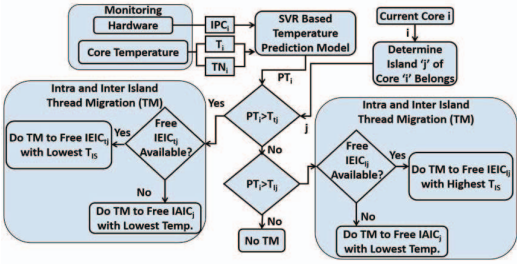


Fig. 6: Overview of TATM technique with support vector regression (SVR) based temperature prediction model.

Algorithm 2: TATM thread migration algorithm

Inputs: Current core temperature (T_i), average neighboring core temperature (TN_i), current core IPC (IPC_i)

1: **for** each core i **do** *// Loop that predicts future temperature*
2: PT_i = SVR_predict_future_temperature (T_i, TN_i, IPC_i)
3: **for** each core i **do** *// Loop that checks for free IAICs*
4: j = Find island of core (i)
5: **if** $IPC_i == 0$ **then** List_IAIC$_j$ = Push i *//add core to* IAIC$_j$ *list*
6: **for** each island j **do** // *Loop that create IEIC list for* T_{ISj}*-island*
7: **for** all islands m **do**
8: **if** $T_{Ism} > T_{Isj}$ **then** IEIC$_{tj}$ = push IAIC$_m$
9: **else if** $T_{Ism} < T_{Isj}$ **then** IEIC$_{lj}$ = push IAIC$_m$
10: **for** each core i **do** *// Loop that performs thread migration (TM)*
11: j = Find island of core (i)
12: **if** $PT_i > T_{tj}$ **then** *// Check predicted temp exceed thermal threshold*
13: **if** List_IEIC$_{tj} \neq \{\}$ *// Do inter-island TM*
14: Migrated_core = Find_lowest_T_{IS}_core(List_ IEIC$_{tj}$)
15: Thread_migration(core_i → Migrated_core)
16: n = island of Migrated_core
17: List_ IAIC$_n$ and List_IEIC$_{tj}$ = Pop Migrated_core
18: **else if** List_ IAIC$_j \neq \{\}$ **then** *// Do intra-island TM*
19: Migrated_core = Find_min_temp_core(List_ IAIC$_j$)
20: Thread_migration(core_i → Migrated_core)
21: List_ IAIC$_j$ and List IEIC = Pop Migrated_core
22: **else if** $PT_i < T_{lj}$ **then** *// if predicted temp is below thermal limit*
23: **if** List_IEIC$_{lj} \neq \{\}$ *// Do inter-island TM*
24: Migrated_core = Find_highest_T_{IS}_core(List_ IEIC$_{lj}$)
25: Thread_migration(core_i → Migrated_core)
26: n = island of Migrated_core
27: List_ IAIC$_n$ and List_IEIC$_{lj}$ = Pop Migrated_core
28: **else if** List_ IAIC$_j \neq \{\}$ **then** *// Do intra-island TM*
29: Migrated_core = Find_min_temp_core(List_ IAIC$_j$)
30: Thread_migration(core_i → Migrated_core)
31: List_ IAIC$_j$ and List IEIC = Pop Migrated_core

Output: Thread migration to IAIC or IEIC cores

Algorithm 2 shows the pseudo-code for the TATM thread migration procedure. Firstly, future temperature (PT_i) of the i^{th} core is predicted using the SVR based predictor with inputs: core temperature (T_i), core IPC (IPC_i), and temperature of neighboring cores (TN_i) in steps 1-2. The list of available free cores ($IAIC_j$) in T_{ISj}*-island* (i.e., those that are not currently executing any thread) is obtained in steps 3-5. In steps 6-9, a loop iterates over islands to generate a list of free cores $IEIC_{tj}$ and $IEIC_{lj}$ in other islands whose T_{IS} is higher and lower than current island respectively. In step 10, a loop iterates over all cores to perform thread migration. Step 12 and 22 checks for possible thread migration conditions (i.e., thermal emergency cases where current core predicted temperature (PT_i) in T_{ISj}*-island* is greater than thermal threshold (T_{tj}) or smaller than thermal limit (T_{lj})). If a thread migration is required as $PT_i > T_{tj}$, then in steps 13-21, we check for free $IEIC_{tj}$, and if they are available then we migrate the thread from the current core to the $IEIC_{tj}$ core with the lowest T_{IS} (inter-island migration), else we migrate the thread to a free $IAIC_j$ with the lowest temperature (intra-island migration). On the other hand, if a thread migration is required as $PT_i < T_{lj}$, then in steps 23-31, we check for free $IEIC_{lj}$, and if they are available then we migrate the thread from the current core to the $IEIC_{lj}$ core with the highest T_{IS} (inter-island migration), else we migrate the thread to a free $IAIC_j$ with the lowest temperature (intra-island migration). This TATM thread migration technique is invoked at every 1ms (epoch) and the sample frequency of SVR is considered as 0.1 ms (10 times lower compared to the epoch for thread migration). This sampling frequency is sufficient to monitor on-chip temperature variations [20].

III. Experiments, Results, And Analysis

A. Experimental Setup

The IPKISS [16] tool was used for the design and simulation of heaters, MRRs, and other silicon photonic components. This tool allows photonic component layout design, virtual fabrication of components in different technologies, physical simulation of components, and optical circuit design and simulation. The circuit-level results obtained from IPKISS were used for system-level simulation.

We target a 64-core CMP system for evaluation of our IHDTM framework. Each core has a Nehalem x86 [17] microarchitecture with 32 KB L1 instruction and data caches and a 256 KB L2 cache, at 32nm and running at 5GHz. We evaluate our framework on two well-known PNoC architectures: Corona [3] and Flexishare [22]. Corona uses a 64×64 multiple write single read (MWSR) crossbar with token slot arbitration. Flexishare uses 32 multiple write multiple read (MWMR) waveguide groups with a 2-pass token stream arbitration. Each MWSR waveguide in Corona and each MWMR waveguide in Flexishare is capable of transferring 512 bits of data from a source node to a destination node.

Table II: Properties of materials used by 3D-ICE tool [5]

Material	Thermal Conductivity	Volumetric Heat Capacity
Silicon	1.30e-4 W/µm K	1.628e-12 J/µm^3 K
Silicon dioxide	1.46e-6 W/µm K	1.628e-12 J/µm^3 K
BEOL	2.25e-6 W/µm K	2.175e-12 J/µm^3 K
Copper	5.85e-4 W/µm K	3.45e-12 J/µm^3 K

BEOL: Back end of line fabrication material

We modeled and simulated these architectures with the IHDTM framework for multi-threaded applications from the PARSEC [6] and SPLASH-2 [7] benchmark suites (Section II.B). Simulations were performed with an execution period of one billion cycles. Power and instruction traces for the benchmark applications were generated using the Sniper 6.0 [17] simulator and McPAT [18]. We used the 3D-ICE tool [5] for thermal analysis. We considered a three layered 3D-stacked CMP system as advocated in existing PNoC architectures [3], [22] with a planar die area footprint of 400mm^2, where the top layer is the core-cache layer, the middle layer is the analog electronic layer [3] which contains control circuits for modulator and photodetector and also the trans-impedance amplifiers of detectors, and the bottom layer is the photonic layer with MRRs, waveguides, ring heaters, and ring trimmers for carrier injection. Some of the key materials used in the construction of the 3D-stack in the 3D-ICE tool and their properties are shown in Table II. We used a heat sink adjacent to the core-cache layer for heat dissipation to the ambient environment.

The MRR thermal sensitivity was assumed to be 0.11nm/K [9]. For PNoCs, we considered 64 dense-wavelength-division-multiplexing (DWDM) waveguides sharing the working band 1530-1625 nm. The MRR trimming power is set to 130µW/nm [8] for current injection (blue shift) and tuning power is set to 240µW/nm [9] for heating (red shift). To compute laser power, we considered detector responsivity as 0.8 A/W [2], MRR through loss as 0.02 dB, waveguide propagation loss as 1 dB/cm, waveguide bending loss as 0.005 dB/90^0, and waveguide coupler/splitter loss as 0.5 dB [2]. We

calculated photonic loss in components using these values, which sets the photonic laser power budget and correspondingly the electrical laser power. For energy consumption of photonic devices, we adapt parameters from [19], with 0.42pJ/bit for every modulation and detection event, and 0.18pJ/bit for modulator/detector driver circuits.

The ambient temperature was set to 303K for our analysis and the for T_{IS1}-*island*, T_{IS2}-*island,* and T_{IS3}-*island* thermal thresholds were set to 373K, 353K, and 333K respectively and the thermal limits were set to 353K, 333K, and 313K respectively. Based on our sensitivity analysis we get the best accuracy for our SVR-based temperature predictor when parameters C and γ are set to 1000 and 0.1 respectively. We also considered thread migration overhead in our simulations that ranged from 500-1000 cycles to account for startup latency (extra cache misses, branch miss predictions) in the migrated core. Further, in the simulation we considered a 250-500 cycles overhead towards migration of threads for writing dirty cache lines from the write back caches, flushing the pipeline in the source core, and also PNoC latency to transfer data from architectural registers from the source core to the migrated core.

B. *Experimental Results*

We compared the performance of our IHDTM framework with two prior works on multicore thermal management: a ring aware policy (RATM) [12] and a predictive dynamic thermal management (PDTM) framework [20]. To compare these frameworks, we consider Corona and Flexishare PNoC architectures. RATM distributes threads uniformly across cores that are closer to PNoC nodes first and then distributes the remaining threads in a regular pattern from outer cores to inner cores. PDTM uses a recursive least square based temperature predictor to determine if the predicted temperature of a core exceeds a thermal threshold, and if so then thread migration is performed from that core to the coolest free core.

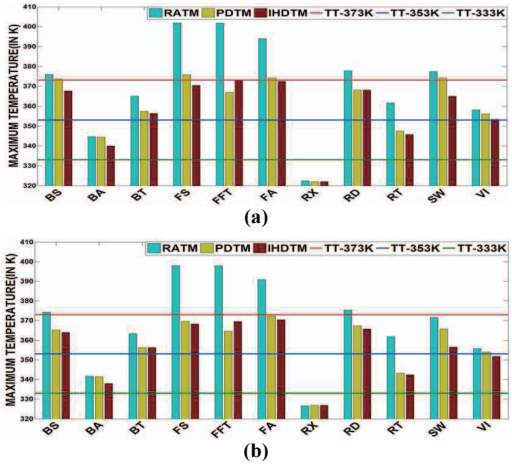


Fig. 7: Maximum temperature comparison of IHDTM with RATM and PDTM for (a) 48 and (b) 32 threaded PARSEC and SPLASH-2 benchmarks executed on 64-core CMP with Corona PNoC.

Fig. 7 shows the maximum temperature obtained with the three frameworks across eleven applications from the PARSEC and SPLASH-2 benchmarks suites with 48 and 32 thread counts executed on a 64-core system with the Corona PNoC architecture. From Fig. 7(a) it can be observed that for the IHDTM framework the FFT application with 48 threads exceeds the threshold (363K) by 0.4K as there are insufficient number of free cores in the 363K-island on the chip whose temperature is below the thermal threshold to migrate threads. However, in Fig. 7(b) our IHDTM framework avoids violating thermal thresholds for all the benchmark applications with 32 threads. On average, IHDTM has 13.27K and 13.72K lower maximum temperature compared to the RATM policy for 48 and 32 threads, respectively. Along with local thermal stabilization by PID controlled heaters, IHDTM migrates threads from hotter cores to cooler cores to control maximum temperature, whereas RATM does a simple thread allocation that is unable to appropriately control maximum temperature. For most of the cases, maximum temperatures with PDTM and IHDTM are below the thermal threshold. On average, IHDTM has 2.37K and 1.56K lower maximum temperature compared to the PDTM policy for 48 and 32 threads, respectively. IHDTM prefers to migrate threads within islands (inter-island) of cores based on the power consumption of running thread, which facilitates reduction in its peak temperature compared to PDTM.

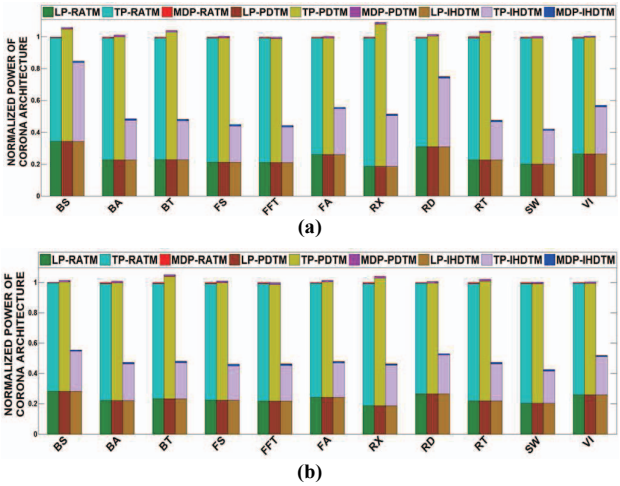


Fig. 8: Normalized power (Laser Power (LP), Trimming and tuning power (TP) and modulating and detecting Power (MDP)) comparison of IHDTM with RATM and PDTM for (a) 48 and (b) 32 threaded applications of PARSEC and SPLASH-2 suites executed on Corona PNoC architectures for a 64-core multicore system. Results shown are normalized w.r.t RATM.

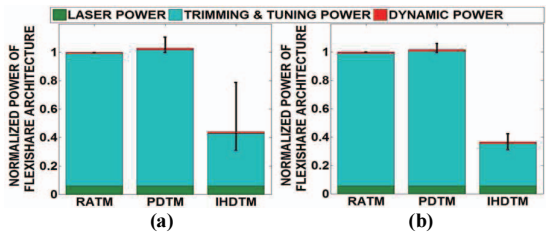


Fig. 9: Normalized average power (laser power (LP), trimming and tuning power (TP) and modulating and detecting power (MDP)) comparison of IHDTM with RATM and PDTM for (a) 48 and (b) 32 threaded applications of PARSEC and SPLASH-2 suites executed on Flexishare PNoC for a 64-core system. Power results are normalized wrt RATM results. Bars represent mean values of power dissipation; confidence intervals show variation in power across PARSEC and SPLASH-2 benchmarks.

Fig. 8 and Fig. 9 show the power consumption comparison for the three thermal-management techniques across multiple 48-threaded and 32-threaded applications for the Corona and Flexishare PNoC architectures, respectively. One of the main reasons why IHDTM has lower power consumption than RATM and PDTM is that it more aggressively reduces thermal tuning and trimming power in both Corona and Flexishare PNoCs. It is evident from Fig. 8(a)-(b) that IHDTM running 48 threads has 61.6% and 62.5%; and IHDTM running 32 threads has 67.3% and 68.5% lower thermal tuning and trimming power on average compared to RATM and PDTM for Corona PNoC architecture respectively. Similarly, from Fig. 9(a) and (b), it can be seen that IHDTM running 48 threads has 62.8% and 63.9%; and IHDTM running 32 threads has 68.5% and 70% lower tuning and trimming power on average compared to RATM and PDTM for Flexishare PNoC respectively. The IHDTM framework intelligently conserves tuning/trimming power compared to RATM and PDTM by performing intelligent intra-island and inter-island thread migration.

IHDTM saves considerable thermal tuning and trimming power to ultimately reduce total power. From the power analysis in Fig. 8 and

Fig. 9, it can be observed that IHDTM with Corona running 48 threads has 45.5% and 46.8%; and IHDTM with Corona running 32 threads has 51.6% and 52.3% lower total power consumption compared to Corona with RATM and PDTM respectively. Further, Flexishare with IHDTM running 48 threads has 55.9% and 57.2%; and 32 threads has 63.5% and 64.1% lower power consumption compared to Flexishare with RATM and PDTM respectively.

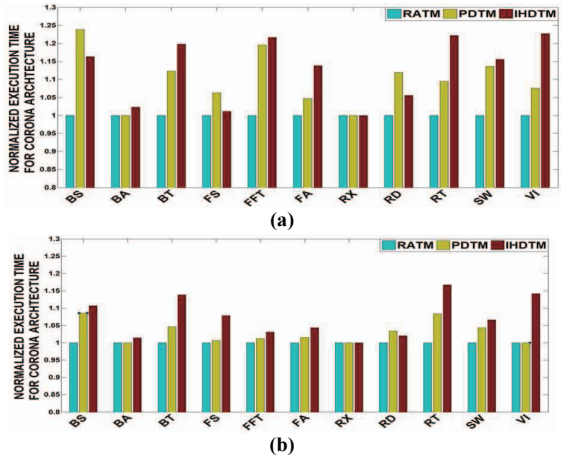


Fig. 10: Normalized execution time comparison of IHDTM with RATM and PDTM for (a) 48 and (b) 32 threaded applications of PARSEC and SPLASH-2 suites executed on Corona PNoC for a 64-core system. Results shown are normalized w.r.t RATM.

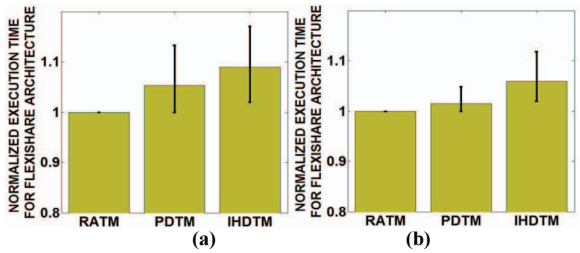


Fig. 11: Normalized average execution time comparison of IHDTM with RATM and PDTM for Flexishare PNoC running (a) 48; and (b) 32 threaded applications from PARSEC and SPLASH-2 suites executed on 64-core system. Results are normalized wrt RATM results. Bars represent mean values of execution time; confidence intervals show variation in execution time across PARSEC and SPLASH-2 benchmarks.

Fig. 10 shows the average execution time comparison between the three frameworks across the 11 48-threaded and 32-threaded applications from the PARSEC and SPLASH-2 suites, for the Corona PNoC architectures respectively. From Fig. 10(a) and (b) it can be seen that Corona with IHDTM running 48 and 32 threads has 12.8% and 7.4% higher execution time respectively compared to Corona with RATM. Corona with IHDTM needs extra execution time to migrate threads between cores whereas the RATM policy simply schedules threads without any migration, and thus does not possess such overheads. The execution time overhead of Corona with IHDTM running 32 threads is lower compared to 48-threaded version, as it lowers traffic congestion in the Corona PNoC which in turn reduces overall latency. Further, Corona with IHDTM running 48 and 32 threads has 2.6% and 4.3% higher execution time respectively compared to PDTM. IHDTM has more number of thread migrations compared to the number of thread migrations in PDTM, as IHDTM performs intra-island and inter-island thread migrations when the thermal emergencies are predicted by the SVR predictor. Similarly, from Fig. 11(a) and (b), the Flexishare with IHDTM running 48 and 32 threads has 9% and 5.9% higher execution time compared to RATM and 3.4% and 4.4% higher execution time compared to Flexishare with PDTM. From the execution time results it can be seen that Flexishare has lower execution time overhead compared to Corona as it uses a faster MWMR crossbar instead of slower MWSR crossbar in Corona.

Lastly, from the power consumption and execution time results, we can obtain energy consumption results for the three frameworks. On an average, for Corona, energy consumption of IHDTM running 48 threads is 38.5% and 45.4% lower compared to RATM and PDTM, respectively. Further energy consumption of Corona with IHDTM running 32 threads is 48.1% and 50.3% lower compared to RATM and PDTM, respectively. On the Flexishare architecture, IHDTM running 48 threads has 52.2% and 56% lower energy consumption compared to RATM and PDTM respectively; and IHDTM running 32 threads has 61.4% and 62.6% lower energy consumption compared to RATM and PDTM, respectively. From the energy consumption results IHDTM has better energy savings for the optimized Flexishare compared to the Corona.

IV. Conclusions

We have presented the IHDTM framework that exploits device-level on-chip thermal islands and system-level dynamic thread migration scheme TATM for the reduction of maximum on-chip temperature and also conserves trimming and tuning power of MRRs in DWDM-based PNoC architectures. The proactive thermal management scheme used in IHDTM results in interesting trade-offs between performance and power/energy across two different state-of-the-art crossbar-based PNoC architectures. Our experimental analysis on the well-known Corona and Flexishare PNoC architectures has shown that IHDTM can notably conserve total power by up to 64.1% and thermal tuning power by up to 70%.

References

[1] The international technology roadmap for semiconductors (ITRS), http://www.itrs.net/, Interconnections, 2011.

[2] S. V. R. Chittamuru et al., A reconfigurable silicon-photonic network with improved channel sharing for multicore architectures, in ACM GLSVLSI, May 2015.

[3] D. Vantrease et al., Light speed arbitration and flow control for nanophotonic interconnects, in IEEE/ACM MICRO, Dec. 2009.

[4] D. Dang et al., Mode-division-Multiplexed Photonic Router for High Performance NoC, in IEEE VLSID, Jan. 2015.

[5] A. Sridhar et al., 3D-ICE: Fast compact transient thermal modeling for 3d ICs with inter-tier liquid cooling, in Proc. ICCAD, Nov. 2010.

[6] C. Bienia et al., The PARSEC benchmark suite: Characterization and Architectural Implications, in Proc. PACT, 2008, pp. 72-81.

[7] S. C. Woo et al., The SPLASH-2 programs: characterization and methodological considerations, in ISCA, 1995, pp. 24-36.

[8] J. Ahn et al., Devices and architectures for photonic chip-scale integration, in Applied Physics A: MSP, 95:989-997, June 2009.

[9] C. Nitta, M. Farrens, and V. Akella, Addressing system-level trimming issues in on-chip nanophotonic networks, in Proc. HPCA, 2011.

[10] S. S. Djordjevic et al., CMOS-compatible, athermal silicon ring modulators clad with titanium dioxide, in Optics Express, 21(12), 2013.

[11] S. V. R. Chittamuru, et al., SPECTRA: A framework for thermal reliability management in silicon-photonic Networks-on-Chip, in IEEE VLSID, Jan 2016.

[12] T. Zhang et al., Thermal management of manycore systems with silicon photonic networks, in Proc. DATE, 2014.

[13] D. Dang et al., PID-controlled Heater-based Thermal Management in Photonic Network-on-Chip, in IEEE ICCD 2015.

[14] B. E. Boser et al., A training algorithm for optimal margin classifiers, in 5th Annu. Workshop Comput. Learning Theory, 1992, pp. 144-152.

[15] http://www.pidlab.com/en/

[16] IPKISS - a generic and modular software framework for parametric design http://www.ipkiss.org/

[17] T. E. Carlson et al., Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulations, in Proc. SC, 2011.

[18] S. Li et al., McPAT: An integrated power, area, and timing modeling framework for manycore architectures, in MICRO, 2009.

[19] S. V. R. Chittamuru, et al., PICO: Mitigating Heterodyne Crosstalk Due to Process Variations and Intermodulation Effects in Photonic NoCs, in IEEE/ACM DAC, Jun. 2016.

[20] I. Yeo, C. Liu, and E. Kim, Predictive dynamic thermal management for multicore systems, in Proc. DAC, 2008, pp. 734-739.

[21] D. Harris et al., Support Vector Regression Machines, in Advances in Neural Information Processing Systems (NIPS), 1996, MIT Press.

[22] Y. Pan, J. Kim, and G. Memik, Flexishare: Channel sharing for an energy efficient nanophotonic crossbar, in Proc. HPCA, 2010.