Thermal Management for S-NUCA Many-Cores via Synchronous Thread Rotations

Yixian Shen, Sobhan Niknam, Anuj Pathania, and Andy D. Pimentel
University of Amsterdam, Amsterdam, The Netherlands
y.shen@uva.nl, s.niknam@uva.nl, a.pathania@uva.nl, a.d.pimentel@uva.nl

***Abstract*—On-chip thermal management is quintessential to a thermally safe operation of a many-core processor. The presence of a physically distributed logically shared Last-Level Cache (LLC) significantly reduces the performance penalty of migrating threads within the cores of an S-NUCA many-core. This cost reduction allows novel thermal management of these many-cores via synchronous thread migration. Synchronous thread migration provides a viable alternative to Dynamic Voltage and Frequency Scaling (DVFS) and asynchronous thread migration used traditionally to manage thermals of S-NUCA many-cores.**

We present a theoretical method to compute the peak temperature in many-cores with synchronous thread migrations. We use the method to create a thermal management heuristic called *HotPotato* that maximizes the performance of S-NUCA many-cores under a peak temperature constraint. We implement *HotPotato* within the state-of-the-art *HotSniper* simulator. Detailed interval thermal simulations with *HotSniper* show an average 10.72% improvement in response time of S-NUCA many-cores when scheduling with *HotPotato* compared to a state-of-the-art thermal-aware S-NUCA scheduler.

I. Introduction

Many-core processors house tens of cores on a single die and excel in executing multi-threaded applications with significant inter-thread communication [1]. In shared memory many-cores, the cores (overlying threads) communicate indirectly using a logically shared memory address space. The cores can always communicate with each other using the off-chip main memory. However, to minimize the communication cost (latency), many-cores come with a low-latency on-chip Last-Level Cache (LLC). LLC itself is often physically distributed between the cores as cache banks to distribute the on-chip cache coherency traffic. The banks are connected using a Network-on-Chip (NoC) to allow for a bottleneck-free flow of traffic [2].

Static Non-Uniform Cache Architecture (S-NUCA) is a memory architecture that statically maps the LLC to the main memory [3], [4]. S-NUCA allows for a quick search for a cache line (page), given the static mapping with minimal cache coherence. During a thread migration on an S-NUCA many-core, only the cache lines stored in private caches on the core from where the thread migrates need flush to the LLC. The shared LLC then refills the cache lines on another core where the migrated thread restarts its execution. Consequently, the physically distributed logically shared LLC and an NoC significantly reduces the cost of thread migrations on many-cores with S-NUCA caches.

S-NUCA many-cores, similar to other many-cores, suffer from thermal issues [5]. Thermal-aware schedulers primarily depend upon Dynamic Voltage and Frequency Scaling (DVFS) technology to manage the thermals for S-NUCA many-cores [6]. DVFS reduces a core's frequency (and voltage) to reduce its power consumption and lower its temperature [7], [8]. However, DVFS also results in a significant drop in the performance of the overlying applications [9]. Thermal-aware schedulers also incorporate thread migrations as an additional knob to prevent hotspots from forming on-chip on S-NUCA many-cores [10]. However, schedulers perform these migrations asynchronously on-need basis and often as a measure of last resort. These on-demand asynchronous thread migrations combined with DVFS represent the state-of-the-art strategy for thermal-aware schedulers for S-NUCA many-cores.

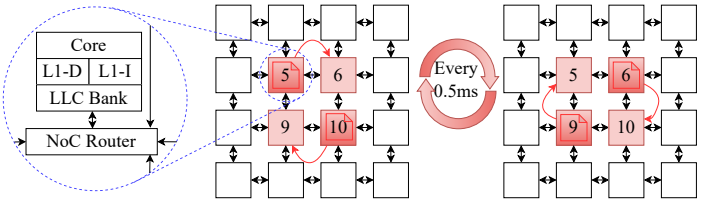
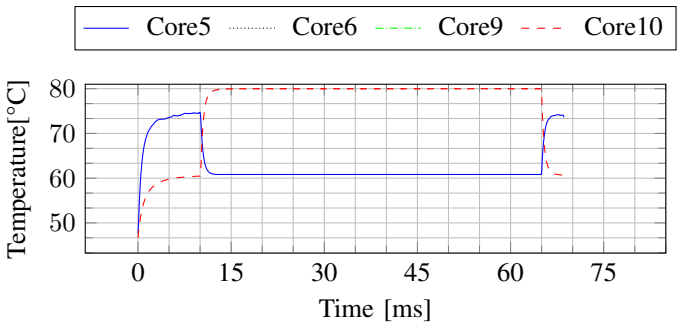


Fig. 1: Synchronous thread rotation on a S-NUCA many-core.

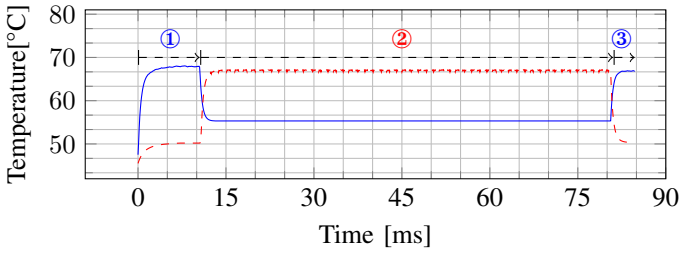
The task migration penalty for S-NUCA is not particularly severe because the LLC cache in S-NUCA is logically shared and physically distributed. Therefore, only the private L1 and L2 caches require a refresh on migrations. *In this work, we make an observation that the average performance penalty from thread migrations is significantly lower than DVFS on S-NUCA many-cores.* Based on this observation, we propose a novel method for thermal management for S-NUCA many-cores that involves synchronously rotating (migrating) threads on S-NUCA many-cores such that no core has a chance to heat up beyond the given thermal threshold. Synchronous task rotations average the temperatures between hot and cold cores, allowing us to investigate the periodic solution for thermal management of S-NUCA many-cores.

Motivational Example. Figure 1 shows an abstraction of a 16-core S-NUCA many-cores. In this example, we primarily focus on the center-most cores of the many-core, namely Cores 5, 6, 9, and 10. We simulate the many-core using detailed interval thermal simulations using the *HotSniper* [12] toolchain. We set the thermal threshold at 70 °C.

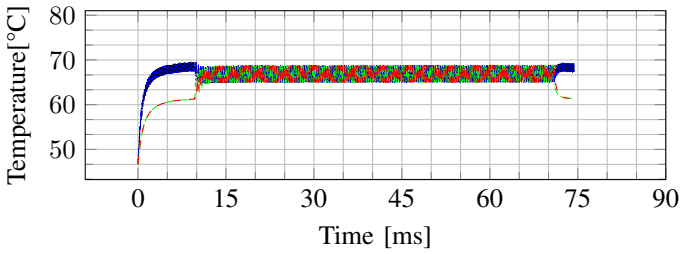
Figure 2(a) shows the thermal trace when a two-threaded *blackscholes* benchmark from *PARSEC* [13] benchmark suite executes on Cores 5 and 10 of the many-core running at their peak frequency of 4 GHz. This execution results in a



(a) Fixed Peak Frequency



(b) Thermally Safe Frequency provided by *TSP* [11]



(c) Synchronous Thread Rotation at Peak Frequency

Fig. 2: Thermal trace with different thermal management techniques on the central cores of a 16-core many-core running a two-threaded instance of *blackscholes*.

fast response time of 68 ms. However, the execution is not thermally sustainable as the temperature during the execution at 80 °C goes significantly beyond the thermal threshold.

Figure 2(b) shows the thermal trace of the above execution under a state-of-the-art power budgeting algorithm called *TSP* [14]. *TSP* limits (budgets) the power consumption of the cores of the many-cores using DVFS such that the execution temperature does not cross the thermal threshold. However, this comes at a slower response time of 84 ms.

The *blackscholes* benchmark has a master-slave structure wherein primary execution constantly switches between the master and slave thread. In Figure 2(b), only the master thread works for data preparation in the initial Phase ① on Core 5, and the slave thread is idle on Core 10. Subsequently, slave threads start to work in Phase ②, and the master thread becomes idle. Finally, in Phase ③, the slave thread becomes idle again, and the master thread wraps up the execution. Figure 2(c) shows the thermal trace when two threads (master and slave) of *blackscholes* are synchronously rotated between the four centers cores of the many-core at a rotation interval of 0.5 ms in every phase, as abstracted in Figure 1. In such an execution, the heat from the execution of the master and slave threads averages out. The temperature of any core does not exceed the threshold, and the response time stands at 74 ms. The response time with synchronous thread rotation incurs a performance penalty of 8.1% due to overheads originating from task migration, but still is 11.9% faster than DVFS-based power budgeting. Therefore, within the thermal threshold, the performance of task rotation synchronously outperforms the DVFS-based TSP approach.

Our Novel Contributions. Based on the above discussion, we make the following novel contributions via this work.

- We are the first to propose thermal management for S-NUCA many-cores using synchronous thread migrations.
- We propose (based on the underlying many-core RC thermal model [15]) an analytical method for calculating the peak temperature of a synchronously rotating sequence of threads on a set of cores at a certain rotation frequency.
- We integrate the method into a run-time scheduler called *HotPotato* that selects a performance-maximizing thermally-safe rotation on S-NUCA many-cores.
- We implement *HotPotato* in the state-of-the-art *HotSniper* simulator. We show the superiority of *HotPotato* over a state-of-the-art thermal-aware scheduler for S-NUCA many-core using detailed interval thermal simulations.

Open-Source Contributions. The source code for the *HotPotato* scheduler as a *HotSniper* plugin is available for download at https://github.com/yixianuva/hotpotato.

II. Related Work

The authors of [16] were the first to propose thread (task) rotation to mitigate the peak temperature of single-core processors. They heuristically calculated the sequence of single-threaded kernels that run to completion sequentially such that the peak temperature of a single-core processor minimizes. The heuristic relies on chaining hot and cold threads in the execution sequence to minimize peak temperature. However, the complexity of the involved optimization problem increases significantly from a single-core to many-core, executing multiple multi-threaded tasks in parallel.

Authors of [3] were the ones to introduce S-NUCA memory architecture. Subsequently, several S-NUCA many-cores made it to the market [17]. However, like other many-cores, S-NUCA many-cores also suffer thermal issues [5], [18]. Therefore, their thermal management remains an active research subject. Authors of the [19] were the first to characterize the performance heterogeneity in cores of an S-NUCA many-core due to the presence of physically distributed LLC and NoC. Authors of [6] were the first to combine the performance heterogeneity and thermal heterogeneity in cores of S-NUCA many-cores for their thermal management.

In [6], [20], the authors present a DVFS-based thermal-aware scheduler called *PCGov* that uses TSP-based [14] power-budgeting for mapping tasks on S-NUCA many-cores. Authors of [10], [21] present a thermal-aware scheduler called *PCMig* that extends *PCGov* with neural network-driven asynchronous

on-demand thread migrations. *PCMig*, to the best of our knowledge, remains the state-of-the-art thermal-aware scheduler for S-NUCA many-cores. In this work, on S-NUCA many-cores, we show *HotPotato*, with its synchronous thread migration (without DVFS), is superior in performance to *PCMig* with its DVFS and asynchronous on-demand thread migrations.

III. System Models

A. Architecture Model

Figure 1 contains the abstraction of the architectural model used in this work. The target architecture is an S-NUCA many-core with n micro-architecturally homogeneous cores. A grid-based NoC employing XY-routing connects the cores. Each core holds a bank of the physically distributed logically shared L2 LLC. Each core also has a private L1-Instruction and Data cache and an NoC-router. The performance (or thermals) of cores is positively (or negatively correlated) to their Average Manhattan Distance (AMD) from other cores [19]. The topography of the many-core dictates that the AMD of the cores increase as we traverse away from the many-core's center.

B. Thermal Model

We employ a well-known RC thermal model based on the duality between thermal behavior and electrical circuits [15]. RC thermal model has N thermal nodes wherein the first n nodes represent the n cores of the many-core, and the remaining $N-n$ nodes correspond to the cooling system. As per the model, we can compute the temperature of each thermal node (a function of its power consumption, the temperature of neighboring thermal nodes, and the ambient temperature) by a set of N first-order differential equations.

$$\mathbf{A}\mathbf{T}' + \mathbf{B}\mathbf{T} = \mathbf{P} + T_{amb}\mathbf{G} \tag{1}$$

where $\mathbf{A} = [a_{i,j}]_{N\times N}$ contains the thermal capacitance values of each thermal node, $\mathbf{B} = [b_{i,j}]_{N\times N}$ represents the thermal conductivity values between neighboring nodes, $\mathbf{T} = [T_i(t)]_{N\times 1}$ denotes the temperature on every node at time instant t, $\mathbf{T}' = [T_i'(t)]_{N\times 1}$ accounts for the first order derivative of the temperature on each node concerning time, $\mathbf{P} = [p_i]_{N\times 1}$ contains the power consumption on each node, and $\mathbf{G} = [g_i]_{N\times 1}$ contains the thermal conductivity between each node and ambient temperature. By defining matrix $\mathbf{C} = -\mathbf{A}^{-1}\mathbf{B}$, we can rewrite Equation (1) as following.

$$\mathbf{T}' = \mathbf{C}\mathbf{T} + \mathbf{A}^{-1}\mathbf{P} + T_{amb}\mathbf{A}^{-1}\mathbf{G} \tag{2}$$

As the temperature of cores approaches the steady state, we can rewrite Equation (1) as follows.

$$\mathbf{T_{steady}} = \mathbf{B}^{-1}\mathbf{P} + \mathbf{B}^{-1}T_{amb}\mathbf{G} \tag{3}$$

where $\mathbf{T_{steady}} = [T_{steady_i}]_{N\times 1}$ contains the steady-state temperature of per node and $\mathbf{B}^{-1}$ is the inverse of the matrix $\mathbf{B}$.

IV. Peak temperature calculation

Thread rotation involves executing threads rotating periodically on thermally-coupled cores of a many-core. We present a computationally-efficient analytical solution to calculate the peak temperature for a given thread rotation on a set of cores. Let $\mathbf{P}$ be the power consumption vector of the rotating threads. Due to performance heterogeneity, the same thread can have different power consumption on different cores, and our proof accounts for these differences.

Let the threads execute for a fixed epoch τ on each core during the rotation. Let $\mathbf{T_{init}} = [T_{init_i}]_{N\times 1}$ be the matrix storing the initial temperature of nodes at time $t = 0$. Initial conditions are mandatory for solving the involved differential equations. We use *MatEx* [22] to solve for the transient temperature via the matrix exponential method. We can obtain the temperature at time τ as a function of $\mathbf{T_{init}}$ using the following equation.

$$\mathbf{T}_\tau = \mathbf{T_{steady}} + e^{\mathbf{C}\tau}(\mathbf{T_{init}} - \mathbf{T_{steady}}) \tag{4}$$

Let δ be the periodicity of the thread rotation. By design, a thread will migrate back to its original starting core after time $\delta\tau$. Let T_{amb} be the ambient temperature. We assume $\mathbf{T_{init}} = [T_{amb}]_{N\times 1}$ to simplify the proof (without affecting the outcome) by shifting the origin to the ambient temperature. Subsequently, $\mathbf{T_{init}}$ and T_{amb} remove themselves from further calculations. Furthermore, by substituting $\mathbf{T_{steady}}$ from Equation (3) into Equation (4), we obtain the temperature after the first rotation epoch.

$$\mathbf{T}_\tau = (\mathbf{I} - e^{\mathbf{C}\tau})\mathbf{B}^{-1}\mathbf{P}_\tau \tag{5}$$

where $\mathbf{P}_\tau$ denotes the average power consumption of the rotating threads over the epoch τ, and $\mathbf{I}$ is an identity matrix of size N. Let $\mathbf{T}_{\delta\tau}$ be the temperature at the end of δ epochs. The temperature $\mathbf{T}_{\delta\tau}$ is the initial temperature for epoch $(\delta+1)\tau$. We define $\mathbf{w} = (\mathbf{I} - e^{\mathbf{C}\tau})\mathbf{B}^{-1}$ as the rotational factor. The subsequent temperature traces evolves based on Equation (4). Therefore, the temperature $\mathbf{T}_{\delta\tau}$ after the first rotation period δ (or after $\delta\tau$ epochs) is as follows.

$$\mathbf{T}_{\delta\tau} = \mathbf{w}\mathbf{P}_{\delta\tau} + e^{\mathbf{C}\tau}\mathbf{w}\mathbf{P}_{(\delta-1)\tau} + \cdots + e^{(\delta-1)\mathbf{C}\tau}\mathbf{w}\mathbf{P}_\tau \tag{6}$$

The temperature $\mathbf{T}_{\delta\tau}$ is a combination of power history over the rotation period δ and rotation factor $\mathbf{w}$. Let d be rotation periods where after the transient temperature pattern in a rotation approaches a steady state and then repeats itself. The first temperature component $\mathbf{T}_{(d\delta+1)\tau}$ after $d\delta$ durations is as follows.

$$\mathbf{T}_{(d\delta+1)\tau} = (\mathbf{I} + \sum_{i=1}^{d} e^{i\delta\mathbf{C}\tau})\mathbf{w}\mathbf{P}_\tau + \sum_{i=1}^{d} e^{(i\delta-1)\mathbf{C}\tau}\mathbf{w}\mathbf{P}_{2\tau} + \cdots + \sum_{i=1}^{d} e^{(i-1+\delta)\mathbf{C}\tau}\mathbf{w}\mathbf{P}_{\delta\tau} \tag{7}$$

$\mathbf{T}_{(d\delta+1)\tau}$, therefore, is a combination of power history and the accumulated rotation component. As per the thermodynamic Equation (4), matrix $\mathbf{A}$ is an invertible matrix and $\mathbf{B}$ is symmetrical. Therefore, we can factorize matrix $\mathbf{C}$. Consequently, we can analytically solve $e^{\mathbf{C}\tau} =$

$[e^{\mathbf{C}\tau}{}_{i,j}]_{N\times N}$ as a matrix exponential using $e^{\mathbf{C}\tau} = \mathbf{VDV}^{-1}$, where $\mathbf{V} = [v_{i,j}]_{N\times N}$ are the eigenvectors of matrix $\mathbf{C}$. Let $\mathbf{D} = \mathbf{diag}(e^{\lambda_1\tau}, e^{\lambda_2\tau}, \cdots, e^{\lambda_N\tau})$ be the diagonal matrix where $\lambda_1, \lambda_2, \cdots, \lambda_N$ represent the eigenvalues of matrix $\mathbf{C}$. The following equation gives the matrix addition after the matrix exponential decomposition for rotational components.

$$\sum_{i=1}^{d} e^{i\mathbf{C}\tau} = \mathbf{V}\cdot\mathbf{diag}(\sum_{i=1}^{d} e^{\lambda_1 i\tau}, \sum_{i=1}^{d} e^{\lambda_2 i\tau}, \cdots, \sum_{i=1}^{d} e^{\lambda_N i\tau})\cdot\mathbf{V}^{-1} \quad (8)$$

Since $\mathbf{A}^{-1}\ \mathbf{B}$ is congruent to identity matrix $\mathbf{I}$ [22], then it is also a positive definite matrix, so $\mathbf{C}= -\mathbf{A}^{-1}\ \mathbf{B}$ is a negative definite matrix. As a result, eigenvalues are all negative. Therefore, as $d \to +\infty$ in the steady state, the sum of each element in eigenvalues of the diagonal matrix in Equation (8) is given by the following equation.

$$\lim_{d\to+\infty} e^{\lambda_1\tau} + e^{\lambda_1 2\tau} + \cdots + e^{\lambda_1 d\delta\tau} = \frac{e^{\lambda_1\tau}}{1-e^{\lambda_1\tau}} \quad (9)$$

Therefore, each element in the diagonal element is upper bounded by a fixed value in the steady state and is independent of d. Due to the linear nature of the matrix transformation, we get a safe upper bound of the peak temperature. We can use Equations (8) and (9) to rewrite Equation (7) as the following.

$$\begin{aligned}\mathbf{T}_{(d\delta+1)\tau} &= \mathbf{V}\cdot\mathbf{diag}(\frac{1}{1-e^{\delta\lambda_1\tau}}, \frac{1}{1-e^{\delta\lambda_2\tau}}, \cdots, \frac{1}{1-e^{\delta\lambda_N\tau}})\cdot\mathbf{V}^{-1}\mathbf{wP}_\tau \\ &+ \mathbf{V}\cdot\mathbf{diag}(\frac{e^{(\delta-1)\lambda_1\tau}}{1-e^{\delta\lambda_1\tau}}, \frac{e^{(\delta-1)\lambda_2\tau}}{1-e^{\delta\lambda_2\tau}}, \cdots, \frac{e^{(\delta-1)\lambda_N\tau}}{1-e^{\delta\lambda_N\tau}})\cdot\mathbf{V}^{-1}\mathbf{wP}_{2\tau} + \cdots \\ &+ \mathbf{V}\cdot\mathbf{diag}(\frac{e^{\lambda_1\tau}}{1-e^{\delta\lambda_1\tau}}, \frac{e^{\lambda_2\tau}}{1-e^{\delta\lambda_2\tau}}, \cdots, \frac{e^{\lambda_N\tau}}{1-e^{\delta\lambda_N\tau}})\cdot\mathbf{V}^{-1}\mathbf{wP}_{\delta\tau}\end{aligned} \quad (10)$$

The subsequent temperature components after d periods $\{\mathbf{T}_{(d\delta+2)\tau}, \mathbf{T}_{(d\delta+3)\tau}, \cdots, \mathbf{T}_{(d\delta+\delta)\tau}\}$ consist of a similar format but with different linear combinations of the rotation components. We can derive the peak temperature of a rotation by traversing the temperature components $\mathbf{T}_{(d\delta+i)\tau}$ and maxing them. We also readjust the origin by factoring in T_{amb}.

$$\begin{aligned}T_{peak} = max\{max\{\mathbf{B}^{-1}T_{amb}\mathbf{G} + \mathbf{T}_{(d\delta+1)\tau}\} + \\ max\{\mathbf{B}^{-1}T_{amb}\mathbf{G} + \mathbf{T}_{(d\delta+2)\tau}\} + \cdots + max\{\mathbf{B}^{-1}T_{amb}\mathbf{G} + \mathbf{T}_{(d\delta+\delta)\tau}\}\}\end{aligned} \quad (11)$$

V. Thread Rotation Scheduling

We present a heuristic that provides a thread rotation schedule for S-NUCA many-cores. We call our scheduler *HotPotato*, and the scheduling it performs *HotPotato* scheduling. The name takes inspiration from analogous *HotPotato* routing [23] in computer networks. An S-NUCA many-core comprises topological rings of AMDs, as shown in Figure 3. The cores within the same ring are performance- and thermal-wise homogeneous. The rings become performance-wise constrained and thermal-wise unconstrained as the AMD value increases. *HotPotato* exploits these concentric rings of AMDs to develop a schedule of synchronously rotating threads. Threads assigned to one ring always rotate within that ring at a given rotational interval. The goal is to develop a schedule that keeps the many-core's peak temperature T_{peak} lower than the thermal threshold T_{DTM} while maximizing its performance. T_{DTM} is the temperature

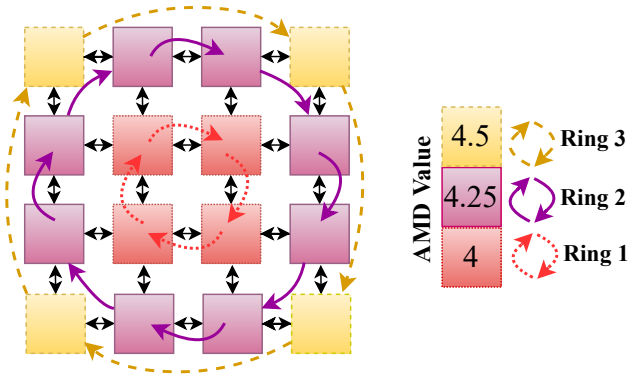


Fig. 3: Abstraction for concentric AMD-based rotation rings.

Algorithm 1 Efficient Peak Temperature Calculation

Input: Floorplan, $\mathbf{P_{history}}$,T_{DTM},τ,N,δ,T_{amb}
Output: T_{peak}

/* One-Time Design-time phase */
$T_{peak} = 0, \mathbf{T_{init}} = [\mathbf{B}^{-1}T_{amb}\mathbf{G}]_{N\times 1}$ ▷ Initialize T_{peak} and $\mathbf{T_{init}}$
$\boldsymbol{\alpha} = [\alpha_{i,j} = 0]_{N\times N}, \boldsymbol{\beta} = [\beta_{i,j} = 0]_{N\times N}$
for each $i = 1, 2, \cdots, N$ **do**
  for each $j = 1, 2, \cdots, N$ **do**
    $\alpha_{i,j} = \sum_{k=1}^{N} v_{i,k} \times \frac{1}{(1-e^{n\lambda_k\tau})}$
    $\beta_{i,j} = \sum_{k=1}^{N}(1-e^{\lambda_k\tau}) \times \tilde{v}_{k,j} \times \tilde{B}_{k,j}$
/* Run-time phase */
for each $e = 1, 2, \cdots, \delta$ **do** ▷ Calculate temperature component
  $\boldsymbol{\Theta} = [\Theta_{i,j} = 0]_{N\times N}, \boldsymbol{\Phi} = [\Phi_i = 0]_{N\times 1}$
  for each $f = 1, 2, \cdots, \delta$ **do** ▷ Based on Equation (10)
    for each $i = 1, 2, \cdots, N$ **do**
      for each $j = 1, 2, \cdots, N$ **do**
        $\Theta_{i,j} = \sum_{k=1}^{N} \alpha_{i,j} \times e^{[(\delta-f+1)\%\delta]\lambda_i\tau} \times \beta_{k,j}$
        $\Phi_j = \Phi_j + \Theta_{i,j} \times P_{[(f+e-1)\%\delta]j}$
  $\mathbf{T}_{d\delta+e} = \boldsymbol{\Phi}$
  $T_{peak} = max\{T_{peak}, max\{\mathbf{T_{init}} + \mathbf{T}_{d\delta+e}\}\}$ ▷ Equation (11)
return T_{peak}

at which many-core triggers the hardware-controlled Dynamic Thermal Management (DTM) that crashes the many-core's operating frequency to save it from damage.

Let there be R AMD-based rings on an n-core S-NUCA many-core. Let n_{active} be the number of cores required by the n_{active} threads executing with a one-thread-per-core model. The total design space for assigning n_{active} threads to R AMD rings is $\frac{n!}{(n-n_{active})!R!}$. We can use Algorithm 1 based on Equation (11) to efficiently determine the peak temperature (thermal safety) of any given schedule (design point) in the design space. Algorithm 1 consists of a design-time phase that pre-calculates the floorplan-based constants (auxiliary matrices $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$) for instant use at run-time. The algorithm uses the power history of a thread from the last 10 ms in its calculation. However, determining a thermally-safe schedule that also maximizes many-core performance is an NP-hard problem. Therefore, we propose a heuristic for *HotPotato* to find a near-optimal solution to the problem.

Algorithm 2 describes the greedy heuristic that describes *HotPotato* decisions when a new thread enters the many-core or when an old thread leaves the many-core. When a new thread enters the many-core, *HotPotato* tries to assign the thread to the lowest AMD ring for the best performance. Since there is only a

Algorithm 2 *HotPotato* Scheduling

Input: Floorplan, $P_{history}$,T_{DTM},AMD,CPI,Threads Γ, τ,Δ,N
Output: Response time

1: /*New threads coming */
2: **for each** $AMD_i = AMD_1, AMD_2, \cdots, AMD_l$ **do**
3: **if** $T_{peak} + \Delta < T_{DTM}$ **then**
4: $coreLoc \leftarrow selectBestCandiate()$ ▷ T_{peak} by Algorithm 1
5: $T_{peak} \leftarrow updatePeakTemperature()$ ▷ Update T_{peak}
6: **Break**
7: **else**
8: **while** $T_{peak} > T_{DTM}$ **do**
9: *Sort* the threads Γ based on CPI in *decreasing* order
10: *Migrate* the thread with lowest CPI to the higher AMD ring
11: $T_{peak} \leftarrow updatePeakTemperature()$ ▷ Update T_{peak}
12: **while** $T_{peak} > T_{DTM}$ **do**
13: $\tau \leftarrow updateRotationSpeed()$ ▷ Update τ
14: $T_{peak} \leftarrow updatePeakTemperature()$ ▷ Update T_{peak}
15: /*Threads finished or thermal headroom $> \Delta$ */
16: **while** $T_{DTM} - T_{peak} > \Delta$ **do**
17: **for each** $AMD_i = AMD_l, AMD_{l-1}, \cdots, 1$ **do**
18: *Sort* the threads Γ based on CPI in *decreasing* order
19: *Migrate* the threads with the highest CPI to the lower AMD ring
20: $T_{peak} \leftarrow updatePeakTemperature()$ ▷ Update T_{peak}
21: **if** $T_{DTM} - T_{peak} \leq \Delta$ **then**
22: **Break**
23: **while** $T_{DTM} - T_{peak} > \Delta$ **do**
24: $\tau \leftarrow updateRotationSpeed()$ ▷ Update τ
25: $T_{peak} \leftarrow updatePeakTemperature()$ ▷ Update T_{peak}
26: **if** $T_{DTM} - T_{peak} \leq \Delta$ **then**
27: **Break**

limited number of empty slots in a ring, *HotPotato* evaluates all possible empty slots in parallel for thermal sustainability, and chooses the one with the lowest peak temperature using Algorithm1 in Lines 3-6. If it is thermally unsustainable, it assigns it to the next higher AMD ring that has lower performance but better thermals. The process continues till it recursively reaches the ring with the highest AMD shown in Lines 7-11. If assigning the thread even to the highest AMD ring is thermally unsustainable, then *HotPotato* reduces the rotation interval τ till enough headroom generates to accommodate the new thread shown in Lines 12-14. *HotPotato* does not move the existing threads from their rings when placing a new thread to avoid cascading peak temperature calculations that are unsustainable.

When an existing thread leaves the system, new thermal headroom manifests. *HotPotato* sorts the thread as per their Cycle per Instruction (CPI). It then tries to migrate the thread with the highest CPI (the most memory-bound thread) to the lowest AMD ring as long as the migration is thermally sustainable, as shown in Lines 16-22. The highest CPI thread is the thread that is most likely to benefit from the improved memory performance of a lower AMD ring. If the highest CPI thread is already in the lowest thermally sustainable AMD ring, it similarly tries to migrate the thread with the next higher CPI. If all the threads are in their most thermally sustainable lowest AMD ring with still thermal headroom, then *HotPotato* reduces the rotation interval τ to the highest thermally sustainable value. If $\tau \rightarrow 0$, then the workload is thermally sustainable without rotation, and therefore rotations stop to maximize the performance, as shown in Lines 23-27.

There is a possibility of a sudden increase or decrease in thermal headroom (given by the user-defined parameter Δ) with a drastic change in power consumption of existing threads on many-core. In such cases, *HotPotato* adjusts the rotation interval τ to deal with the new circumstances.

Complexity Analysis: We discuss the complexity of the design-time phase and run-time phase. In the design-time phase, we calculate the auxiliary matrices α and β. The complexity is $\mathcal{O}(N^2)$. Peak temperature calculation in Algorithm 1 requires iteratively computing the rotation components that take $\mathcal{O}(2\delta^2 N^2)$. In Algorithm 2, in the run-time phase, we assume that the varying rotation speed range is η. In the worst case, it traverses R AMD rings with the complexity $\mathcal{O}(2\eta ln(\eta) R\delta^3 N^2)$.

TABLE I: Core parameters for simulated S-NUCA processor.

Core Parameter	
Number of Cores	64
Core Model	x86, 4.0 GHz, 14 nm, out-of-order
L1 I/D cache	16/16 KB, 8/8-way,64B-block
LLC	128 KB per core, 16-way, 64B-block
NoC Latency	1.5ns per hop
Noc link width	256 Bit
The area of core	0.81 mm^2

VI. Evaluation

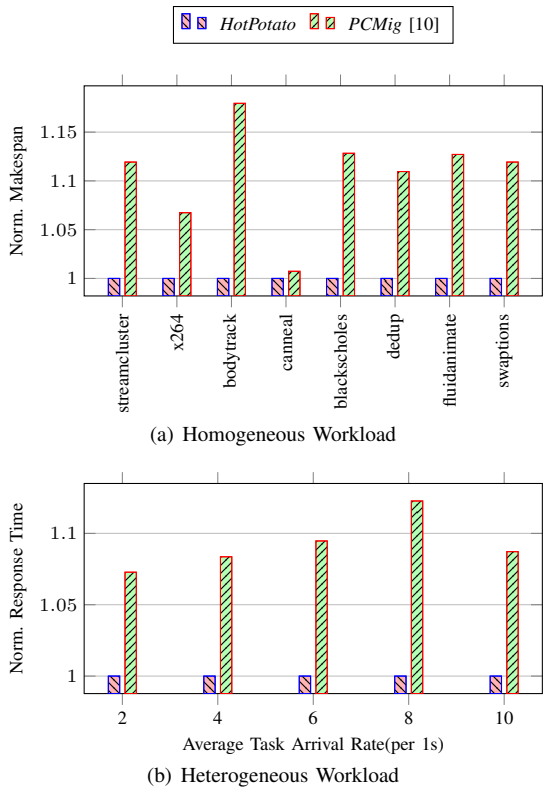
Experimental Setup. We use the interval thermal simulation toolchain *HotSniper* [12] for simulating an S-NUCA many-core. Table I lists the simulated core and network parameters. We specify The thermal headroom Δ at 1 °C. We set the idle core power and initial rotation speed at 0.3 W and 0.5 ms, respectively. The ambient and threshold temperatures are set at 45 °C and 70 °C, respectively.

We use *PARSEC* [13] benchmark suite to simulate the workload. In particular, we use the *streamcluster*, *x264*, *bodytrack*, *canneal*, *blackscholes*, *dedup*, *fluidanimate*, and *swaptions* benchmarks with *sim-small* input. We do not use *facesim* and *raytrace* benchmarks due to the lack of small-size inputs. We also do not use *ferret*, *freqmine*, and *vips* benchmarks due to unresolved simulation errors in *HotSniper*.

Baseline. We compare the *HotPotato* scheduler with the *PCMig* scheduler [6]. *PCMig* is the state-of-the-art scheduler for the thermal management of S-NUCA many-cores. It uses DVFS and asynchronous thread migrations as knobs. While *HotPotato* does not use DVFS, we allow *PCMig* to perform fine-grained DVFS at a step size of 100 MHz.

Comparative Evaluation with Homogeneous Workload. We fully load the 64-core S-NUCA many-core with varisized multi-threaded instances of the same benchmark. We then simulate a fixed system wherein all instances start execution together. Fig 4(a) reports the normalized makespan of the execution with *HotPotato* and *PCMig* schedulers. Results show *HotPotato*, on average, provides a 10.72% speedup for different benchmarks. *Canneal* being a memory-intensive benchmark, produces very little heat. Consequently, we observe the lowest speedup gains (0.73%) with *Canneal*.

Comparative Evaluation with Heterogeneous Workload. We create a random 20-benchmark multi-program multi-threaded workload. We then simulate an open system wherein



(a) Homogeneous Workload

(b) Heterogeneous Workload

Fig. 4: Comparative evaluation results for *HotPotato* and *PCMig* [10] scheduler on a 64-core many-core.

tasks arrive at different arrival rates following a Poisson distribution to create a system under a varying load. *HotPotato* outperforms *PCMig* under all load scenarios. The relative speedup gains with *HotPotato* are minimal when the system is under-loaded or over-loaded, as there is a limited scope of thermal optimizations. In a medium-loaded system, *HotPotato* provides up to 12.27% improvement over *PCMig*.

Run-time Overhead. Across 10000 runs under full load, *HotPotato* takes 23.76 μs to calculate a synchronous thread rotation schedule for a 64-core many-core on one of the many-core's cores. Therefore, *HotPotato* projects an overhead of 4.75% for a thread rotation epoch of 0.5 ms.

VII. Conclusion & Future Work

In this work, we present a scheduler called *HotPotato* for the thermal management of S-NUCA many-cores. *HotPotato* builds upon the observation that the performance penalty of thread migration is lower than DVFS on S-NUCA many-cores. It, therefore, uses a heuristic based on synchronous thread migrations rather than commonly employed DVFS and asynchronous thread migrations for managing the thermals of S-NUCA many-cores. The heuristic uses a computationally-efficient method of our design to analytically calculate the peak temperature from thread rotations making *HotPotato* feasible for run-time use. Thermal interval simulations using the *HotSniper* toolchain show a thermally-sustainable 10.72% average increase in performance over the state-of-the-art.

Future Work: We plan to unify synchronous task rotation with DVFS for even more efficient thermal management of S-NUCA many-cores. Subsequently, we plan to explore the idea of synchronous task rotation with 3D S-NUCA many-cores [24] using the state-of-the-art *CoMeT* [25] interval thermal simulator designed for 3D-stacked processors.

References

[1] J. L. Manferdelli, N. K. Govindaraju, and C. Crall, "Challenges and opportunities in many-core computing," *Proceedings of the IEEE*, 2008.

[2] W. Choi, K. Duraisamy, R. G. Kim, and et al., "Hybrid network-on-chip architectures for accelerating deep learning kernels on heterogeneous manycore platforms," in *ESWEEK*, 2016.

[3] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger, and S. W. Keckler, "A nuca substrate for flexible cmp cache sharing," in *ICS*, 2005.

[4] J. Merino, V. Puente, P. Prieto, and J. Á. Gregorio, "Sp-nuca: a cost effective dynamic non-uniform cache architecture," *ACM SIGARCH*, 2008.

[5] W. Huang, M. R. Stant, K. Sankaranarayanan, and et al., "Many-core design from a thermal perspective," in *DAC*, 2008.

[6] M. Rapp, M. Sagi, A. Pathania, A. Herkersdorf, and J. Henkel, "Power- and cache-aware task mapping with dynamic power budgeting for many-cores," *TC*, 2019.

[7] S. Eyerman and L. Eeckhout, "Fine-grained dvfs using on-chip regulators," *TACO*, 2011.

[8] Q. Wang, X. Mei, H. Liu, Y.-W. Leung, Z. Li, and X. Chu, "Energy-aware non-preemptive task scheduling with deadline constraint in dvfs-enabled heterogeneous clusters," *TPDS*, 2022.

[9] Y. G. Kim, M. Kim, and et al., "An adaptive thermal management framework for heterogeneous multi-core processors," *TC*, 2020.

[10] M. Rapp, A. Pathania, and et al., "Neural network-based performance prediction for task migration on s-nuca many-cores," *TC*, 2020.

[11] S. Pagani, H. Khdr, J.-J. Chen, M. Shafique, M. Li, and J. Henkel, "Thermal safe power (tsp): Efficient power budgeting for heterogeneous manycore systems in dark silicon," *TC*, 2016.

[12] A. Pathania and J. Henkel, "Hotsniper: Sniper-based toolchain for many-core thermal simulations in open systems," *ESL*, 2018.

[13] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *PACT*, 2008.

[14] S. Pagani, H. Khdr, W. Munawar, J.-J. Chen, M. Shafique, M. Li, and J. Henkel, "Tsp: Thermal safe power: Efficient power budgeting for many-core systems in dark silicon," in *ESWEEK*, 2014.

[15] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. R. Stan, "Hotspot: A compact thermal modeling methodology for early-stage vlsi design," *VLSI*, 2006.

[16] R. Jayaseelan and T. Mitra, "Temperature aware task sequencing and voltage scaling," in *ICCAD*, 2008.

[17] S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, J. MacKay, M. Reif, L. Bao, J. Brown *et al.*, "Tile64-processor: A 64-core soc with mesh interconnect," in *ISSCC*, 2008.

[18] B. Wu, P. Dai, Y. Cheng, Y. Wang, J. Yang, Z. Wang, D. Liu, and W. Zhao, "A novel high performance and energy efficient nuca architecture for stt-mram llcs with thermal consideration," *TACD*, 2019.

[19] A. Pathania and J. Henkel, "Task scheduling for many-cores with s-nuca caches," in *DATE*, 2018.

[20] M. Rapp, A. Pathania, and J. Henkel, "Pareto-optimal power-and cache-aware task mapping for many-cores with distributed shared last-level cache," in *ISLPED*, 2018.

[21] M. Rapp, A. Pathania, T. Mitra, and J. Henkel, "Prediction-based task migration on s-nuca many-cores," in *DATE*, 2019.

[22] S. Pagani, J.-J. Chen, M. Shafique, and J. Henkel, "Matex: Efficient transient and peak temperature computation for compact thermal models," in *DATE*, 2015.

[23] R. Teixeira, A. Shaikh, T. Griffin, and J. Rexford, "Dynamics of hot-potato routing in ip networks," in *SIGMETRICS*, 2004.

[24] F. Li, C. Nicopoulos, T. Richardson, Y. Xie, V. Narayanan, and M. Kandemir, "Design and management of 3d chip multiprocessors using network-in-memory," in *ISCA*, 2006.

[25] L. Siddhu, R. Kedia, S. Pandey, M. Rapp, A. Pathania, J. Henkel, and P. R. Panda, "Comet: An integrated interval thermal simulation toolchain for 2d, 2.5 d, and 3d processor-memory systems," *TACO*, 2022.