Power- and Cache-Aware Task Mapping with Dynamic Power Budgeting for Many-Cores

Martin Rapp, Mark Sagi, Anuj Pathania, Andreas Herkersdorf, *Senior Member, IEEE*, and Jörg Henkel, *Fellow, IEEE*

Abstract—Two factors primarily affect the performance of multi-threaded tasks on many-core processors with logically-shared and physically-distributed Last-Level Cache (LLC): the LLC latencies of threads running on different cores and the per-core power budgets that aim to guarantee thermally safe operation. Two knobs affect these factors: First, the mapping of threads to cores affects both the LLC latencies and the power budgets. Second, dynamic power budgeting refines the power budgets during task execution. A mapping that spatially distributes threads across the many-core increases the power budgets, but unfortunately also increases the LLC latencies. Contrarily, mapping all threads near the center of the many-core minimizes the LLC latencies, but unfortunately also decreases the power budgets. Consequently, both metrics cannot be simultaneously optimal, which leads to a Pareto-optimization for task mapping that has formerly not been exploited. Dynamic power budgeting reallocates the power budgets according to the tasks' execution phases. This results in a dynamically changing non-uniform power budget, which further increases the performance. We are the first to present a run-time algorithm *PCGov* combining task-agnostic task mapping and task-aware dynamic power budgeting for many-cores with shared distributed LLC. *PCGov* yields up to 21 percent lower response time and 13 percent lower energy consumption compared to the state-of-the-art, with a low overhead of less than 0.5 percent.

Index Terms—Processor scheduling, power dissipation, thermal stability, dark silicon, cache memory, task mapping, low power design, TSP (Thermal Safe Power)

1 Introduction

Sophisticated task mapping and power budgeting policies are required to efficiently utilize the increasing number of cores in many-core processors [2]. The task mapping policy determines those cores where threads of a task are pinned while the power budgeting policy sets a limit to the power consumptions of the cores in order to prevent thermal violations. Both policies need to be matched to attain best performance and efficiency.

Due to increasing power densities in smaller technology nodes, active cores of a many-core are constrained by a power budget that aims to ensure thermally safe operation [3]. The most commonly used power budget is *Thermal Design Power* (TDP) [4]. TDP is a constant per-chip power budget determined at design-time and therefore it is unaware of the number and location of active cores (the mapping). Hence, it needs to be pessimistic to be able to prevent thermal violations. Authors of [5] present *Thermal Safe Power* (TSP) to overcome this limitation by considering the actual mapping. TSP provides a per-core power budget for a given mapping that prevents thermal violations in the steady-state. By considering the actual mapping, TSP can give a higher power budget than TDP while still being thermally safe. However, TSP is task-agnostic and hence results in a uniform power budget for all active cores. *A uniform power budget can change over time, e.g., if the mapping changes. Thereby, uniformity means that all threads receive the same power budget.*

1.1 Task Mapping

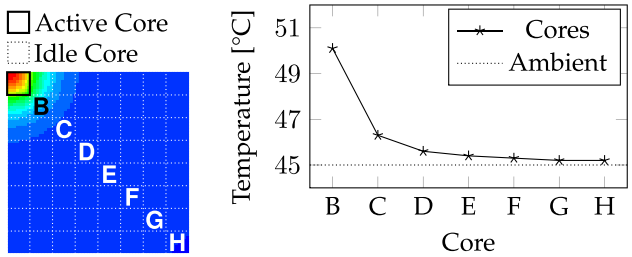
The problem of assigning multiple resources to multiple tasks is NP-complete in the general case [6]. Therefore, finding the optimal task mapping is difficult at run-time. Instead, heuristics that exploit domain-specific knowledge are needed [7]. This work focuses on open systems [8], where new tasks arrive in an unpredictable order. We employ the one-thread-per-core model well suited for many-cores [9]. We further assume rigid tasks whose threads cannot be migrated during their execution [8].

Pinning threads to spatially close cores leads to thermal hotspots, whereas spreading these threads far across the many-core leads to better heat distribution and thereby to a higher power budget. The increased power budget can potentially increase performance of a task. Fig. 1 shows that heat conductance on the many-core approximately falls off exponentially with the distance.[1] Cores B to H in Fig. 1 are all idle, but their temperatures differ notably. The farther a

- *M. Rapp and J. Henkel are with the Karlsruhe Institute of Technology, Karlsruhe 76131, Germany. E-mail: {martin.rapp, henkel}@kit.edu.*
- *M. Sagi and A. Herkersdorf are with the Technical University of Munich, Munich 80333, Germany. E-mail: {mark.sagi, herkersdorf}@tum.de.*
- *A. Pathania is with the National University of Singapore, Singapore 119077. E-mail: pathania@comp.nus.edu.sg.*

Manuscript received 11 July 2018; revised 1 Feb. 2019; accepted 21 July 2019. Date of publication 20 Aug. 2019; date of current version 19 Dec. 2019. (Corresponding author: Martin Rapp.)
Recommended for acceptance by N. Enright Jerger.
Digital Object Identifier no. 10.1109/TC.2019.2935446

1. Section 6 describes the experimental setup.



(a) Core Locations (b) Core Temperatures

Fig. 1. Heat conductance across the many-core falls off approximately exponentially. Cores D to H have similar temperatures even though their distances to the active core (that executes a thread) differ significantly.

core is from the active core, the lower is its temperature. However, once a sufficient distance is reached, changes in temperature are insignificant. The temperatures of cores D to H are nearly the same even though their distances to the active core differ significantly. *We observe that if active cores have sufficiently high distance to each other, further increasing it only slightly increases the power budget.*

Fig. 2 shows the architecture of a many-core with both logically-shared and physically-distributed LLC. Each core holds a bank of the LLC. All cores are connected by a Network-on-Chip (NoC). Static Non-Uniform Cache Access (S-NUCA) is a policy that allows threads on all cores to access the LLC in a scalable manner. This policy is employed in commercially available many-cores [10]. To maintain scalability, S-NUCA determines a static association of memory address to LLC bank at design-time [11]. This causes a thread to access the LLC banks on all cores equally likely irrespective of the core it is pinned. LLC latency is affected by the hop count between the thread's core and the LLC bank, which corresponds to the Manhattan Distance (MD). The MD between two points **a** and **b** is the sum of absolute differences of their Cartesian coordinates: $d(\mathbf{a}, \mathbf{b}) = \sum_i |a_i - b_i|$. Cores that have a lower Average Manhattan Distance (AMD) to all other cores (LLC banks) have a lower average LLC latency [12]. The closer a core is to the center of the many-core, the lower is the AMD. Fig. 3c shows the cycle stacks[13] of a slave thread of four-threaded *streamcluster* when it is mapped to cores in the center or the corners of the many-core, respectively. Both cases employ the same constant frequency during the execution of the tasks. The only difference is the mapping, which determines the AMD. Higher AMD significantly increases the LLC-related components of the cycle stack (mem-ifetch and mem-nucad), while the remaining components of the cycle stack are almost unchanged. Thereby, overall performance is

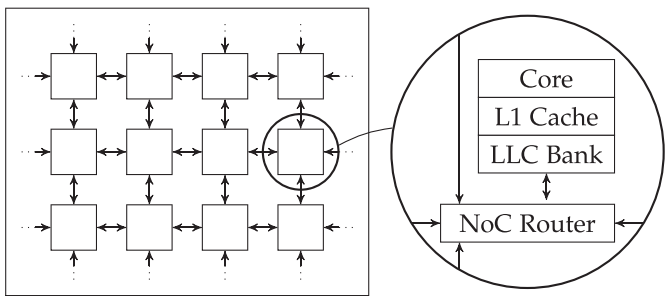
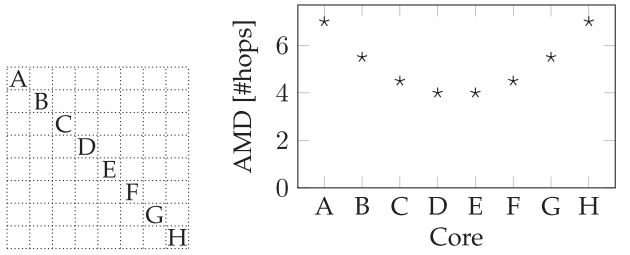
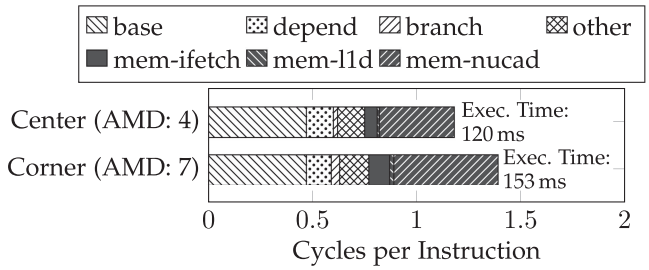


Fig. 2. Architecture of a many-core with logically-shared and physically-distributed LLC.



(a) Core Locations (b) LLC Access Characteristics



(c) Cycle Stacks of a *Streamcluster* Slave Thread

Fig. 3. (a) and (b): AMD of cores to the LLC banks is lowest for the cores closest to the center of the many-core. Cores C and F close to the center experience only a small increase in AMD. (c): The AMD significantly influences the amount of cycles spent waiting for the LLC (mem-ifetch and mem-nucad) and therefore affects the overall performance.

reduced. So, in S-NUCA many-cores, task performance can be increased by pinning threads to cores close to the center.

For a sufficiently high power budget, the performance of *PARSEC* [14] tasks is dominated by the core with the maximum AMD in this task's mapping since the thread pinned to this core forms a bottleneck [12]. Therefore, we use the maximum AMD of all cores of a task as a metric to optimize the task mapping. Figs. 3a and 3b show that the AMD to LLC banks increases steeply near the corner of the many-core (cores A and H), but changes only slightly near the center of the many-core (cores C and F). *Therefore, we make the observation that cores near the center of the many-core experience only a slight increase in LLC latency.* These observations are not restricted to S-NUCA many-cores. *Similar observations also hold true for directory-based many-cores where the cache directory is distributed along the cores [15].*

Maximum power budget and minimum LLC latency are obtained by contradictory mapping decisions. While the former tends to maximize the distances between threads, the latter tends to cluster threads near the center of the many-core. Since both metrics cannot be optimal simultaneously, there exist multiple Pareto-optimal mappings. *In a Pareto-optimal mapping, power budget cannot be improved without sacrificing LLC latency and vice versa.* Based on the insights from Figs. 1 and 3, it is very likely that there exist a lot of near-optimal mappings with respect to both optimization goals. *To maximize task performance, a trade-off between pinning threads as far as possible from each other in order to maximize the power budget and pinning threads as near as possible to the center in order to minimize the LLC latency is to be found.* We demonstrate this using the following motivational example.

Motivational Example. Fig. 4 shows the execution times of different four-threaded *PARSEC* tasks on a 64-core S-NUCA many-core under three different mappings. TSP [5]

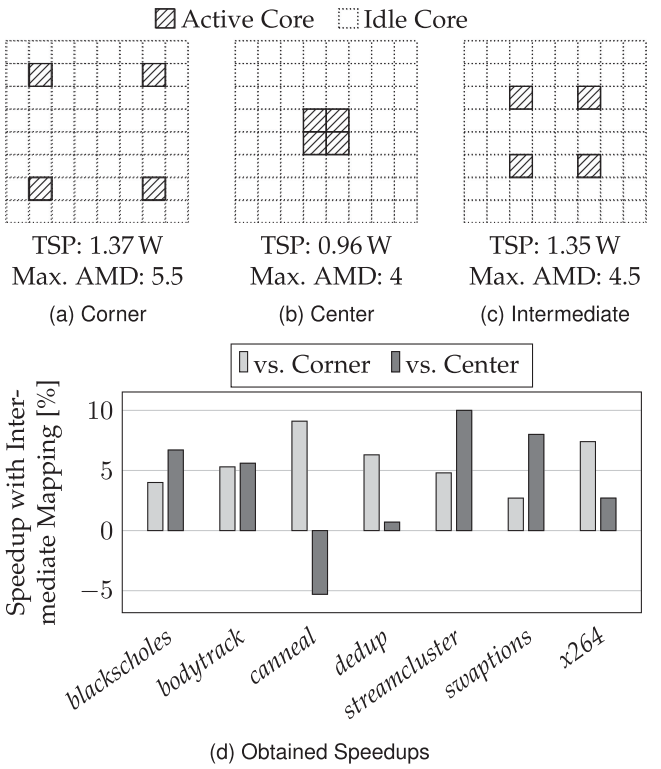


Fig. 4. Impact of mapping on *PARSEC* tasks performance. Neither mapping threads to the corner cores nor mapping them to the center cores results in the best performance. Instead, an intermediate mapping yields the highest performance for almost all *PARSEC* tasks.

gives a uniform per-core power budget for each mapping. Mapping Fig. 4a uses cores close to the corners to gain a high distance between active cores in order to maximize the power budget. Using the exact corner cores would result in a lower power budget since cores in the corners have fewer neighboring cores that can absorb heat. Mapping Fig. 4b uses cores as close as possible to the center of the many-core to minimize the LLC latency. Mapping 4c uses inner, but not the center cores to balance LLC latency and power budget. Dynamic Voltage and Frequency Scaling (DVFS) enables cores to run at different voltage and frequency levels allowing them to restrict the power consumption by sacrificing performance. The DVFS levels of the cores are adapted at run-time to not exceed the power budget.

Mapping Fig. 4c almost always provides the best task performance among the mappings shown in Fig. 4. Mapping 4a suffers from high LLC latency and Mapping 4b suffers from a low power budget while Mapping Fig. 4c has both a high power budget and a low LLC latency. The average speedup with Mapping Fig. 4c among all tasks is 5.7 percent (or 4.1 percent) compared to mapping to corners (or center).

Different *PARSEC* tasks experience different performance with different task mappings due to their different characteristics. Mapping Fig. 4c provides the best performance for all tasks except *canneal*, for which Mapping Fig. 4b is the best. *Canneal* is memory-intense and has the most LLC accesses per second among all *PARSEC* tasks. Therefore, the LLC latency has a much higher impact on performance than the power budget which eliminates all scope for any trade-off.

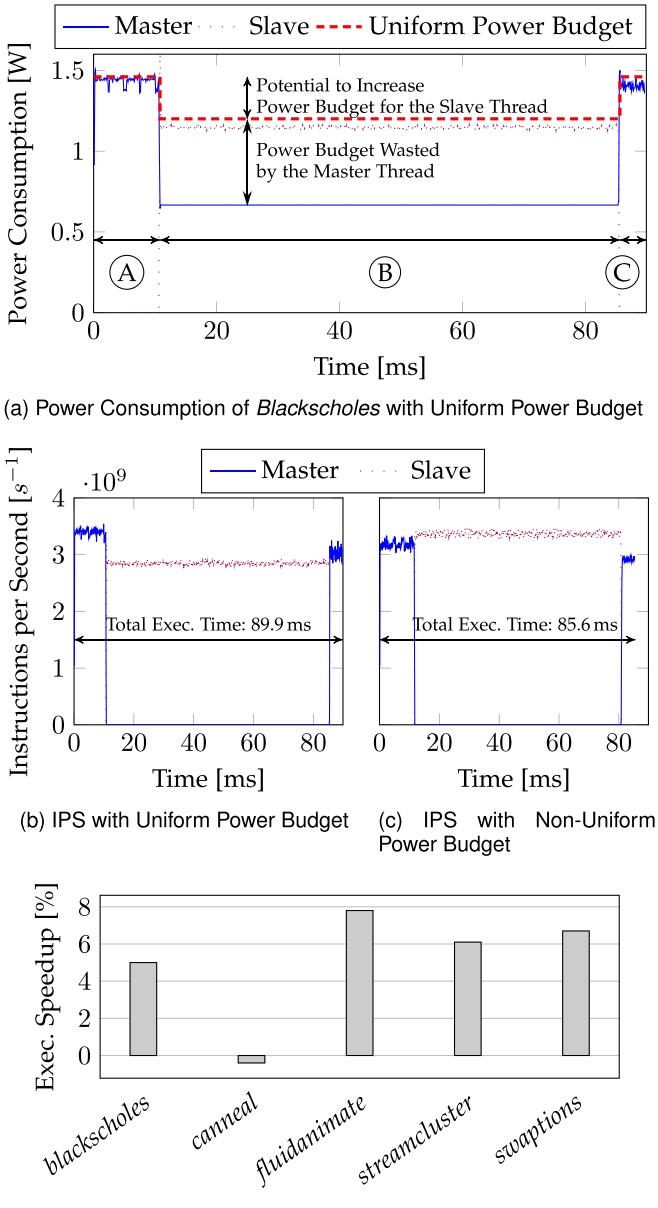


Fig. 5. Using a uniform power budget for all threads results in wasted power budget. Fig. 5a shows that the *blackscholes* master thread cannot use its power budget for most of the time. Redistributing this power budget to the slave thread results in higher performance (Figs. 5b and 5c). Fig. 5d shows the observed speedups with a non-uniform power budget over a uniform power budget for all *PARSEC* tasks that are capable to produce two threads.

1.2 Dynamic Power Budget Reallocation

The execution of tasks typically shows distinct phases [16]. Fig. 5a shows the execution of two-threaded *blackscholes*. The execution can be divided into three phases denoted by A, B, and C. In Phase A, the master thread prepares the work while the slave thread has not yet been spawned. Phase B starts when the slave thread spawns. The slave thread processes the prepared data while the master is idle. After the slave has finished execution, the master resumes and completes task execution in Phase C. Fig. 5a also shows the power budgets of the threads if the uniform power budget TSP is used. In Phase A, the slave thread is not yet spawned and consequently, the master thread receives a

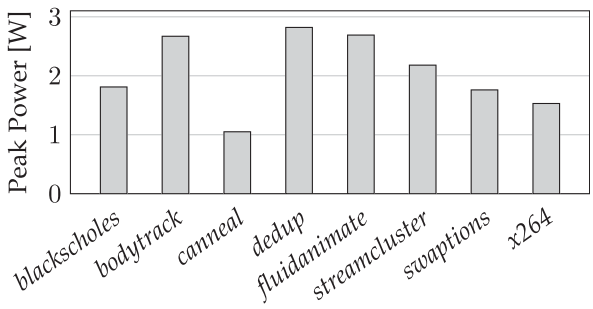


Fig. 6. Peak power consumptions of a single thread of different *PARSEC* tasks differ strongly.

high power budget. In Phase B, master and slave need to share the power budget, which causes it to drop to prevent thermal violations. In Phase C, the master's power budget returns to its original level. The same observations also hold true for higher numbers of slave threads. Here, two threads were chosen for the sake of simplifying illustrations.

With a uniform power budget, both master and slave threads are provided the same power budget in Phase B. But, the behaviors of these two threads differ strongly: the slave thread is compute-intense and would benefit from a higher power budget, while the master thread is idle and cannot make use of its assigned power budget. Therefore, a non-uniform power budget is better applicable. The power budget of the master thread can be reduced to the bare minimum, which allows the slave's power budget to be raised while still being thermally safe. Figs. 5b and 5c show the Instructions per Second (IPS) trace with the uniform and non-uniform power budgets, respectively. Redistributing the unused power budget from the master thread to the slave thread results in a higher IPS for the slave thread and ultimately in a 5.0 percent reduction in the response time of the task. Fig. 5d shows the observed speedups for all *PARSEC* tasks that are capable of producing two threads. Like observed with the mapping of threads to cores, different tasks respond differently to different policies. The highest speedup is observed with *fluidanimate* (7.8 percent), the lowest with *canneal*. The reason why *canneal* does not show any speedup is that canneal is memory-intense and cannot consume high power since the cores are stalled often while waiting for the memory. Therefore, the power budget does not restrict *canneal* at all and reallocation has no effect. The average speedup for these five tasks is 5.0 percent.

Not only the threads of a single task can behave heterogeneously, but also threads of different tasks may show heterogeneous characteristics. Fig. 6 shows the peak power consumption of a single thread for different *PARSEC* tasks. The peak power consumptions vary strongly. It is apparent that not all tasks can utilize high power budgets. Fig. 7 shows an example of how this observation can be exploited to boost the performance. *Canneal*, which is a memory-intense task, cannot make use of its assigned power budget. *Blackscholes*, on the other hand, is a compute-intense task and thereby could benefit from a higher power budget. To prevent the impact of inhomogeneous LLC latency, all threads are pinned to cores with the same AMD class. Reducing the power budget for the threads of *canneal* to a value close to its actual power consumption allows to increase the power budget for the threads of *blackscholes*. In

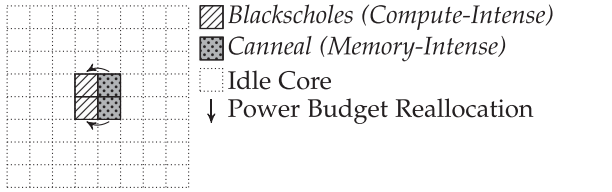


Fig. 7. Transfer of unused power budget between heterogeneous tasks increases the performance. The response time of *blackscholes* reduces from 97.8 ms to 95.4 ms while the response time of *canneal* does not change.

this example, threads of the same task still receive the same power budget. This allows us to separate the effects of power budget reallocation between different tasks and between threads of the same task. The performance of *canneal* is unaffected by this reallocation since *canneal* was not able to make use of its power budget in the first place. But the response time of *blackscholes* was reduced from 97.8 ms to 95.4 ms (2.5 percent). So, the overall performance is increased by reallocating unused power budget. *Similar observations also hold true for other combinations of tasks, where one task is memory-intense and another task is compute-intense.*

1.3 Novel Contributions

No previous work considers the impact of both power budget and LLC latency on task performance jointly while considering dynamic execution phases of tasks. Therefore, we make the following novel contributions in this work:

- We are the first to explore the trade-off between power budget and LLC latency for task-agnostic task mapping on S-NUCA many-cores. We develop a method to determine all Pareto-optimal mappings using an Integer Linear Program (ILP).
- We are the first to present a run-time task-mapping algorithm that improves performance by exploiting this trade-off while considering previously mapped tasks.
- We extend this algorithm by a run-time component that dynamically reallocates power budgets between threads to react to different task execution phases and task heterogeneity. This algorithm works in tandem with the mapping algorithm. The combined algorithms outperform the state-of-the-art w.r.t. performance, energy consumption and prevention of thermal violations.

Source Code. On acceptance of this paper, we will release the source code that facilitates reproducibility of results.

2 Related Work

A number of strategies have been proposed for task mapping and dynamic power budgeting on many-cores with various optimization goals and constraints. Most of the works focus on either task mapping or dynamic power budgeting, but there are also some works that consider these two related problems jointly.

Task mapping can be broadly classified into two classes: design-time and run-time mapping [17]. If the workload is known in advance, the mapping can be obtained at design-time. Authors of [18] present a design-time task mapping

algorithm formulated as an ILP that minimizes temperatures, thermal gradients, or energy under performance constraints. However, they do not consider task communication or cache architecture. If the workload is not known in advance, task mappings need to be determined at run-time. Most of existing run-time task mapping techniques assume point-to-point communication between threads. These works consider a variety of goals and constraints: task performance [19], [20], [21], communication latency [22], maximum temperature [22], reliability [21], or power consumption [19]. However, in a shared-memory architecture, threads do not communicate point-to-point. To the best of our knowledge, there is only one work that considers the heterogeneity w.r.t. AMD of cores on S-NUCA many-cores. Authors of [12] present a run-time task mapping algorithm *SNSched* that does consider this and thus improves the average task response time. However, they do not consider power or temperature.

There are a lot of related works that target the problem of power budgeting on many-cores [23]. Some of them use a priority-based greedy approach to distribute power in case of a global power budget shortage. Authors of [24] present an agent-based power budget reallocation technique. Each core is managed by an agent that receives an income based on its associated core's current temperature and it may spend that income on power budget. In the case of a global power budget shortage, the power budget is distributed based on predefined priorities. No trade-off is performed that could exploit the heterogeneity between threads. Authors of [25] present a two-layered approach for power-constrained many-cores. A centralized algorithm greedily distributes the available power budget to all cores based on their Instructions per Cycle (IPC). Threads with higher IPC are given preference, which aims to increase the energy efficiency. Other approaches achieve optimal results but rely on strong a priori knowledge. Authors of [26] present a *Dynamic Programming* approach to determine DVFS levels that maximizes the performance under a given power budget. Their approach relies on exact task-specific models for power consumption and performance depending on the frequency. All of these works use TDP and therefore they are not aware of the actual mapping and the thermal coupling of adjacent cores.

Some approaches have been proposed that target both task mapping and power budgeting. Again, task mapping can be done at design-time [27] or at run-time. *Bubble Budgeting* [28] is a run-time approach that aims to maximize performance under the constraint of thermal safety. The goal of the mapping policy is to map threads of a task spatially compactly with idle cores (*bubbles*) in between. The idle cores are used to prevent hotspots by dissipating heat. Threads are assumed to communicate point-to-point, hence compactness reduces the communication latency. The mapping is extended by a power budgeting algorithm that uses a heuristic to determine the DVFS-levels of all threads and refines this heuristic in a second step. To do this, the power consumption of threads at different DVFS-levels need to be known in advance.

PAT++ [29] is a run-time task mapping and power budgeting algorithm with similar goals and task models as *Bubble Budgeting*. Authors propose a heuristic algorithm to sparsely map threads of a task within a small region of the core. The motivation herein is similar to the previous approach. The idle cores dissipate heat but increase the point-to-point communication latency between threads. *PAT++* further tries to maximize the distance between threads of different tasks to minimize the mutual thermal impact of tasks. This task mapping algorithm is extended by both a power budgeting controller and a boosting controller. The power budget is calculated using *pessimistic* TSP, which only takes the number of active cores into account instead of the full mapping. The power budgeting controller reallocates surplus power budget according to tasks' priorities and network characteristics. Remaining thermal headroom is exploited by the boosting controller that allows threads to exceed their power budget for short periods of time.

To the best of our knowledge, there is no work that addresses the joint problem of task mapping and power budget reallocation on many-cores with logically-shared physically-distributed LLC. Most related works assume point-to-point communication of threads, which does not hold on this architecture and those works that consider the LLC characteristics do not consider power or temperature. A novel approach specifically tailored for this architecture is therefore required.

3 Pareto-Optimal Task Mappings

Task-agnostic mapping of k threads to n cores has two conflicting optimization goals: maximizing the power budget by TSP (and thus increase performance) and minimizing the maximum AMD (LLC latency). As explained in Section 1, these two metrics cannot both be simultaneously optimal. Hence, the goal of this section is to find all Pareto-optimal mappings [30]. For that purpose, we propose a 0-1 ILP.

3.1 Thermal Model

Thermal models of chips are commonly expressed as RC-thermal networks where temperature and heat flow correspond to voltage and current in an electrical circuit, respectively [5]. The many-core and its cooling system are modeled by N thermal nodes. The first n nodes correspond to the n cores of the many-core along with their private caches, LLC banks, and NoC routers. The remaining $N - n$ nodes describe its cooling system. The changes in temperatures $\mathbf{T'} = [T'_i]_N$ of all nodes are given by N differential equations

$$\mathbf{AT'} + \mathbf{BT} = \mathbf{P} + T_{amb}\mathbf{G}.$$

Thereby, matrix $\mathbf{A} = [A_{ij}]_{N\times N}$ describes the dynamic thermal behavior, vector $\mathbf{T} = [T_i]_N$ contains the current temperatures, matrix $\mathbf{B} = [B_{ij}]_{N\times N}$ describes the thermal conductivity between nodes, vector $\mathbf{P} = [P_i]_N$ contains the current power consumption of all nodes, T_{amb} is the ambient temperature, and vector $\mathbf{G} = [G_i]_N$ describes the thermal conductivity between nodes and ambient.

The steady-state temperatures $\mathbf{T_{steady}}$ are given by

$$\mathbf{T}_{steady} = \mathbf{B}^{-1}(\mathbf{P} + T_{amb}\mathbf{G}), \tag{1}$$

where matrix $\mathbf{B}^{-1} = [B^{-1}_{ij}]_{N\times N}$ is the inverse of matrix $\mathbf{B}$. None of the steady-state core temperatures must exceed the thermal threshold T_{DTM} for thermally safe operation. We

TABLE 1
Notations Used in This Paper

General	
n	Number of cores
N	Number of thermal nodes of the RC-thermal model
P_{TDP}	Thermal design power
P_{TSP}	Thermal safe power associated with a certain mapping
P_{idle}	Power consumption of idle cores
$\mathbf{A} = [A_{ij}]_{N\times N}$	Dynamic thermal behavior
$\mathbf{B} = [B_{ij}]_{N\times N}$	Thermal conductivity between nodes
$\mathbf{G} = [G_i]_N$	Thermal conductivity between nodes and ambient
$\mathbf{T} = [T_i]_N$	Current temperatures
$\mathbf{T_{steady}} = [T_{steady,i}]_N$	Steady-state temperatures
$\mathbf{P} = [P_i]_N$	Current power consumption
T_{amb}	Ambient temperature
T_{DTM}	Critical temperature
Task mapping	
k	Number of requested cores
$\mathbf{x} = [x_i]_n$	Task mapping mask
H	Helper variable for ILP
AMD_{max}	Maximum AMD
MC	Set of mapping candidates
$TSP(M)$	Per-core power budget using TSP for the given mapping M
$AMD(i)$	AMD of core i to LLC banks
α	Parameter to balance impact of power budget and AMD
Power budget reallocation	
P_b	Power budget of a core (non-uniform)
P	Current power consumption of a core
U	Current utilization of a core
δ	Hysteresis to compensate fluctuations in the power consumption

additionally consider a global power budget P_{TDP}. Idle cores consume a constant power P_{idle}. Table 1 gives an overview over all variable and parameters used in this work.

3.2 ILP Formulation

The optimization variables of the ILP are $\mathbf{x} = [x_i]_n$, $x_i \in \{0, 1\}$, where $x_i = 1$ means that core i is active, whereas $x_i = 0$ means that core i is idle. A mapping and its associated power budget need to fulfill three constraints:

- Exactly k cores need to be selected.
- The global power budget P_{TDP} must not be exceeded.
- There must be no thermal violations in the steady-state, i.e. the per-core power budgets P_{TSP} must be satisfied.

Selecting exactly k cores is met if $\sum_{i=1}^{n} x_i = k$. For the remaining two constraints we introduce a helper variable $H > 0$ that allows us to express them as linear inequalities. H is defined by

$$P_{TSP} = P_{idle} + \frac{1}{H}. \tag{2}$$

The value of $\frac{1}{H}$ corresponds to the difference between the power budget of active cores and the power consumption of idle cores. This value is the same for all active cores, because we employ uniform TSP. The power budget is maximized if the helper variable H is minimized. The total power consumption of the chip consists of k active cores, each consuming at most the power budget P_{TSP}, and $n - k$ idle cores, each consuming the idle power P_{idle}. The total power consumption must not exceed the global power budget P_{TDP}

$$P_{TDP} \geq k \cdot P_{TSP} + (n - k)P_{idle}.$$

Using Equation (2), solving for H gives

$$H \geq \frac{k}{P_{TDP} - n \cdot P_{idle}}.$$

The steady-states is thermally safe if $\forall i : T_i \leq T_{DTM}$. Using Equations (1) and (2), we obtain

$$T_i = \sum_{j=1}^{N} B_{ij}^{-1} \cdot \left(P_{idle} + x_i \cdot \frac{1}{H} + T_{amb} \cdot G_j \right) \leq T_{DTM}.$$

Solving for H

$$H \geq \frac{\sum_{j=1}^{n} B_{ij}^{-1} \cdot x_j}{T_{DTM} - \sum_{j=1}^{n} B_{ij}^{-1}(P_{idle} + T_{amb} \cdot G_j)}.$$

Putting all three constraints together, the power-budget-maximizing task mapping and its power budget is obtained by solving the following ILP:

$$\text{Minimize } H,$$

such that

$$\sum_{i=1}^{n} x_i = k \qquad H \geq \frac{k}{P_{TDP} - n \cdot P_{idle}}$$

$$H \geq \frac{\sum_{j=1}^{n} B_{ij}^{-1} \cdot x_j}{T_{DTM} - \sum_{j=1}^{n} B_{ij}^{-1}(P_{idle} + T_{amb} \cdot G_j)} \;\; \forall i \in [1 \ldots n].$$

This ILP does not yet account for LLC latency. Since the AMD values of the cores are discrete and symmetric, there exist only a small number of unique values [12]. For example, the 64-core many-core used in this work has only 9 unique AMD values. That enables us to find all Pareto-optimal mappings by fixing the maximum AMD AMD_{max} in the mapping and maximizing the power budget for this value of AMD_{max}. Therefore, we add a fourth constraint to the ILP that restricts the available cores in the mapping to cores with a smaller AMD than the upper limit AMD_{max}.

$$x_i = 0 \quad \forall i : AMD(i) > AMD_{max}.$$

The resulting ILP is solved several times, once for each unique AMD value AMD_{max}. This allows to find all Pareto-optimal mappings.

The above ILP implicitly makes the assumption that the many-core is idle when k new cores need to be assigned, which is in practice almost never the case. In case there are already p cores allocated on the many-core, the ILP needs to be modified slightly. Instead of finding a mapping of k threads, a mapping of $k + p$ threads is searched while

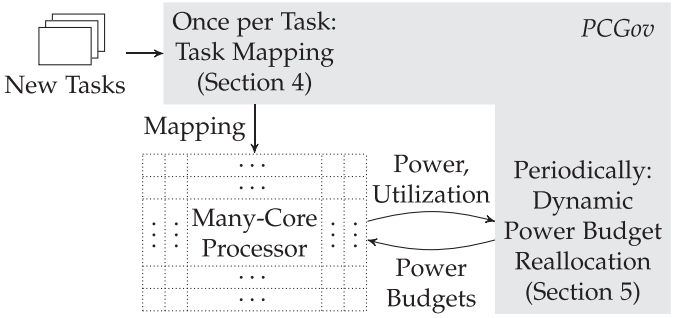


Fig. 8. The proposed algorithm *PCGov* comprises two parts: task mapping and dynamic power budget reallocation. Task-agnostic task mapping is only called once per task. Dynamic power budget reallocation periodically sets the power budgets according to the current power consumptions and core utilizations.

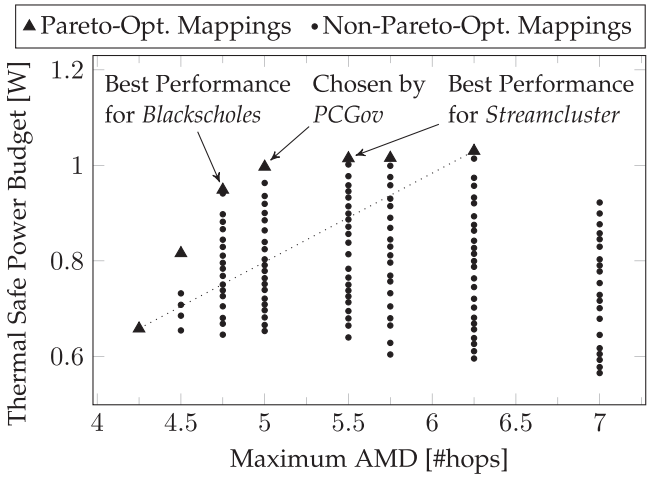


Fig. 9. Pareto-curve of power budgets and maximum AMDs of different mappings of twelve threads to a 64-core many-core. The best performance for *blackscholes* and *streamcluster* is attained with mappings that perform a trade-off between power budget and AMD. Our algorithm *PCGov*, which is designed to be task-agnostic, selects the mapping that equally balances power budget and AMD.

enforcing that $x_i = 1$ if core i is already used by the p present threads to account for task rigidity.

Solving this ILP is too computationally expensive for use at run-time. Determining the optimal mapping of 8 cores on a 64-core many-core takes 8.7 s using *MATLAB* on an *Intel Core i5-3470*. Calculating the optimal mappings at design-time and storing them in a lookup table is not feasible either since there are too many possible scenarios. This creates the need for an efficient heuristic run-time algorithm that finds mappings close to the Pareto-optimal ones, so-called near-Pareto-optimal mappings. A near-Pareto-optimal has only an insignificantly lower power budget than the Pareto-optimal mapping with the same maximum AMD of used cores. The heuristic algorithm is presented in the next section and its efficiency is evaluated empirically using the proposed ILP.

4 Run-Time Task Mapping Algorithm

This section presents our run-time task mapping algorithm. It forms the first part of our proposed algorithm *PCGov*, which is depicted in Fig. 8. Dynamic power budget reallocation forms the second part and is described in Section 5. Since the mapping is to be decided when a new task arrives, and hence before the task starts execution, the mapping algorithm needs to be task-agnostic. The mapping algorithm comprises two parts:

1) Find near-Pareto-optimal mappings.
2) Select one of these mappings that is expected to maximize performance.

4.1 Find Near-Pareto-Optimal Mappings

Algorithm 1 presents the heuristic algorithm to find near-Pareto-optimal mapping candidates MC. It finds a near-Pareto-optimal mapping M for each unique AMD value AMD_{max} where only cores that do not exceed AMD_{max} are used. The k cores of each mapping M are selected *greedily* starting with an empty mapping (line 3). The available cores are the idle cores that do not exceed the AMD limit (line 5). For each of these cores, the resulting power budget is calculated, one at a time, in combination with the already selected cores (line 6). The core that results in the highest power budget is added to the mapping M (line 7). This is repeated until k cores are selected.

Calculating the TSP for all available cores can be done in $O(n^2)$ by sharing intermediate results across the individual calculations. Since this needs to be done k times for each of the $O(n)$ unique AMD values, the total complexity for finding all these near-Pareto-optimal mappings is $O(k \cdot n^3)$.

Algorithm 1. Find Near-Pareto-Optimal Mappings

Input: k: number of requested cores, U: set of used cores
Output: near-Pareto-optimal mapping candidates MC
1: $MC \leftarrow \varnothing$
2: **for all** $AMD_{max} \in \text{UNIQUE}\ (AMD(i)|i \in [1 \dots n])$ **do**
3: $M \leftarrow \varnothing$ ▷ start with empty mapping
4: **while** $|M| < k$ **do** ▷ *greedily* select k cores
5: $avblCores \leftarrow \{i|\text{AMD}(i) \leq AMD_{max}\} \setminus M \setminus U$
6: $P_{TSP,all} \leftarrow \{\text{TSP}(M \cup \{i\} \cup U)|i \in avblCores\}$
7: $M \leftarrow M \cup \{arg\,max_i\ P_{TSP,all}(i)\}$
8: **end while**
9: $MC \leftarrow MC \cup \{M\}$
10: **end for**
11: **return** MC

4.2 Select One of the Near-Pareto-Optimal Mappings

After the near-Pareto-optimal mappings have been determined, one of them needs to be selected. The impact of power budget and LLC latency on the performance differs for different tasks as shown in Fig. 4d. Selecting the mapping that maximizes performance for a task, can only be accomplished with detailed profiling. Hence, the goal of our task-agnostic algorithm *PCGov* is to select the mapping that is expected to result in high performance for *most* tasks.

Fig. 9 shows the Pareto-curve for mapping twelve threads on a 64-core many-core. The best *blackscholes* or *streamcluster* performance is obtained with neither the mapping with highest power budget nor the one with lowest LLC latency, but with mappings that perform a trade-off between both metrics. Fig. 4d shows that similar observations hold true for almost all other *PARSEC* tasks. We use the weighted sum of power budget $\text{TSP}(M)$ and maximum

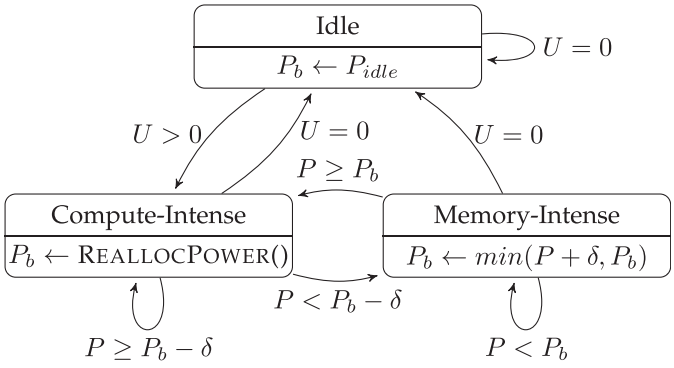


Fig. 10. FSM to track the state of a thread and reallocate power budgets. Each state has its own policy to determine the power budget.

AMD of the selected cores $max_{i \in M}\,\mathrm{AMD}(i)$ to decide the selected mapping $M^{\star}$.

$$M^{\star} = \underset{M \in MC}{arg\,max} \left(\mathrm{TSP}(M) - \alpha \cdot \underset{i \in M}{max}\ \mathrm{AMD}(i)\right).$$

The parameter α balances the impact of power budget and AMD on the selection. Without task profiling, we aim to make a balanced trade-off between power budget and LLC latency. We calculate α for each set of mapping candidates individually

$$\alpha = \frac{\underset{M \in MC}{max}\ \mathrm{TSP}(M) - \underset{M \in MC}{min}\ \mathrm{TSP}(M)}{\underset{M \in MC}{max}\ \underset{i \in M}{max}\ \mathrm{AMD}(i) - \underset{M \in MC}{min}\ \underset{i \in M}{max}\ \mathrm{AMD}(i)}.$$

This formula makes the selection independent of the scales that are used. Graphically, our algorithm selects the mapping with the highest distance to the dotted line in Fig. 9 that connects the mapping with the highest power budget and the mapping with the lowest maximum AMD. Selecting one of the mapping candidates has a time complexity of $O(n)$, which results in the total complexity $O(k \cdot n^3)$ for task mapping. This algorithm is to be run only once whenever a new task arrives.

5 Run-Time Dynamic Power Budget Reallocation Algorithm

This section describes our run-time power budget reallocation algorithm, which forms the second part of *PCGov*. It observes the current core utilizations and power consumptions of threads and adjusts their power budgets accordingly.

Fig. 10 shows the Finite State Machine (FSM) that is associated with each thread. It comprises three states based on the core utilization and power consumption: *Idle*, *Compute-Intense*, and *Memory-Intense*. A thread stays in the *Compute-Intense* state as long as its power consumption potentially could exceed the power budget if the core ran at a higher frequency. The DVFS loop throttles the core to prevent a power budget violation. Hence, the observed power consumption will be slightly lower than the power budget. If a thread cannot exceed its associated power budget, i.e. if the power consumption is significantly lower than the power budget, the FSM transitions to the *Memory-Intense* state. The parameter δ inhibits oscillations between states by compensating small fluctuations in the power consumption. A thread transitions back to the *Compute-Intense* state if its power consumption rises again. If a thread is idle, i.e. its core utilization is zero, the FSM transitions to the *Idle* state.

The policy to determine the power budget depends on the state. Idle threads are threads that are blocked and wait for an event before resuming operation. The power budget of these threads is set to the bare minimum P_{idle}, which forces the DVFS control loop to choose the lowest possible voltage and frequency for the associated core. This results in the highest possible surplus power budget available for other threads. The power consumption of memory-intense threads does not reach their power budget. Hence, their power budget can safely be reduced to a level slightly higher than their current power consumption to free the otherwise wasted power budget. *This does not degrade their performance even though their power budget is reduced*, but allows to reallocate otherwise wasted power budget to threads that can make use of it. Compute-intense threads are the threads whose performance could increase with a higher power budget. Hence, their power budget should be maximized.

Algorithm 2. REALLOCPOWER: Reallocate Power Budgets

Input: $P_{b,I+M}(i)$: Power budgets of *idle* and *memory-intense* threads,
Output: $P_{b,C}(i)$: Power budgets of *compute-intense* threads.
1: $C \leftarrow \{i \,|\, \text{Core } i \text{ is compute-intense}\}$
2: $P_C \leftarrow \mathrm{TSP}(C \,|\, P_{b,I+M})$
3: **for all** $i \in C$ **do**
4: $\quad P_{b,C}(i) \leftarrow P_C$
5: **end for**

It is important to note that power budget that was freed on any thread cannot be simply added to the power budget of another thread. Care must be taken to prevent new thermal hotspots. *The sum of all per-core power budgets may change if power is reallocated to ensure thermal safety*. Algorithm 2 increases the power budget for compute-intense threads in a thermal-aware fashion. First, the set C of cores that receive power budget is determined. Then, the thermally safe power budget for these cores is recalculated using TSP while considering the already determined power budget for the other cores. TSP supports thermal nodes with known power consumption. To use this, we temporarily assume that the power consumptions of idle and memory-intense threads exactly match their power budgets and therefore overestimate their power consumption by at most δ.

The FSMs of the threads are mostly independent. This allows for a distributed implementation. One FSM instance is run on every core. These FSMs independently track the state of their associated threads. However, the calculation of power budgets for compute-intense threads needs information exchange between the FSMs. To do this, all FSMs in the *Idle*- and *Memory-Intense*-state determine their power budget and send it to a central manager that calculates the power budget for *compute-intense* threads applying Algorithm 2 and broadcasts it to all associated FSMs.

Each FSM of one thread has a time complexity $O(1)$. Since TSP is to be recalculated after reallocating the power budget, Algorithm 2 has a complexity of $O(n^2)$. However, Algorithm 2 does not need to be run in every DVFS-epoch. This is needed only if the number of threads in the system changes or if the state of the FSM of a thread changes.

6 Experimental Evaluation

6.1 Experimental Setup

We use the *Sniper* many-core simulator [31] to simulate multi-program execution of multi-threaded tasks in order to evaluate our algorithm *PCGov*. *Sniper* models shared resource contention while still offering a feasible simulation time. We use an extension of *Sniper*, *HotSniper* [32] that combines *Sniper* with periodic power traces using the integrated *McPAT* tool [33] and a thermal simulation using *HotSpot* [34]. The simulated many-core comprises 64 cores aligned in an 8×8 grid that communicate over a NoC. DVFS sets core frequencies in multiples of 100 MHz from a minimum of 1.0 GHz up to a maximum of 4.0 GHz. The per-core private L1 data and instruction caches have a size of 16 KB each and an access latency of 3 cycles. The shared LLC uses S-NUCA policy and has a total size of 8 MB where each core holds an LLC bank of size 128 KB with a latency of 8 cycles. The NoC uses XY-routing with a latency of 1.5 ns (6 CPU cycles @ 4 GHz) per hop and a link width of 256 bits. The many-core is assumed to be fabricated in the 10 nm technology. We used *HotSpot* to create a thermal model of the many-core and its cooling system. TDP is set at 30 W. The ambient temperature is 45 °C and the thermal threshold T_{DTM} is 80 °C. The power consumption of an idle core is set at 0.2 W due to the use of its associated LLC bank even when the core itself is idle. The parameter δ is set at 50 mW.

Each workload consists of 20 randomly selected *PARSEC* tasks with *simsmall* inputs as they are large enough to stress the caches for most tasks[2] but still allow the simulation to finish in a feasible time. We use 8 of the 13 *PARSEC* tasks: *blackscholes*, *bodytrack*, *canneal*, *dedup*, *fluidanimate*, *streamcluster*, *swaptions* and *x264*. The two tasks *facesim* and *raytrace* were not used since they do not offer a *simsmall* input and thus take unreasonably long to execute. The three tasks *ferret*, *freqmine*, and *vips* could not be evaluated due to unresolvable instrumentation errors of the *PIN* tool [35] that is used by *Sniper*. The task arrival rates follow a Poisson-distribution with varying average arrival rates.

6.2 Evaluation of the Mapping Candidates of *PCGov*

The first part of the mapping algorithm of *PCGov* finds near-Pareto-optimal mappings. We evaluate how close these mappings are to the optimal by extracting all near-Pareto-optimal mappings that were obtained while executing the workloads. These are all the mapping candidates MC, not just the selected mappings. We compare the power budget of each of these mappings to the power budget of the Pareto-optimal mapping with the same maximum AMD. The Pareto-optimal mappings are obtained using the ILP described in Section 3. Fig. 11 shows the results for each number of requested cores k. For small values of k, the mappings obtained by our greedy algorithm are very close to optimal or even optimal. For larger values, the quality drops slightly. This is because the search space expands combinatorially with the number of requested cores k, and hence, the likelihood for a greedy algorithm to find the optimal solution decreases. Overall, the greedy algorithm determines near-Pareto-optimal mappings with a, in practice, negligible loss in power budget of less than 3.2 percent compared to the Pareto-optimal mappings.

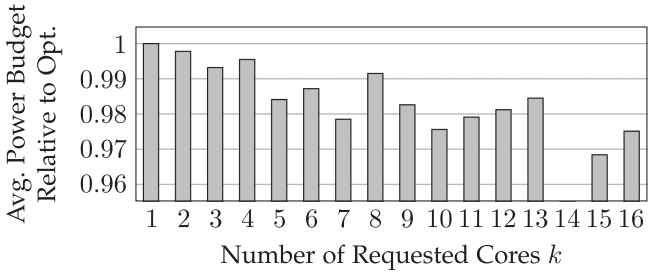


Fig. 11. Power budget of the near-Pareto-optimal mapping candidates obtained by *PCGov* relative to the Pareto-optimal mappings obtained by the ILP.

6.3 Comparison to State-of-the-Art

We compare our run-time task mapping algorithm *PCGov* to the state-of-the-art algorithms *Bubble Budgeting* [28], and *PAT++* [29] described in Section 2, since these algorithms address the same problem as studied in this work. We adapted *Bubble Budgeting* and *PAT++* to our platform.

Fig. 12a compares the average response time of the tasks for different task arrival rates for each of the three task mapping and DVFS policies. In these experiments, the average (peak) system utilization varies from 4.3 percent (34 percent) for a task arrival rate of 1 per 1,000 ms up to 34 percent (91 percent) for a task arrival rate of 11 per 1,000 ms. The response times of individual tasks vary significantly from 24 ms up to 2.3 s. In order to prevent long-running tasks from dominating the average response time, we compare the geometric means of the response times. *PCGov* always results in the lowest response time and hence in the highest performance with up to 21 and 13 percent improvement over *PAT++* and *Bubble Budgeting*, respectively. This is because *PCGov* considers not only the power budget but also the LLC latency. *PAT++* and *Bubble Budgeting* consider power budget and task communication but assume message passing, which is not the case in S-NUCA many-cores. Hence, these approaches tend to map tasks to the corners of the many-core and thus result in high LLC latencies and reduced performance. These observations hold true for varying task arrival rates.

Fig. 12b compares the average energy consumption per task for execution of the workloads. The energy consumption shows similar trends as the performance. *PCGov* always

2. Fig. 3c shows that the LLC-related components mem-ifetch and mem-nucad take up a large part of the cycle stack, which shows that L1 caches cannot fit the working set of *streamcluster*, but the LLC needs to be used extensively. Similar behavior can be seen for most other tasks. Due to heterogeneity in *PARSEC*, some tasks (*canneal*) depend strongly on the DRAM, while others (*fluidanimate*) depend very little on the memory hierarchy. By employing all these tasks in our evaluation we show that our proposed technique is robust w.r.t. variations in the tasks.

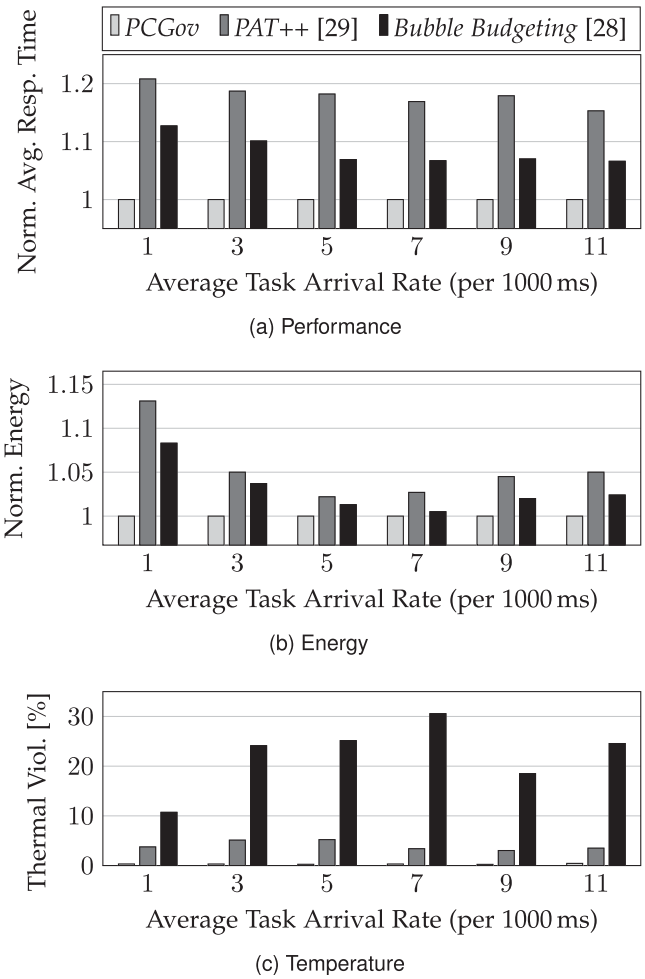


Fig. 12. Comparison of *PCGov*, *PAT++* [29] and *Bubble Budgeting* [28] policies for multi-program workloads at different arrival rates. *PCGov* results in significantly lower response time, lower energy consumption, and fewer thermal violations than *PAT++* and *Bubble Budgeting*.

results in the lowest energy consumption for all task arrival rates with up to 13 and 8.3 percent improvement over *PAT++* and *Bubble Budgeting*, respectively.

Fig. 12c compares how often thermal violations occur during execution. This is mainly a property of the power budgeting and DVFS algorithm, not of the task mapping. The power budgeting policy of *PCGov* is designed to prevent thermal violations. The resulting power budget is enforced on a per-core basis. This results in very rare thermal violations of less than 0.4 percent. Since we employ a reactive approach, thermal violations due to sudden changes in the power consumption of threads cannot be perfectly avoided. *PAT++* uses a per-chip power budget, which cannot prevent thermal hotspots. Hence, the duration of thermal violations is higher (up to 5.2 percent). The thermal model used by *Bubble Budgeting* determines the power budget of a core by considering its neighborhood. However, the model does not consider the location of the core on the chip. The cores in the corners of the many-core have a significantly lower power budget compared to other cores since they have fewer neighbors that can dissipate heat. *Bubble Budgeting* cannot consider this and hence it causes frequent thermal violations (up to 31 percent), mostly in the corner cores.

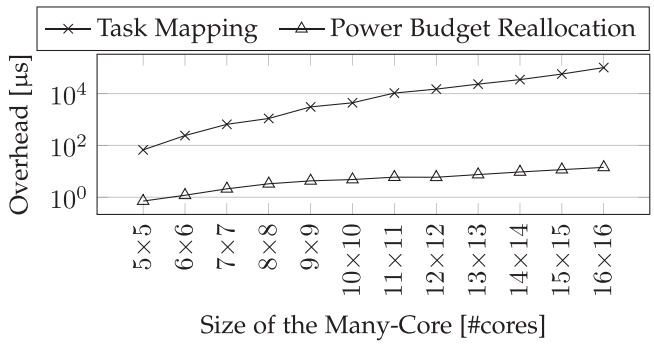


Fig. 13. Average overhead per execution for task mapping and dynamic power budget reallocation depending on the size of the many-core.

Summarizing, the proposed *PCGov* shows substantial improvements over both *PAT++* [29] and *Bubble Budgeting* [28]. It significantly reduces response time, energy consumption and thermal violations.

6.4 Run-Time Overhead

Performance Impact. Fig. 13 shows the average run-time overhead of our algorithm *PCGov*. The overhead is split into two components: task mapping and power budget reallocation. Task mapping is done only once per task, while power budget reallocation is done periodically. The overhead is given per execution of the algorithms, i.e. per task mapping and per power budget reallocation epoch, respectively.

The overhead of task mapping is measured by creating random scenarios, where the number and location of already active cores is set randomly at a utilization of 50 percent and a random number of requested cores k is selected between 1 and 16. The average overhead increases with the number of cores from 67 μs on a 5×5 many-core up to 101 ms on a 16×16 many-core when running on a single core. It is important to note that mapping is only performed once per task. The average execution time of tasks in the experiments from Section 6.3 is 340 ms. This results in a relative overhead of 30 percent for task mapping on a 16×16 many-core. However, up to 10×10 cores, the overhead stays below 1.3 percent. We conclude that our proposed algorithm is suitable for many-cores with up to 10×10 cores, but may be too slow for larger processors. On the studied 8×8 many-core, task mapping takes 1 ms on average which results in an overhead of 0.3 percent.

When measuring the overhead for the power budget reallocation, it needs to be considered that Algorithm 2, which calculates the power budget for *compute-intense* tasks, does not need to be executed in every epoch. The less the power consumption of active tasks fluctuates, the fewer this algorithm is executed. However, this heavily depends on the executed workloads. To account for this, we extract power and utilization traces from the workloads executed in Section 6.3 to stimulate the power budgeting algorithm and measure its execution time. Algorithm 2 had to be executed in less than 10 percent of all power budgeting epochs. The overhead increases with the size of the many-core from 0.7 μs on a 5×5 many-core up to 14.2 μs on a 16×16 many-core. Considering the power budget reallocation period of 100 μs, this results in a relative overhead of less than 0.1 percent for all studied sizes of the many-core, since

Algorithm 2 is executed on only one of the cores. The average overhead on our studied 8×8 many-core is 3.3 μs.

Power and Thermal Impact. Our proposed technique is executed at run-time, i.e., in parallel to the executed workload. Therefore, we need to investigate the impact of our technique on power and temperature. Executing task mapping (Algorithm 1 and selection of the best candidate) on a single core results in a power consumption of up to 1.4 W. This is comparable to the power consumption of actual tasks, which ranges from 1.1 W to 2.8 W (see Fig. 6). Therefore, it needs to be considered by the dynamic power budget reallocation algorithm to maintain thermally safe operation. However, since the power budget algorithm is agnostic of tasks and simply considers threads independently, no changes to the power budget reallocation algorithm are required. The thread that decides the mapping can be treated exactly as a thread of any other task.

The power budget reallocation algorithm is executed periodically and in therefore also runs in parallel to the executed tasks. As seen earlier, the execution time of this algorithm is 3.3 μs every 100 μs on a single core. The peak power consumption during its execution is 1.7 W. Since the control period is small compared to thermal time constants of a processor cooling system, the thermal impact can be treated as constant and we only need to consider the average power consumption over the full control period, i.e., including 96.7 μs idle time. The average power consumption is 0.25 W, which is very close to the idle power consumption of 0.2 W. Therefore, the power budget reallocation algorithm has a negligible thermal impact on the many-core.

6.5 Impact of the Hysteresis Parameter δ

The hysteresis parameter δ is employed to reduce the power budget reallocation overhead. Without δ (i.e., δ=0 mW), the power budget of compute-intense tasks has to be recalculated every time the power consumption of a memory-intense task changes. This is the case in every control step due to noise. The hysteresis δ avoids this by creating a corridor for the power consumption. As long as the power consumption stays within this corridor, power budgets do not have to be recalculated. The wasted power budget (the difference between power budget and actual power consumption) is bounded by δ. Summarizing, δ should be high enough to filter out noise in the power consumption, but as low as possible to bound the wasted power budget.

Fig. 14 shows the average power budget reallocation overhead depending on the hysteresis δ. The overhead is highest for δ=0 mW where it is 16 s per control period (100 s) when executed on a single core. The overhead decreases strongly with increasing δ, but only slightly decreases above 50 mW. We choose δ=50 mW, because this value allows to filter out fluctuations in the power consumption due to noise, but still is small compared to the absolute power consumption of tasks of up to 2.8 W (see Fig. 6).

7 Conclusion

We presented a run-time task mapping and power budgeting algorithm called *PCGov*. This algorithm exploits a trade-off between power budget and LLC latency for task mapping. We showed that optimizing for power budget and optimizing for LLC latency result in contradictory mapping decisions. Optimizing for only one of the two does not provide the best task performance. Instead, a trade-off between the two is most beneficial. Furthermore, *PCGov* exploits heterogeneity in the power consumption both within threads of the same task and between threads of different tasks. Reallocating power budget to threads with a higher demand results in increased performance. Both measures combined results in up to 21 percent higher performance compared to a state-of-the-art algorithm on a 64-core many-core accompanied by a reduction of up to 13 percent in the energy consumption. We show that our power budget reallocation algorithm is very effective in avoiding thermal violations. Furthermore, the overhead of the proposed algorithm on a 64-core many-core is less than 0.5 and 0.1 percent for task mapping and periodic power budget reallocation, respectively.

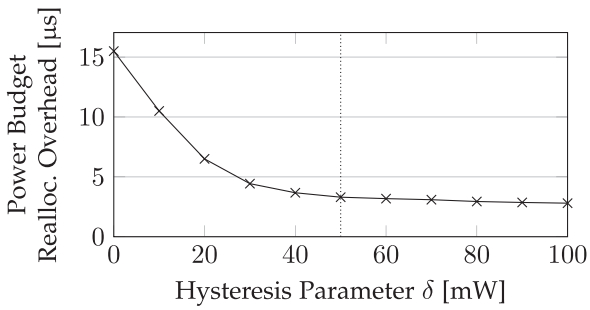


Fig. 14. Impact of varying the hysteresis parameter δ on the performance and dynamic power budget reallocation overhead. Setting δ at 50 mW allows to filter most of the noise in the power consumption.

Acknowledgments

This work was supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Centre "Invasive Computing" (SFB/TR 89). A subset of this work was first presented in ISLPED'18 [1].

References

[1] 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 *Proc. Int. Symp. Low Power Electron. Des.*, 2018, pp. 1–6.

[2] J. Henkel, A. Herkersdorf, L. Bauer, T. Wild, M. Hübner, R. K. Pujari, A. Grudnitsky, J. Heisswolf, A. Zaib, B. Vogel, et al., "Invasive manycore architectures," in *Proc. 17th Asia South Pacific Des. Autom. Conf.*, 2012, pp. 193–200.

[3] J. Dongarra, S. Tomov, P. Luszczek, J. Kurzak, M. Gates, I. Yamazaki, H. Anzt, A. Haidar, and A. Abdelfattah, "With extreme computing, the rules have changed," *Comput. Sci. Eng.*, vol. 19, no. 3, pp. 52–62, 2017.

[4] T. S. Muthukaruppan, M. Pricopi, V. Venkataramani, T. Mitra, and S. Vishin, "Hierarchical power management for asymmetric multi-core in dark silicon era," in *Proc. Des. Autom. Conf.*, 2013, Art. no. 174.

[5] 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," *IEEE Trans. Comput.*, vol. 66, no. 1, pp. 147–162, Jan. 2017.

[6] M. R. Garey and D. S. Johnson, "Complexity results for multiprocessor scheduling under resource constraints," *SIAM J. Comput.*, vol. 4, no. 4, pp. 397–411, 1975.

[7] E. L. de Souza Carvalho, N. L. V. Calazans, and F. G. Moraes, "Dynamic task mapping for MPSoCs," *IEEE Des. Test Comput.*, vol. 27, no. 5, pp. 26–35, Sep./Oct. 2010.
[8] D. G. Feitelson and L. Rudolph, "Metrics and benchmarking for parallel job scheduling," in *Proc. Workshop Job Scheduling Strategies Parallel Process.*, 1998, pp. 1–24.
[9] S. Boyd-Wickizer, H. Chen, R. Chen, Y. Mao, M. F. Kaashoek, R. Morris, A. Pesterev, L. Stein, M. Wu, Y.-H. Dai, et al., "Corey: An operating system for many cores," in *Proc. Symp. Operating Syst. Des. Implementation*, 2008, vol. 8, pp. 43–57.
[10] C. Ramey, "TILE-Gx100 ManyCore processor: Acceleration interfaces and architecture," in *Proc. IEEE Hot Chips Symp.*, 2011, pp. 1–21.
[11] C. Kim, D. Burger, and S. W. Keckler, "An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches," in *Proc. Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2002, pp. 211–222.
[12] A. Pathania and J. Henkel, "Task scheduling for many-cores with S-NUCA caches," in *Proc. Des. Autom. Test Europe Conf. Exhibition*, 2018, pp. 557–562.
[13] W. Heirman, T. E. Carlson, S. Che, K. Skadron, and L. Eeckhout, "Using cycle stacks to understand scaling bottlenecks in multi-threaded workloads," in *Proc. Int. Symp. Workload Characterization*, 2011, pp. 38–49.
[14] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *Proc. Int. Conf. Parallel Archit. Compilation Techn.*, 2008, pp. 72–81.
[15] A. Sodani, R. Gramunt, J. Corbal, H.-S. Kim, K. Vinod, S. Chinthamani, S. Hutsell, R. Agarwal, and Y.-C. Liu, "Knights landing: Second-Generation Intel Xeon Phi Product," *IEEE Micro*, vol. 36, no. 2, pp. 34–46, Mar./Apr. 2016.
[16] S. M. PD, H. Yu, and K. Wang, "3D many-core microprocessor power management by space-time multiplexing based demand-supply matching," *IEEE Trans. Comput.*, vol. 64, no. 11, pp. 3022–3036, Nov. 2015.
[17] P. K. Sahu and S. Chattopadhyay, "A survey on application mapping strategies for network-on-chip design," *J. Syst. Archit.*, vol. 59, no. 1, pp. 60–76, 2013.
[18] A. K. Coskun, T. S. Rosing, K. A. Whisnant, and K. C. Gross, "Temperature-aware MPSoC scheduling for reducing hot spots and gradients," in *Proc. Asia South Pacific Des. Autom. Conf.*, 2008, pp. 49–54.
[19] A. Schranzhofer, J.-J. Chen, and L. Thiele, "Dynamic power-aware mapping of applications onto heterogeneous MPSoC platforms," *IEEE Trans. Ind. Inf.*, vol. 6, no. 4, pp. 692–707, Nov. 2010.
[20] I. Anagnostopoulos, V. Tsoutsouras, A. Bartzas, and D. Soudris, "Distributed run-time resource management for malleable applications on many-core platforms," in *Proc. Des. Autom. Conf.*, 2013, Art. no. 168.
[21] A. Das, A. Kumar, and B. Veeravalli, "Reliability-driven task mapping for lifetime extension of networks-on-chip based multiprocessor systems," in *Proc. Des. Autom. Test Europe Conf. Exhibition*, 2013, pp. 689–694.
[22] D. Zhu, L. Chen, T. M. Pinkston, and M. Pedram, "TAPP: Temperature-aware application mapping for NoC-based many-core processors," in *Proc. Des. Autom. Test Europe Conf. Exhibition*, 2015, pp. 1241–1244.
[23] A. K. Singh, C. Leech, B. K. Reddy, B. M. Al-Hashimi, and G. V. Merrett, "Learning-based run-time power and energy management of multi/many-core systems: Current and future trends," *J. Low Power Electron.*, vol. 13, no. 3, pp. 310–325, 2017.
[24] T. Ebi, D. Kramer, W. Karl, and J. Henkel, "Economic learning for thermal-aware power budgeting in many-core architectures," in *Proc. IEEE/ACM/IFIP Int. Conf. Hardware/Software Codes. Syst. Synthesis*, 2011, pp. 189–196.
[25] Z. Chen and D. Marculescu, "Distributed reinforcement learning for power limited many-core system performance optimization," in *Proc. Des. Autom. Test Europe Conf. Exhibition*, 2015, pp. 1521–1526.
[26] X. Wang, B. Zhao, L. Wang, T. Mak, M. Yang, Y. Jiang, and M. Daneshtalab, "A pareto-optimal runtime power budgeting scheme for many-core systems," *Microprocessors Microsyst.*, vol. 46, pp. 136–148, 2016.
[27] H. Khdr, S. Pagani, M. Shafique, and J. Henkel, "Thermal constrained resource management for mixed ILP-TLP workloads in dark silicon chips," in *Proc. Des. Autom. Conf.*, 2015, Art. no. 179.
[28] X. Wang, A. K. Singh, B. Li, Y. Yang, H. Li, and T. Mak, "Bubble budgeting: Throughput optimization for dynamic workloads by exploiting dark cores in many core systems," *IEEE Trans. Comput.*, vol. 67, no. 2, pp. 178–192, Feb. 2018.
[29] A. Kanduri, M.-H. Haghbayan, A. M. Rahmani, M. Shafique, A. Jantsch, and P. Liljeberg, "adBoost: Thermal aware performance boosting through dark silicon patterning," *IEEE Trans. Comput.*, vol. 67, no. 8, pp. 1062–1077, Aug. 2018.
[30] S. Wildermann, M. Glas, and J. Teich, "Multi-objective distributed run-time resource management for many-cores," in *Proc. Des. Autom. Test Europe Conf. Exhibition*, 2014, pp. 1–6.
[31] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2011, Art. no. 52.
[32] A. Pathania and J. Henkel, "HotSniper: Sniper-based toolchain for many-core thermal simulations in open systems," *IEEE Embedded Syst. Lett.*, vol. 11, no. 2, pp. 54–57, Jun. 2019.
[33] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "The McPAT framework for multicore and manycore architectures: Simultaneously modeling power, area, and timing," *ACM Trans. Archit. Code Optim.*, vol. 10, no. 1, 2013, Art. no. 5.
[34] 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," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 14, no. 5, pp. 501–513, May 2006.
[35] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: Building customized program analysis tools with dynamic instrumentation," *ACM SIGPLAN Notices*, vol. 40, no. 6, pp. 190–200, 2005.

Martin Rapp received the BSc degree with distinction and MSc degree with distinction in computer science from the Karlsruhe Institute of Technology, in 2014 and 2016, respectively. Currently, he is working toward the PhD degree under the supervision of Prof. Dr. Jörg Henkel. He joined the chair for Embedded Systems in November 2017 as a research assistant. His current research interests include circuit-level aging and resource management for many-core architectures with a focus on thermal management.

Mark Sagi received the master of science degree from the Technical University of Munich (TUM), in 2015 and joined the chair of Integrated Systems at TUM, in 2016. His research interests are on power/thermo management of many-core systems using machine learning with a focus on representation learning algorithms. His PhD is supervised by Prof. Dr. Herkersdorf.

Anuj Pathania received the bachelor of technology (B.Tech) degree from Guru Gobind Singh Indraprastha University (GGSIPU), India, in 2009, the master of computing (MComp.) degree from the National University of Singapore (NUS), Singapore, in 2012, and the PhD degree from the Karlsruhe Institute of Technology (KIT), Germany, in 2018. His research focuses on resource management algorithms with emphasis on performance-, power- and thermal-efficiency in embedded systems. He has published papers in top peer-reviewed conferences and journals in his field of research.

Andreas Herkersdorf received the Dr degree from ETH Zurich, Switzerland, in 1991. He is a professor with the Department of Electrical and Computer Engineering and also affiliated to the Department of Informatics, Technical University of Munich (TUM). Between 1988 and 2003, he has been in technical and management positions with the IBM Research Laboratory in Rüschlikon, Switzerland. Since 2003, he leads the chair of Integrated Systems at TUM. He is a senior member of the IEEE, member of the DFG (German Research Foundation) Review Board and serves as editor for Springer and De Gruyter journals for design automation and information technology. His research interests include application-specific multi-processor architectures, IP network processing, Network on Chip and self-adaptive fault-tolerant computing.

Jörg Henkel (M'95-SM'01-F'15) received the master's and the PhD (Summa cum laude) degrees, both from the Technical University of Braunschweig, Germany. He then joined the NEC Laboratories, Princeton, NJ. He is currently with the Karlsruhe Institute of Technology (KIT), Germany, where he is directing the chair for Embedded Systems (CES). His current research interests include design and architectures for embedded systems with focus on low power and reliability. He has received the 2008 DATE Best Paper Award, the 2009 IEEE/ACM William J. Mc Calla ICCAD Best Paper Award, the CODES+ISSS 2011, 2014 and 2015 Best Paper Awards. He was the general chair of major CAD events including ICCAD and ESWeek. He is the chairman of the IEEE Computer Society, Germany Section, and was the editor-in-chief of the *ACM Transactions on Embedded Computing Systems* for two terms. He is currently the editor-in-chief of the *IEEE Design&Test Magazine*. He is also an Initiator and Spokesperson of the national priority program on Dependable Embedded Systems of the German Science Foundation and the site coordinator (Karlsruhe site) of the three-university collaborative research center on invasive computing. He is a fellow of the IEEE and holds 10 US patents.