Temperature-Aware Energy-Optimal Scheduling of Moldable Streaming Tasks onto 2D-Mesh-Based Many-Core CPUs with DVFS

Christoph Kessler[1][0000−0001−5241−0026], Jörg Keller[2][0000−0003−0303−6140], and Sebastian Litzinger[2][0000−0003−2200−7337]

[1] Linköping University, Sweden
christoph.kessler@liu.se
[2] FernUniversität in Hagen, Germany
{joerg.keller,sebastian.litzinger}@fernuni-hagen.de

Abstract. We consider the problem of energy-optimally mapping a set of moldable-parallel tasks in the steady-state pattern of a software-pipelined streaming computation onto a generic many-core CPU architecture with a 2D mesh geometry, where the execution voltage and frequency levels of the cores can be selected dynamically from a given set of discrete DVFS levels. We extend the Crown Scheduling technique for parallelizable tasks to temperature-aware scheduling, taking into account the tasks' heat generation, the heat limit for each core, and the heat diffusion along the 2D mesh geometry of typical many-core CPU architectures. Our approach introduces a systematic method for alternating task executions between disjoint "buddy" core groups in subsequent iterations of crown schedules to avoid long-time overheating of cores. We present two integer linear program (ILP) solutions with different degrees of flexibility, and show that these can be solved for realistic problem sizes with today's ILP solver technology. Experiments with several streaming task graphs derived from real-world applications show that the flexibility for the scheduler can be greatly increased by considering buddy-cores, thus finding feasible solutions in scenarios that could not be solved otherwise. We also present a fast heuristic for the same problem.

Keywords: Temperature-aware scheduling · Parallelizable tasks · Many-core CPU · Energy optimization · DVFS.

1 Introduction

Modern CMOS CPU chips are increasingly constrained by temperature. Local hot spots such as permanently highly-loaded cores can become problematic because higher temperature affects (static) power consumption negatively and because very high temperatures accelerate the aging process of the hardware (and excessively high temperatures will damage it immediately). Active cooling, even if made adaptive, will thus have to consider the hottest spot (core) on the chip at any time. Dynamic throttling by the operating system's governor is completely unaware of application requirements. Instead, application-level

temperature-aware scheduling can help to keep a high and more stable computation throughput across the chip while avoiding overheated cores.

In many cases, the problem of temperature-aware mapping a set of tasks to cores for execution is handled as a dynamic (online) optimization problem. The problem is made more complex by the fact that a high temperature of one core also affects its neighbor cores (with a certain delay) due to heat diffusion on the chip. This property can be used for passive cooling, in addition to local techniques on the hot core itself, such as core DVFS-level downscaling or by pausing running tasks. At the same time, the number of cores per chip continues to grow, also for the foreseeable future. In combination with lower clock frequencies this calls for leveraging parallelism inside tasks, using a parallel algorithm on multiple cores to utilize available resources at limited frequency.

Streaming computations refer to data-parallel computations over large collections of data where input data arrives as a stream of packets and operations are organized as a directed acyclic graph of streaming tasks connected by FIFO-buffered edges. This model, also known as Kahn process networks [13], allows for pipelined execution of dependent operations over subsequent data packets on different execution resources. We consider the steady-state pattern of such software-pipelined computation of streaming tasks. By placing instances of producer and consumer tasks operating on the same data in subsequent schedule iterations, task instances in the same iteration can be considered independent. In many soft-realtime applications, such as video processing, a specific (minimum) data rate is required, translating into a (maximum) makespan for the schedule of one iteration (or round). Hence, even with enough cores available for execution, the heaviest task might be the performance bottleneck in the computation. For such critical tasks, parallelization as well as selecting a high DVFS level (with the negative impact on heat) are options that a static scheduler can exploit to keep the throughput constraint. The schedule for the steady-state pattern should thus be constructed to achieve the throughput goal with minimum power consumption. Nevertheless, a large non-parallelizable task may have to run at the highest DVFS level, which might lead to heat overload of its assigned core.

In this paper, we consider the problem of energy-optimally mapping a set of *moldable* tasks (parallelizable tasks where the degree of parallelism is fixed prior to execution) modeling the steady state pattern of a software-pipelined streaming computation onto a generic many-core CPU architecture with a 2D mesh geometry, where the voltage and frequency levels of the cores can be selected dynamically from a given set of discrete DVFS levels, where a given upper limit for the tolerable temperature on each core must be respected, and where all tasks of one round of the steady state must be executed exactly once within a given common deadline matching e.g. a required frame processing rate.

Our approach builds atop the *Crown Scheduling* technique [17, 18] for moldable streaming tasks. Crown scheduling leverages a recursive binary partitioning of the set of p cores into a hierarchy of $2p - 1$ core groups (called the *crown*, cf. Figure 1) that become the only mapping targets for tasks, and hence restricts core allocations of tasks to powers of 2. Also, the independent tasks of one round

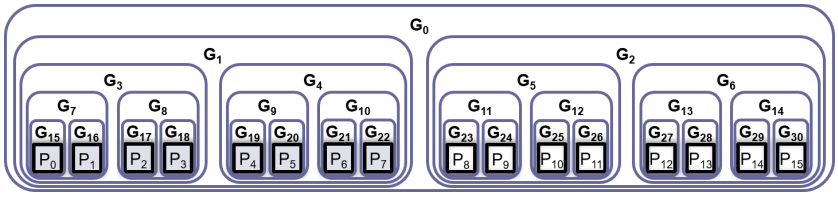

Fig. 1. A binary crown over $p = 16$ cores consists of $2p - 1 = 31$ groups $G_0, ..., G_{30}$.

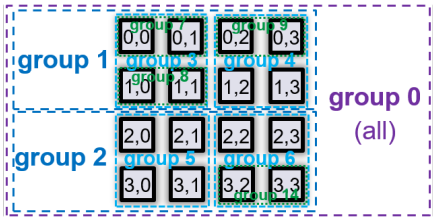

Fig. 2. 4×4 core mesh with 2D core indexing and organized as a 2D balanced binary crown structure by recursive binary space partitioning, where group 0 (root group, G_0) comprises all 16 cores, its child groups indexed 1 to 2 contain 8 cores each, down to the 16 leaf, i.e. single-core, groups.

are ordered by non-increasing core allocation, avoiding idle times between tasks on the same core and decoupling DVFS scaling decisions of any non-overlapping core groups in the crown. Earlier work has shown that restricting core allocation to powers of 2 has only negligible effect on schedule quality in practice [17, 18], but the reduction from $2^p - 1$ to $2p - 1$ mapping targets cuts down problem complexity considerably and makes even exact solutions of this complex optimization problem by integer linear programming (ILP) feasible for relevant problem sizes.

We extend the crown scheduling technique to temperature-aware scheduling, taking into account the tasks' heat generation, the heat limit for each core, and the heat diffusion along the 2D mesh geometry of typical many-core CPU architectures. The 2D layout of the crown and the resulting core index mapping is obtained by a recursive alternating binary space partitioning, as shown in Fig. 2 for a 4×4 core mesh. Hence, all crown subgroups will cover locally contiguous 2D subareas on the chip, and a group's neighbor group to the left and right is physically neighbored also in the 2D mesh network by the embedding.

For scenarios where the power and heat situation can be predicted accurately, the ILPs can compute a static schedule prior to execution, which is then applied for the complete mission time of the streaming application. For scenarios where execution time is too short to warrant static scheduling with long scheduling time, or where the heat situation changes during execution, a heuristic scheduler is presented that also applies the buddy core concept to adapt ILP-based schedules. Here, ILP-based schedules can still serve as a reference to demonstrate how close the heuristic comes to offline solutions.

This work makes the following technical contributions:

– We formalize the problem of energy-optimal, temperature-aware scheduling of multi-variant moldable streaming tasks on a 2D many-core chip geometry with DVFS, given a throughput requirement, and propose an extension of *crown scheduling* [17, 18] for this problem.

- We propose the *buddy-core technique* as a systematic method for alternating tasks between disjoint core groups in subsequent iterations of crown schedules to avoid long-time overheating of cores or core groups.
- We present two ILP-based solutions of the problem that leverage the buddy-core technique with different degrees of flexibility, and show that it can be solved for realistic problem sizes with today's ILP solver technology. Experiments with several streaming task graphs derived from real-world applications show that scheduler flexibility is greatly increased by considering buddy-cores, especially in scenarios with tight deadlines and few/no DVFS levels, thus finding feasible solutions in otherwise unsolvable scenarios.
- We also present a fast heuristic for the same problem and evaluate it with the more flexible ILP solution as a baseline.

The remainder of this paper is organized as follows: Section 2 revisits related work. Section 3 introduces our task and architecture models as well as crown scheduling. Section 4 presents the buddy-core technique. Sections 5 and 6 present ILP models with fixed and flexible buddy core selection, resp. Section 7 reports on experimental results for ILP and heuristic scheduling, and Section 8 concludes and proposes future work.

2 Related Work

Basically all existing literature of temperature-aware mapping and scheduling for multi-core architectures focuses on sequential tasks, thus disregarding the effect of optional parallelization for increasing flexibility and relaxing temperature hot spots. Moreover, a number of approaches rely on dynamic techniques alone, i.e. it is not clear how close they are to offline solutions.

Lu et al. [16] present a reinforcement learning-based approach for allocating sequential tasks at runtime to cores of a network-on-chip manycore CPU depending on current core and router temperatures in an attempt to minimize maximum temperatures in the future. DVFS is not considered in this work. Coskun et al. [6] design and evaluate OS-level dynamic scheduling policies with negligible performance overhead. They do not consider parallel tasks. Rayan and Yu [20] consider OS-level techniques for CPU throughput throttling for data centers to keep CPU temperature below a certain threshold.

Bampis et al. [2] consider approximation algorithms for static and on-line scheduling to minimize for makespan, maximum temperature or weighted average temperature. They consider non-multicore multiprocessors as target, i.e., spatial temperature diffusion between processors is not modeled. They assume unit-sized tasks and use a simplified theoretical temperature model that assumes that a processor's temperature change per unit of time equals half the difference between its previous processor temperature and the steady-state temperature of the task. They do not consider DVFS nor energy optimization.

Chantem et al. [5] present a mixed integer linear program (MILP) for static temperature-aware scheduling and mapping for hard real-time applications on

MPSoCs with DVFS. Also their work only considers sequential tasks. Where necessary, idle periods are inserted into the schedule to cool down a core before the next task can be started safely. They also present a heuristic algorithm for the same problem. But they do not consider moldable tasks and do not model a 2D many-core geometry nor spatial heat diffusion.

Jayaseelan and Mitra [12] consider temperature-aware dynamic scheduling of a mix of soft real-time and best-effort tasks for single-core embedded processors. They use an exponential function-based increase/decay model for temperature changes depending on power, with the model parameters calibrated from simulation experiments. Like our approach, they classify tasks into hot and cold tasks, based on the sign of the difference of their steady-state temperature from the maximum safe temperature at which a (hot) task still can be started. Hot and cold tasks are kept in two separate queues, and the scheduler keeps dynamically track of the share of CPU time that can be safely allocated to hot tasks. Voltage scaling is used to control the trade-off between fairness and temperature safety, by promoting down-scaled deadline-critical hot tasks to the cold queue.

The synthesis approach in Alkabani et al. [1] uses a linear programming framework that searches the best N versions of the mapping and schedule, deploys them together on the hardware platform and constructs a thermal-aware rotational schedule that switches between these in order to balance the thermal profile of the processor. Their experiments show a very low overhead and an average 5% decrease in the steady-state peak temperature produced on the benchmark designs compared to using a schedule that balances the amount of usage of different modules.

Bao et al. [3, 4] consider temperature-aware dynamic voltage scaling in combination with mapping and scheduling of single-threaded tasks and of idle time intervals on a sequential CPU resp. on a multiprocessor to minimize the temperature-dependent share of energy (i.e., leakage energy). The technique leverages the non-linearity of the temperature impact on leakage energy e.g. by equally spreading out idle time in a schedule between tasks to maximize their cooling effect, instead of closing up idle times to a single idle interval. Parallel tasks or temperature diffusion in multicore architectures are not considered.

Pierson et al. [19] present mixed ILPs for optimizing energy consumption or makespan of sets of jobs in a datacenter with heat constraints, taking into account heat diffusion. Their jobs are only executed once, while we target a long sequence of similar execution rounds for a streaming application.

3 Architecture and Application Model

All symbols and parameters introduced in this and the following sections are summarized in Table 1.

3.1 Generic Multi-/Many-Core Architecture with DVFS

We consider a generic multi-/many-core architecture with 2D mesh layout of p cores on the chip as introduced in Section 1. Each core can select its DVFS

Table 1. Notation summary

symbol	meaning
n	number of tasks
n_j	number of task j's variants
$w_{j,v}$	minimum parallelism degree for task j's variant v
$W_{j,v}$	maximum width of task j's variant v
$\text{eff}_{j,v}(q)$	parallel efficiency for task j's variant v when run on q cores
p	number of cores
s	number of discrete core operating frequency levels
f_k	core operating frequency on frequency level k
f_{min}	minimum core operating frequency
f_{max}	maximum core operating frequency
G_l	set of groups core l belongs to
M	length of execution round
$\text{work}(j, v)$	workload of task j's variant v
$Pow(j, v, k)$	core power consumption when running task j's variant v at frequency f_k
$\text{time}(q, j, v, k)$	runtime of task j's variant v on q cores at frequency f_k
$\text{E}(q, j, v, k)$	energy consumption for execution of task j's variant v on q cores at frequency f_k
$x_{i,j,v,k}$	binary ILP decision variable, =1 iff task j's variant v is run in core group i at frequency f_k
$\text{size}(i)$	number of cores in core group i
$\text{heat}(l) = \text{heat}(u, w)$	core-local heat load of core l, i.e. core at position u, w in 2D grid
$\text{Heat}(l) = \text{Heat}(u, w)$	overall heat load of core l, i.e. core at position u, w in 2D grid
α	weight for net influx/outflux of heat to/from direct neighbor cores
β	weight for net in-/outflux of heat to/from diagonal neighbor cores
maxHeat	core heat limit
T_0	room temperature
$P_{diff}(T)$	additional power consumption for core at temperature T
hot_l	binary ILP decision variable, =1 iff core l is classified as hot core
$sHeat(l)$	scaled overall heat load of core l
C_{hot}	hotness threshold
$MinHeat$	minimum value of $\text{Heat}(l)$
$MaxHeat$	maximum value of $\text{Heat}(l)$
γ_j	reduction of buddy core group idle period, as fraction of buddied task j's duration
$y_{i,j,k}$	binary ILP decision variable, =1 iff task j runs in variant 0 in core group i at frequency f_k with buddying enabled
$\text{heatodd}(l)$	continuous ILP decision variable, core-local heat load of core l in odd rounds
$\text{heateven}(l)$	continuous ILP decision variable, core-local heat load of core l in even rounds
$\text{Heatodd}(l)$	overall heat load of core l in odd rounds
$\text{Heateven}(l)$	overall heat load of core l in even rounds
$\text{penalty}(i, j)$	continuous ILP decision variable, penalty for execution of task j's variant 0 in core group i in the first round and subsequent execution of its variant V in a different core group in the second round
$F(i)$	set of all groups sharing a core with group i
$\text{misscost}(i, i')$	$\text{penalty}(i, j)$ for second-round group i'
ϵ	weight for penalty term in objective function
LARGECONST	a large constant
d	deadline tightness factor
λ_j	$\text{work}(j, 0)$, i.e. workload of task j's variant 0

level dynamically (namely, between any two tasks) from a given set of s discrete frequencies $\{f_1 = f_{min}, ..., f_s = f_{max}\}$ in ascending order, applying voltage co-scaling to always use the lowest voltage still applicable for a chosen frequency. We make no assumptions about the power/energy cost function and its dependence on the DVFS level; it could be measured by microbenchmarking on the target system as in [9] or derived from a theoretical model, and becomes a set of constant model parameters to our optimization, with power values indexed by frequency level, temperature, and possibly instruction mix, i.e. task index.

One can simplify the table of power values by partitioning temperatures into a small number of temperature ranges, and classifying instruction mixes into task type categories. We will do especially the former by categorizing a core as hot or cold. Following [3], the difference in power consumption between hot and cold core is mostly due to static power difference and thus can be simplified to a core type-specific constant.

A core that draws power produces heat, which influences its temperature and the temperature of the nearby cores via heat diffusion. Thus, the temperatures of a set of cores with given workloads can be derived by solving differential equations. As a simplification, we will compute the heat flow between cores by a discretized and linearized set of equations, to include this into the ILP.

3.2 Multi-variant Moldable Streaming Tasks

We consider a streaming task graph with n nodes or tasks as introduced in Section 1. We assume *multi-variant tasks* where each task j (i.e., a node in the streaming task graph) can have $n_j \geq 1$ different variants $(j, 0), ..., (j, n_j - 1)$ that might internally differ e.g. in the algorithm used, in the compiler options or in settings of tunable parameters such as tile sizes.

Some of the task variants (j, v) of task j might be *inherently sequential* (modeled by setting its maximum parallelism parameter $W_{j,v} = 1$), *fully moldable* ($W_{j,v} = \infty$) or *partially moldable* ($W_{j,v}$ is given as some fixed maximum number of cores). We also allow to define a *minimum parallelism degree* $w_{j,v}$ for each task variant, which will be 1 in most cases but might be > 1 in special scenarios, which we will exploit later. Moldable task variants (j, v) (i.e., $W_{j,v} > 1$) internally use a parallel algorithm that executes simultaneously on all cores assigned to (j, v), and have an efficiency parameter table $\text{eff}_{j,v}(q)$ that models its scalability to q cores, with $1 \leq q \leq W_{j,v}$. The $\text{eff}_{j,v}$ values can either be provided by a performance model of the parallel algorithm within (j, v), or by measuring the execution time of (j, v) for the different applicable values of q on the target[3].

As an example, consider the binary stream merge tasks in parallel stream mergesort where each task instance works on a pair of sorted input buffers to merge. The binary merge functionality is usually implemented by the well-known sequential merge algorithm, which performs work linear in the size of the input.

[3] We assume that tasks are computation-bound, i.e. that task runtime is inverse to core frequency, so that decisions on resource allocation and frequency scaling can be separated. Extensions to memory-bound or communication-bound tasks are possible.

Also parallel algorithms for merging are known [11], e.g. based on binary search, which performs work $O(N \log N)$ for input buffer size N and also has higher constant factors. As long as the root merger task (which is the performance bottleneck in the merger tree pipeline because it has the highest data throughput) still can be accommodated using DVFS scaling, the sequential merge variant will usually be the most energy-efficient one. However, if the root merger can no longer make the requested throughput rate but enough idle cores are available, even an inefficient parallel variant might (have to) be preferred for it.

Each task variant (j, v) has a certain instruction mix that influences the core power consumption for any given frequency level. To reduce complexity in ILPs, one might combine similar power profiles and thus create task types.

3.3 Scheduling for the Steady State

As introduced in Section 1, to run a streaming task graph with a throughput of X packets per second, every $M = 1/X$ seconds one instance of the streaming task graph must complete. Hence, each task must execute once within a round of length M. As the tasks belong to different graph instances, they can be considered independent. The tasks communicate via the on-chip network, but as the communication is from one round to the next, we assume that the on-chip network's delay is low enough and its capacity large enough to achieve this, and leave modelling of the tasks' communications on the on-chip network (and the resulting energy consumption) as a subject of future work.

For each task, we must determine in which variant, with which possible width and which frequency it will be executed. As the tasks are known beforehand, this can be done prior to execution as static scheduling, which also arranges the tasks such that a core never must execute two tasks simultaneously (non-preemptive scheduling) and that all tasks complete before the deadline. Finally, heat constraints have to be obeyed. Among all feasible schedules for such a situation, we are interested in one with minimum energy consumption per round.

Yet, given all these constraints, there might not be any feasible schedule. For such situations, there is the possibility of using a repeating sequence of differing schedules [1], so that the sequence obeys all constraints (in particular the heat constraints), while the sum of the schedules' energy consumptions is minimized. To achieve this, we will build on previous work.

Crown scheduling [17, 18] is a technique for resource allocation, mapping and DVFS selection for a given set of independent moldable tasks, which is based on a recursive partitioning of the set of p cores into up to $2p-1$ core groups jointly referred to as the *crown* (see Figure 1), and requests that tasks are allocated and mapped to entire such groups only, thus cutting down the complexity of possible mapping targets of each task from up to 2^p-1 in the unrestricted case to $2p-1$. It also requests that in a schedule, tasks are ordered by decreasing core allocation, so that a parallelized task always starts simultaneously on the different cores it is mapped to. Scheduling decisions are done such that all tasks complete before the deadline and that cores' energy consumption is minimized. Extensions of

the crown scheduling technique for architectures with DVFS islands and with multiple core types have been described in [14].

4 Temperature-Aware Crown Scheduling with Buddying

Some tasks may be considered by the scheduler as inherently hot yet they constitute a performance bottleneck for the steady-state schedule, i.e., they might be sequential and cannot be shortened by parallel processing, and they need to run at the highest DVFS level to make the deadline constraint. For these tasks, the only way to avoid thermal runaway on the cores they are mapped to is to spread out execution in consecutive rounds over multiple cores in an alternating fashion so that cores used in the previous round can cool down again during idle periods in the following round. This alternation however requires that we reserve sufficient capacity on a number of spare cores for these tasks. In general, it would be wasteful to do a conservative a-priori reservation, so the decision about such reservations needs to be made together with the other scheduling decisions (i.e., core allocation, mapping, and DVFS level selection). We also refrain from a complex multi-round schedule with a cyclic migration scheme for all tasks because this causes additional cache misses and overhead for many tasks, and it also would increase scheduling complexity considerably. Instead, we use one reserve core per core running a hot task, i.e., hot tasks lead to a duplication of core allocation. This choice is only for simplification of presentation and can be generalized to more than two-way alternation (preferably, by powers of 2).

The set of cores is first organized in pairs of physically neighbored cores ("buddies"); w.l.o.g. let us assume these are pairs of neighbored odd and even-indexed cores, only one of which will be used for a hot task at any time, and the runtime system can toggle between a core and its "buddy core" for the execution of a hot task between any two subsequent (hot) tasks whenever a core gets too warm. Note that the "idle" buddy core is not necessarily idle for a complete round but still runs at least all cool tasks that have been mapped to it, while it will simply skip the instances of hot tasks that will be executed by its buddy core instead. Because the buddies are physically neighbored and thus share most cache levels, the penalty of switching between buddy cores for a hot task due to additional cold misses in core-local cache levels will be relatively low. This buddy-core concept can be generalized towards (non-root) groups in the crown.

Hence, in a crown schedule, any hot task must now be mapped *twice*, both to a core group and to the group of their buddy cores, and hence its real core allocation will be twice as large as for the ordinary cool tasks. Note that for hot parallel tasks this applies to all cores it uses, i.e., the effective core allocation of a hot task is twice its nominal allocation. In contrast, cool tasks will only receive their nominal core allocation and be mapped once, as before.

We model this by defining a second task variant $(j, 1)$ for each task j. The original version of task j now becomes variant $(j, 0)$. There might be more variants of a task, e.g. using a different internal algorithm, but we will not consider them in particular in the sequel. The new variant $(j, 1)$ internally em-

ploys temperature-dependent alternation among buddy cores as described above, it thus has the double values for minimum and maximum malleability[4], i.e., $w_{j,1} = 2w_{j,0} \geq 2$ and $W_{j,1} = 2W_{j,0}$, and its efficiency parameters can be derived accordingly as $\text{eff}_{j,1}(2q) = \text{eff}_{j,0}(q)/2$ for any $q \geq 1$, which will affect time and E (energy) parameters accordingly, see Sect. 5.1. Thus, a task in variant 1 can simply be mapped by the crown scheduler to a group, and the runtime system alternately uses one half of the group's cores or the other in two succeedings rounds. Variant $(j, 1)$ is to be selected by the scheduler instead of $(j, 0)$ if variant $(j, 0)$ with selected parallelism and DVFS level is considered too hot.

For greater flexibility we will allow also cold tasks to use the alternating variant and hot tasks to use the non-alternating variant, and leave the choice to a global optimizer, which we describe next.

5 ILP Model with Fixed Buddy Cores

5.1 Time and Energy

For a task variant (j, v) running on a core group with q cores at DVFS level k, let $\text{time}(q, j, v, k)$ denote the time and $\text{E}(q, j, v, k)$ denote the overall energy used by the task variant. These model parameter values can either be predicted by a model or obtained by measuring energy for a microbenchmark modeling the task type of task variant (j, v). For example, if sampled or predicted parallel efficiency values for the parallel algorithm used in a moldable task are available, the parallel execution time for the task can be modeled by

$$\text{time}(q, j, v, k) = \text{work}(j, v) / \left(q \cdot \text{eff}_{j,v}(q) \cdot f_k\right). \tag{1}$$

If $Pow(j, v, k)$ is the (average) power consumption of a core when running task variant (j, v) at frequency level k (the dependence on temperature will be captured in Section 5.5), then the energy consumption of this task can be modeled by

$$\text{E}(q, j, v, k) = q \cdot \text{time}(q, j, v, k) \cdot Pow(j, v, k)\ . \tag{2}$$

5.2 ILP Solution Variables

Our ILP model uses binary variables $x_{i,j,v,k} = 1$ iff variant v of task j is selected and mapped to group i at DVFS level k.

5.3 Constraints

Allocation, Mapping and Scaling constraint Exactly one variant of each task j must be selected and be mapped to exactly one core group and exactly one DVFS level:

$$\forall j: \quad \sum_{v,i,k} x_{i,j,v,k} = 1 \tag{3}$$

[4] Core allocations must be multiples of 2, which is automatically preserved by crown scheduling if $w_{j,v} \geq 2$.

Also, a task variant (j, v) must at least receive its minimum core allocation $w_{j,v}$ and not exceed its maximum core allocation $W_{j,v}$:

$$\forall j, v, k: \ \forall i \text{ where } \text{size}(i) < w_{j,v}: \quad x_{i,j,v,k} = 0 \tag{4}$$

$$\forall j, v, k: \ \forall i \text{ where } \text{size}(i) > W_{j,v}: \quad x_{i,j,v,k} = 0 \tag{5}$$

Here $\text{size}(i)$ denotes the number of cores in core group i. Note that we do *not* add a constraint that forbids hot tasks to use the first (i.e., regular, non-alternating) variant. This also eliminates the need to strictly define some criterion separating hot and cold tasks from each other. In this way we leave it to the optimizer alone to decide by its global view of the problem for which tasks the regular or the alternating variant is more suitable.[5]

In general, the alternating variant of a task will be less energy-efficient than the regular variant because it yields only half the performance from the used resources at otherwise same settings (e.g., same DVFS level), and it will thus only be selected by the optimizer where really necessary to meet the throughput and temperature constraints below.

Throughput constraint Each core must complete its assigned tasks within the given deadline M:

$$\forall l: \quad \sum_{i \in G_l} \sum_{j,v,k} x_{i,j,v,k} \cdot \text{time}(\text{size}(i), j, v, k) \leq M, \tag{6}$$

where G_l denotes the set of groups core l belongs to.

Temperature constraint The *core-local heat load* $\text{heat}(l) = \text{heat}(u, w)$ of core $l = (u, w)$ is the sum over the energies of its assigned task variants, divided by their accumulated time (in other words, it is the average power of core l over one round):

$$\text{heat}(l) = \left(\sum_{i \in G_l} \sum_{j,v,k} x_{i,j,v,k} \cdot \text{E}(\text{size}(i), j, v, k)/\text{size}(i) \right) /M. \tag{7}$$

The *overall heat load* $\text{Heat}(l) = \text{Heat}(u, w)$ of core $l = (u, w)$ is given by its local heat load plus the weighted heat differences to its direct neighbor core heat loads, which models nearest-neighbor heat diffusion:

$$\begin{aligned} \text{Heat}(u,w) = {} & \text{heat}(u,w) \cdot (1 - 4(\alpha + \beta)) \\ & + \alpha \left(\text{heat}(u{-}1,w) + \text{heat}(u,w{-}1) + \text{heat}(u,w{+}1) + \text{heat}(u{+}1,w) \right) \\ & + \beta \left(\text{heat}(u{-}1,w{-}1) + \text{heat}(u{-}1,w{+}1) + \text{heat}(u{+}1,w{-}1) \right. \\ & \quad \left. + \text{heat}(u{+}1,w{+}1) \right) \end{aligned} \tag{8}$$

[5] In (rather unlikely) borderline cases it could actually make sense to run even a cold task in alternating mode to make it even colder, so that it can compensate for the heat impact of a hot task on the same core(s) without having to use the more inefficient alternating variant for that hot task.

where $0 \leq \beta < \alpha \ll 1$ are the weights that model the net influx/outflux of heat to/from the core's up to four corner and up to four direct neighbor cores, respectively. For cores located on the chip boundary, i.e., $u = 0$ or $u = q - 1$ or $w = 0$ or $w = q - 1$, the above equation is adapted accordingly as not all neighbor cores exist; for these non-existing neighbors the temperature difference is assumed to be 0. We also note that $\alpha + \beta < 0.25$ because Heat values cannot be negative. Note that, for simplicity, we do not model second-order effects in heat diffusion, i.e., other cores more than one position away in a dimension do not contribute to a core's overall heat.

We require that the overall heat load of a core does not exceed its heat limit:

$$\forall l: \quad \text{Heat}(l) \leq \text{maxHeat} \tag{9}$$

5.4 Objective Function

We minimize the overall energy for the steady state, which is calculated as

$$E = \sum_{i,j,v,k} x_{i,j,v,k} \cdot \text{E}(\text{size}(i), j, v, k) \tag{10}$$

By running the ILP for different settings of the maxHeat parameter, we can explore several thermal design options for a fixed-application embedded system. For instance, a higher maxHeat value could be permitted if employing active air cooling, which however also costs some energy overhead to drive the fan.

5.5 Temperature-Dependent Power Modeling

So far, the heat flow information has only been used to forbid situations where cores may overheat. However, the core temperature also influences the core's power consumption. We use the *Heat* variables to guess the cores' temperatures and classify each core as cold or hot. This is a simplification because the heat flow and thus the temperature may vary over a round, but as the *Heat* variables only provide information summarized over a round, we cannot have more fine-granular information on this basis. The use of only two categories hot and cold is a further simplification, but can be extended to more categories.

According to [3], the temperature mostly influences static power consumption. Thus, the power profile of a core, i.e. power consumption for each operating frequency for a given instruction mix and a core temperature equaling room temperature T_0, is shifted by a temperature-dependent constant $P_{diff}(T)$ when a higher core temperature $T > T_0$ is assumed. Hence, to apply the knowledge about core temperature within the above simplifications, the additional energy for each hot core amounts to $P_{diff}(T_{hot}) \cdot M$.

Thus, we introduce p binary variables hot_l where $hot_l = 1$ iff core l is hot, and adapt the target function by adding a term $\sum_l hot_l \cdot P_{diff}(T_{hot}) \cdot M$.

To set these binary variables, we scale the Heat variables to interval $[0; 1]$ denoted by *sHeat*(see below), and define a threshold $C_{hot} \in [0; 1]$ that distinguishes

between hot and cold cores via

$$hot_l \geq sHeat(l) - C_{hot} \tag{11}$$
$$hot_l < 1 + sHeat(l) - C_{hot} \tag{12}$$

Constraint (11) forces hot_l to 1 if $sHeat(l) > C_{hot}$, constraint (12) forces hot_l to 0 if $sHeat(l) \leq C_{hot}$.

After deriving bounds *MinHeat* and *MaxHeat* for the scope of variables *Heat*,

$$sHeat(l) = \frac{Heat(l) - MinHeat}{MaxHeat - MinHeat} .$$

6 ILP Model with Arbitrary Buddies

In the ILP model of Section 5 we assumed that a fixed, nearest-neighbor core or core group is predefined as the buddy core where buddying is necessary to avoid long-term overheating of a core or core group. This implies that the buddy core group always has the same width as the overheated core group. This restriction can however be relaxed to increase the flexibility for the scheduler: the buddy group could be any group, also narrower or wider than the overheated core group to be offloaded, as long as these two groups do not intersect. In particular, we do no longer request that the buddy cores are direct neighbors, i.e. in one common group in the crown structure. Where a more distant group is selected as buddy group for a task, we charge a certain time penalty for the expected increase in cache misses and/or data movement costs compared to nearest-neighbor buddy selection. Another requirement that can be relaxed is that the idle phase on the buddy core group needs to be equally long as the buddied task—the idle phase could also be shorter if that is already sufficient for cooling down the core.

We present a generalized ILP model that allows such arbitrary buddy selection. To do so, we extend the scope of optimization to two subsequent rounds of the steady-state pattern, where we allow the idle period on the buddy group to be shortened compared to the buddied task's duration by up to a factor $0 < \gamma < 1$ in one round.

We use the following decision variables:

- $x_{i,j,v,k} = 1$ iff task j is executed in variant $v \in \{0, \ldots, V\}$ in core group i at frequency level k. Variant 0 is the default variant, variant V is the buddy core variant which is only available for the default variant. Variants 1 to $V - 1$ (if available) are alternative variants for which no buddy variant is necessary.
- $y_{i,j,k} = 1$ iff task j is executed in variant 0 in core group i at frequency level k *and* buddy cores are employed.
- heatodd(l) is the local heat load for core l in odd rounds.
- heateven(l) is the local heat load for core l in even rounds.

– penalty(i, j) is the penalty for executing task j in the first round in variant 0 in core group i, when task j is executed in variant V, i.e. as a buddy task, in the second round, in a different group i'.

The objective function (13a) is the sum of the energy consumption caused by execution of tasks in two subsequent rounds and penalties for mapping tasks and their buddy tasks to distant buddy cores. In the first (odd) round, buddy variants V of tasks are not executed. In the second (even) round, variant 0 of a task is only executed if the buddy variant V is not chosen. If V is chosen ($y_{i,j,k} = 1$), we therefore subtract energy for variant 0.

The optimization problem then reads as follows:

$$\min \quad \sum_{i,j,v<V,k} x_{i,j,v,k} \cdot \mathrm{E}(\mathrm{size}(i), j, v, k) + \sum_{i,j,v,k} x_{i,j,v,k} \cdot \mathrm{E}(\mathrm{size}(i), j, v, k) - \sum_{i,j,k} y_{i,j,k} \cdot \mathrm{E}(\mathrm{size}(i), j, 0, k) + \epsilon \cdot \sum_{i,j} \mathrm{penalty}(i, j) \tag{13a}$$

$$\text{s.t.} \quad \forall j \quad \sum_{i,v<V,k} x_{i,j,v,k} = 1, \tag{13b}$$

$$\forall j \quad \sum_{i,k} x_{i,j,V,k} \leq \sum_{i,k} x_{i,j,0,k}, \tag{13c}$$

$$\forall i, j \quad 1 - \sum_{k} x_{i,j,0,k} \geq \sum_{i' \in F(i),k} x_{i',j,V,k}, \tag{13d}$$

$$\forall j, v \quad \sum_{i:\mathrm{size}(i) > W_{j,v},k} x_{i,j,v,k} = 0, \tag{13e}$$

$$\forall i, j, k \quad y_{i,j,k} \geq x_{i,j,0,k} + \sum_{i',k'} x_{i',j,V,k'} - 1, \tag{13f}$$

$$\forall i, j, k \quad y_{i,j,k} \leq x_{i,j,0,k}, \tag{13g}$$

$$\forall i, j, k \quad y_{i,j,k} \leq \sum_{i',k'} x_{i',j,V,k'}, \tag{13h}$$

$$\forall l \quad \sum_{i \in G_l} \left(\sum_{j,v<V,k} x_{i,j,v,k} \cdot \mathrm{time}(\mathrm{size}(i), j, v, k) + \sum_{j,k} x_{i,j,V,k} \cdot \mathrm{time}(\mathrm{size}(i), j, V, k) \cdot \gamma_j \right) \leq M, \tag{13i}$$

$$\forall l \quad \sum_{i \in G_l} \left(\sum_{j,v,k} x_{i,j,v,k} \cdot \mathrm{time}(\mathrm{size}(i), j, v, k) - \sum_{j,k} y_{i,j,0,k} \cdot \mathrm{time}(\mathrm{size}(i), j, 0, k) \cdot (1 - \gamma_j) \right) \leq M, \tag{13j}$$

$$\forall i,j \quad \text{penalty}(i,j) \geq \left(\sum_k x_{i,j,0,k} - 1\right) \cdot \text{LARGECONST} + \sum_{i',k} x_{i',j,V,k} \cdot \text{misscost}(i,i'), \tag{13k}$$

$$\forall i,j \quad \text{penalty}(i,j) \geq 0, \tag{13l}$$

$$\forall i,j \quad \text{penalty}(i,j) \leq \sum_{i',k} x_{i',j,V,k} \cdot \text{misscost}(i,i'), \tag{13m}$$

$$\forall l \quad \left(\sum_{i \in G_l,j,v<V,k} \text{E}(\text{size}(i),j,v,k)/\text{size}(i)\right)/M = \text{heatodd}(l), \tag{13n}$$

$$\forall l \quad \left(\sum_{i \in G_l,j,v,k} \text{E}(\text{size}(i),j,v,k)/\text{size}(i) - \sum_{i:l \in i,j,k} y_{i,j,k} \cdot \text{E}(\text{size}(i),j,0,k)/\text{size}(i)\right)/M = \text{heateven}(l), \tag{13o}$$

$$\forall l \quad (\text{Heatodd}(l) + \text{Heateven}(l))/2 \leq \text{maxHeat}. \tag{13p}$$

We apply the following constraints:

- general constraints:
  - (13b)**:** Exactly one variant (either the default or an alternative one) must be chosen for each task, and the task shall be mapped to one core group at one frequency level (except for buddy core variant, see below).
- buddy constraints:
  - (13c)**:** The buddy core variant may be used only in conjunction with the default variant 0.
  - (13d)**:** The buddy variant must not share cores with the standard variant (i.e. variant 0). Here, $F(i)$ denotes the set of all groups that share a core with group i.
- width constraints:
  - (13e)**:** We do not allocate a number of cores greater than the maximum width of a task.
- throughput constraints:
  - (13f)**,** (13g)**,** (13h)**:** $y = 1$ iff variant 0 is selected and buddying is enabled, i.e. variant V is also selected.
  - (13i)**:** The sum of task runtimes per core must not exceed the deadline in odd rounds, i.e. for execution of the standard variant, where applicable.
  - (13j)**:** The sum of task runtimes per core must not exceed the deadline in even rounds, i.e. for execution of buddy core variant, where applicable.
- constraints for penalty terms:
  - (13k)**:** If variant 0 is executed in group i and the buddy variant in group i', the penalty is misscost(i,i'), which is a constant that can be derived

from measurements or microbenchmarks[6]. The large constant serves to neutralize the constraint if variant 0 is not chosen.

(13l): The penalty cannot be negative.

(13m): If no buddy variant is planned for the execution of task j, there is no penalty.

– thermal constraints:

(13n), (13o): The core-local heat load is the average power consumption of a core in odd and even rounds, respectively.

(13p): The average overall core heat over odd and even rounds may not exceed the maximum heat threshold. The definitions of Heatodd(l) and Heateven(l) are in analogy to (8), and omitted for the sake of brevity.

7 Evaluation

In order to demonstrate the applicability of the proposed approach, we have conducted experiments with two applications: parallel mergesort and H.263 encode. The mergesort application consists of 15 tasks forming a tree-shaped task graph as shown in Figure 3 (left). Moving towards the root, the workload is doubled at each level. We assume that the individual tasks are executed sequentially. The H.263 encode application originates from the Dataflow Benchmark Suite (DF-bench) [7]. It comprises 9 tasks, the task graph is depicted in Figure 3 (right). Some edges related to control flow had to be pruned to obtain an acyclic graph. Again, a task itself is assumed to run sequentially.

By experiment, we mean that we compute a schedule by solving an ILP, and consider the energy from the objective function the result of the experiment. Previous experiments on real platforms have demonstrated that the ILP model predicts energy consumption on real machines with sufficient accuracy [8, 9]. Thus, we do not perform a system simulation.

For our experiments, the deadline computation is inspired by [18]:

$$M = d \cdot \frac{\sum_j \frac{\lambda_j}{p \cdot f_{max}} + \sum_j \frac{\lambda_j}{p \cdot f_{min}}}{2},$$

where d can be set to various values and thus enables control over deadline tightness, and where $\lambda_j = \text{work}(j, 0)$. The second parameter we vary for the experiments is maxHeat, i.e. the per-core heat limit. For the two task sets in question, we have examined all combinations of d and maxHeat for $d \in \{2.2, 2.5, 3.0, 3.5, 4.0\}$ and maxHeat $\in \{0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$, which amounts to 45 distinct parameter settings in total. Decreasing d tightens the deadline and serves to find the point where normal Crown scheduling is not able to find a feasible schedule while Crown scheduling with buddy cores still is. Similarly, decreasing maxHeat increases sensitivity to overheating, and thus helps to

[6] If the penalty might differ between tasks, then the misscost table could be further indexed by the task index.

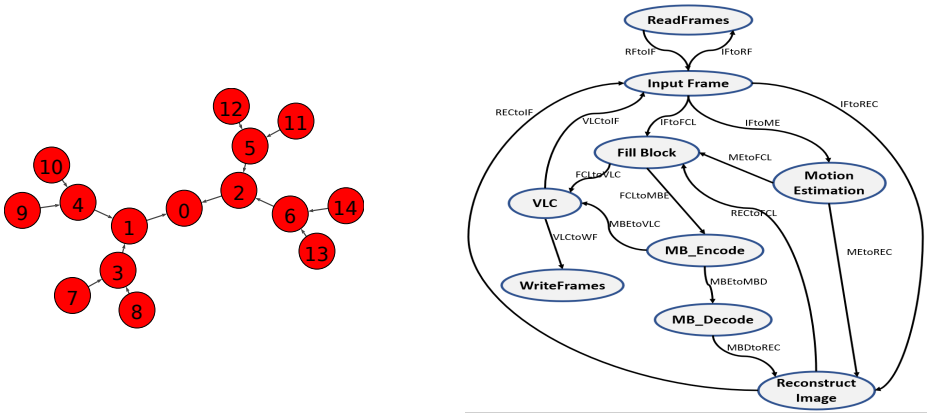

Fig. 3. Left: task graph for the mergesort application. Right: task graph for the H.263 encode application.

find the point where the cores get too hot to find a feasible schedule with using buddy cores.

For each task set, we compare scheduling with buddying to regular crown scheduling under heat constraints. For the latter case, we add constraints $\forall j : \sum_{i,k} x_{i,j,1,k} = 0$ to prohibit the use of buddy cores.

Furthermore, we have performed experiments for two different ways of modeling thermal behavior, which is represented by the values of α and β. We set $\alpha = \beta = 0$ to consider core-local heat only, whereas $\alpha = 0.1$, $\beta = 0.05$ models heat diffusion to neighboring cores. For all experiments, we assume $p = 16$ and an architecture which exhibits the power characteristics of the ARM big.LITTLE, cf. [9]. As we base our experiments on a homogeneous architecture, we adopt the values for the big cores only. We have implemented the ILP in Section 5 in Python with the `gurobipy` package and subsequently employed the Gurobi 8.1.0 solver. All scheduling computations were performed on an AMD Ryzen 7 2700X with 8 cores and SMT.

Our primary interest lies in determining whether the buddy core concept enables us to retrieve feasible solutions where otherwise deadlines are too tight or heat constraints too restrictive to produce a feasible schedule. The results for the mergesort task set are displayed in Figure 4. On the left hand side, only core-local heat is considered, while the results on the right hand side feature heat diffusion to neighboring cores. The plots show for which value combinations of d and maxHeat a feasible solution exists. If feasible schedules exist with and without buddying, both have the same energy consumption. It becomes clear immediately that a significantly wider range of parameter combinations can be covered with the buddy core concept. When looking at core-local heat only, making use of buddy cores leads to successful accommodation of tighter maxHeat constraints for each value of d. In the scenario where heat diffusion to neighboring cores is assumed, there is no need for buddy core execution at longer deadlines

for the examined values of maxHeat. This does not come as a surprise since heat diffusion mitigates heat spikes at individual cores, especially when the heat load is distributed heterogeneously among the cores as is the case here due to the number of cores exceeding the number of tasks. A similar behavior can be observed for the H.263 encode task set, cf. Figure 5, although enabling buddy cores pays off more substantially even in the heat diffusion case, where feasible solutions for more restrictive heat constraints can be found for all but the tightest and the most loose deadlines.

Figure 6 shows corresponding results for an architecture without DVFS, i.e., all tasks run at the same (high) frequency (here, 1.6 GHz). Deadlines and the values for maxHeat correspond to the original experiments in Figures 4 and 5, respectively. By comparing the diagrams, it is easy to see that the benefit of using buddy cores is strongest in this scenario with no DVFS.

Figure 7 shows that with heat diffusion included in the model (which reduces heat stress on cores due to the cooling effect of their direct neighbors) and flexible buddy selection (cf. Section 6) enabled, feasibility is still increased by buddying.

There are cases (such as H.263 encode for deadline factor 2.5 and maxHeat 0.3 or mergesort for deadline factor 2.2 and maxHeat 0.3 or deadline factor 2.5 and maxHeat 0.2) where flexible selection of buddy cores enables a feasible solution when none exists with fixed buddy selection only. In these cases the penalty is accepted in order to select non-neighbored buddy cores and thereby be able to keep the maxHeat constraint.

In an additional set of experiments, we have sought to determine the influence of γ. To this end, we have set $\gamma \in \{0.1, 0.5, 1.0\}$ for both the H.263 encode and the mergesort task set. The smaller γ gets, the more chances the scheduler has to place tasks on the idle buddy core, but this also reduces the time for this core to cool down. The values chosen represent both ends of the spectrum plus a value in-between. Experiments were performed for the same parameter combinations of deadline factor and maxHeat as for the earlier experiments. For both task sets, no difference in the number of feasible solutions could be established. Presumably, this is due to large sequential tasks. The only option is to increase frequency to the maximum. If the deadline is still violated, the value of γ is irrelevant as no other task variants would be mapped to those cores.

In another experiment, we stressed the temperature-aware ILP crown scheduler with a set of $n = 8$ sequential tasks and tight deadline and a 4×4 core CPU with $\alpha = 0.05$, $\beta = 0$, penalty $= 0$. With buddying disabled and the heat limit chosen so that it is just feasible, we obtain a checkerboard mapping, which is expected as it maximizes the self-cooling effect due to unused cores. For a buddying-enabled scenario with $n = 4$ the ILP crown scheduler of Sect. 6 yields an alternating checkerboard schedule with alternation between different white fields in odd and even rounds if tightening the heat limit even more.

As an alternative to the ILP-based approach, we have devised a simple *heuristic algorithm* applying the buddy core concept to a temperature-unaware schedule which turns out infeasible when considering temperature constraints ex-post. The heuristic starts from a schedule generated by converting a scheduler for mal-

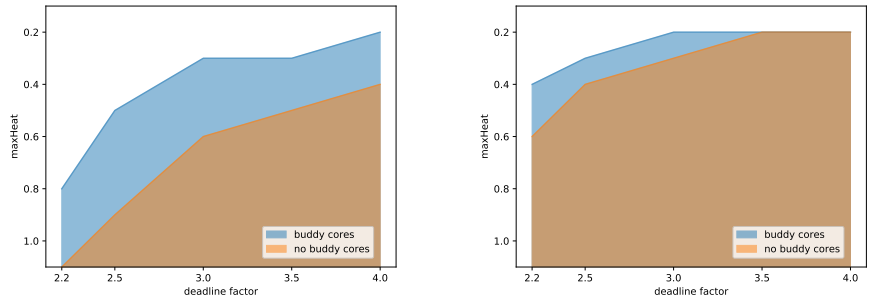

Fig. 4. Value combinations of d and maxHeat for which a feasible solution exists for the mergesort task set. Left: core-local heat only, right: heat diffusion to neighboring cores.

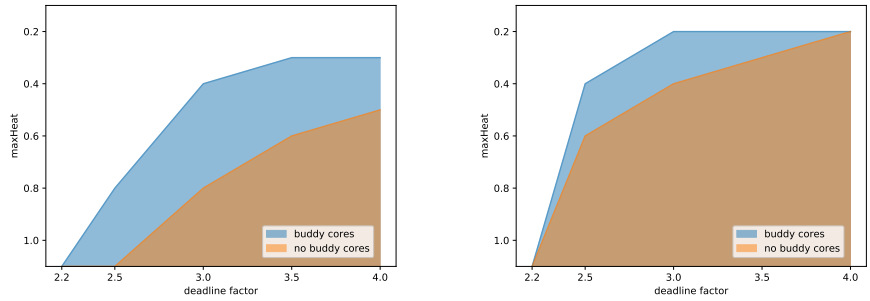

Fig. 5. Value combinations of d and maxHeat for which a feasible solution exists for the H.263 encode task set. Left: core-local heat only, right: heat diffusion to neighboring cores.

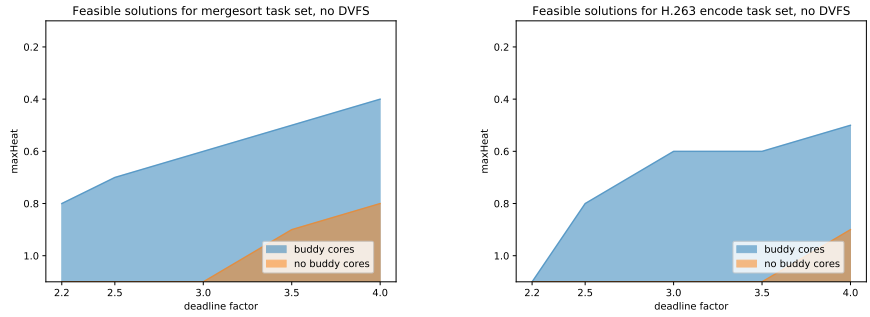

Fig. 6. Value combinations of d and maxHeat for which a feasible solution exists for the mergesort task set (left) and for the H.263 encode task set (right), respectively, on an architecture with no DVFS (i.e., all tasks run at the same frequency, here 1.6GHz).

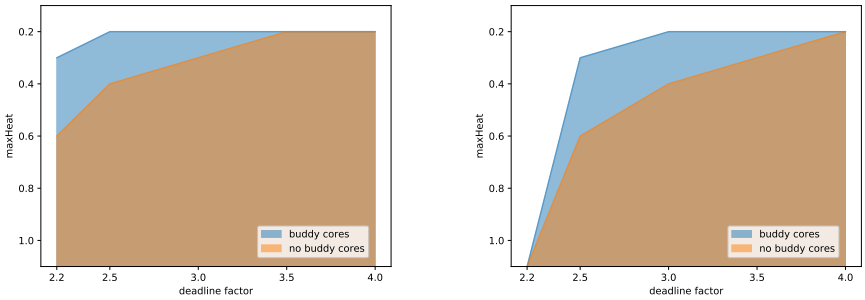

Fig. 7. Value combinations of d and maxHeat for which a feasible solution exists with heat diffusion modeled and flexible buddy selection (Sect. 6) enabled for the mergesort task set (left) and for the H.263 encode task set (right), respectively, with DVFS enabled. Other settings: $\gamma = 1.0$, $\alpha = 0.1$, $\beta = 0.05$, Heat is the arithmetic mean of Heat for odd and even rounds of the steady-state pattern, no penalty is charged for the "natural" fixed buddy cores from Sect. 5.

leable, pre-emptive tasks and continuous frequencies to moldable tasks and discrete frequencies [15]. Alternatively, it may set out from a temperature-unaware crown schedule generated by the ILP (or heuristic) described in [18]. The heuristic then applies the temperature model to identify cores that would become too hot. For each such core, it linearly searches for a free core to be used as a buddy resource for all tasks of the overheated core. If at some point there is no free core left although one were required to be designated buddy, the heuristic immediately terminates without a feasible solution. If on the other hand each core in need of a buddy is assigned one, the heuristic again checks whether there remain cores which become too hot. Accordingly, either a feasible solution has been found, or the heuristic did not manage to alter the original solution in such a way that it is ultimately feasible. We have conducted experiments for the mergesort and H.263 encode task sets, with DVFS enabled and $\alpha = \beta = 0$ (i.e. assuming no heat diffusion to neighboring cores). The heuristic encounters a feasible solution in exactly those cases where the ILPs of Sections 5 and 6 deliver one. This holds true for both kinds of original schedules, the temperature-unaware crown schedules as well as the schedules produced by the heuristic.

Finally, we demonstrate that temperature-aware scheduling may lead to higher energy efficiency compared to temperature-agnostic scheduling when employing a temperature-dependent power model. We set $d = 3$, maxHeat $= 0.4$, and assume $P_{diff}(T_{hot}) = 0.5\,\mathrm{W}$ based on Figure 1 in [10]. Figure 8 shows the energy consumption predicted by the scheduler for temperature-aware and temperature-agnostic scheduling under various hotness threshold values C_{hot}. It can be noted that hotness thresholds have to be rather high (0.7 for the mergesort task set and 0.9 for the H.263 encode task set) for energy efficiency not to improve when performing temperature-aware scheduling. What is more, lower

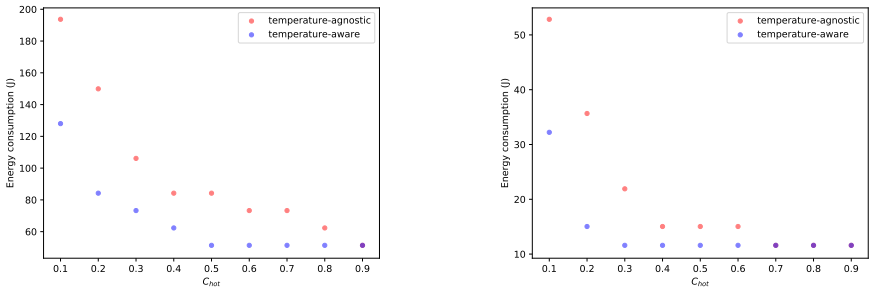

Fig. 8. Predicted energy consumption under various hotness threshold values. Left: H.263 encode task set, right: mergesort task set.

values of C_{hot} may decrease energy consumption by up to 43.8% (H.263 encode) or 57.8% (mergesort) with regard to temperature-agnostic scheduling.

8 Conclusion and Future Work

We addressed the problem of deadline-constrained, energy-efficient temperature-aware scheduling of moldable tasks on a many-core CPU with a 2D mesh geometry. We introduced the buddying technique as an additional, software-only configuration option for tasks to control energy and temperature of cores. We integrated the buddying technique into heuristic and (crown-)optimal schedulers for moldable tasks. Our experimental results indicate that buddying adds flexibility to the scheduler to significantly improve the chances for finding a feasible solution in temperature-constrained scenarios, compared to temperature-unaware scheduling. Future work will comprise experiments with further streaming task graphs and validations by execution on real platforms, as well as extensions of the buddying concept to heterogeneous platforms and e.g. by allowing task and buddy task to use different task variants.

Acknowledgments

C. Kessler acknowledges partial funding by ELLIIT, project GPAI.

References

1. Alkabani, Y., Koushanfar, F., Potkonjak, M.: N-version temperature-aware scheduling and binding. In: Proc. Int. Symposium on Low Power Electronics and Design, San Francisco, CA, USA. pp. 331–334. ACM (Aug 2009)
2. Bampis, E., Letsios, D., Lucarelli, G., Markakis, E., Milis, I.: On multiprocessor temperature-aware scheduling problems. In: Frontiers in Algorithmics and Algorithmic Aspects in Information and Management. pp. 149–160. Springer (2012)

3. Bao, M., Andrei, A., Eles, P.I., Peng, Z.: Temperature-aware task mapping for energy optimization with dynamic voltage scaling. In: 11th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems. pp. 44–49 (2008)
4. Bao, M., Andrei, A., Eles, P.I., Peng, Z.: Temperature-aware idle time distribution for energy optimization with dynamic voltage scaling. In: Proc. Design, Automation, and Test in Europe Conference and Exhibition (DATE'10). pp. 21–26 (2010)
5. Chantem, T., Hu, X.S., Dick, R.P.: Temperature-aware scheduling and assignment for hard real-time applications on MPSoCs. IEEE Trans. Very Large Scale Integr. Syst. **19**(10), 1884–1897 (2011). https://doi.org/10.1109/TVLSI.2010.2058873
6. Coskun, A.K., Rosing, T.S., Whisnant, K.: Temperature aware task scheduling in MPSoCs. In: Proc. DATE'07 (2007)
7. Eindhoven Technical University, Electronic Systems: Dataflow Benchmark Suite (DFbench). `http://www.es.ele.tue.nl/dfbench/` (2010)
8. Eitschberger, P.: Energy-efficient and Fault-tolerant Scheduling for Manycores and Grids. Ph.D. thesis, FernUniversität in Hagen, Germany (2017)
9. Holmbacka, S., Keller, J.: Workload type-aware scheduling on big.LITTLE platforms. In: Algorithms and Architectures for Parallel Proc. pp. 3–17. Springer (2017)
10. Hällis, F., Holmbacka, S., Lund, W., Slotte, R., Lafond, S., Lilius, J.: Thermal influence on the energy efficiency of workload consolidation in many-core architectures. In: 24th Tyrrhenian Int.l Workshop Digital Commun. pp. 1–6 (2013)
11. Jaja, J.: An introduction to parallel algorithms. Addison Wesley (1992)
12. Jayaseelan, R., Mitra, T.: Temperature aware scheduling for embedded processors. J. Low Power Electron. **5**(3), 363–372 (2009)
13. Kahn, G.: The semantics of a simple language for parallel programming. In: Proc. IFIP Congress on Information Processing. pp. 471–475. North-Holland (1974)
14. Kessler, C., Litzinger, S., Keller, J.: Static scheduling of moldable streaming tasks with task fusion for parallel systems with DVFS. IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems (TCAD) **39**(11) (2020)
15. Litzinger, S., Keller, J.: Influence of discretization of frequencies and processor allocation on static scheduling of parallelizable tasks with deadlines. PARS-Mitteilungen **35**(1), 95–108 (2020)
16. Lu, S.J., Tessier, R., Burleso, W.: Reinforcement learning for thermal-aware many-core task allocation. In: Proc. GLSVLSI'15. ACM (May 2015)
17. Melot, N., Kessler, C., Eitschberger, P., Keller, J.: Co-optimizing core allocation, mapping and DVFS in streaming programs with moldable tasks for energy efficient execution on manycore architectures. In: Proc. 19th Int. Conf. on Application of Concurrency to System Design (ACSD 2019) (2019)
18. Melot, N., Kessler, C., Keller, J., Eitschberger, P.: Fast Crown scheduling heuristics for energy-efficient mapping and scaling of moldable streaming tasks on manycore systems. ACM Trans. Archit. Code Optim. **11**(4) (Jan 2015)
19. Pierson, J., Stolf, P., Sun, H., Casanova, H.: MILP formulations for spatio-temporal thermal-aware scheduling in cloud and HPC datacenters. Clust. Comput. **23**(2), 421–439 (2020). https://doi.org/10.1007/s10586-019-02931-3
20. Rajan, D., Yu, P.S.: Temperature-aware scheduling: When is system-throttling good enough? In: Proc. 9th Int. Conference on Web-Age Information Management (WAIM'08), Zhangjiajie, China. pp. 397–404. IEEE Computer Society (Jul 2008)