Scheduling Moldable Parallel Streaming Tasks on Heterogeneous Platforms with Frequency Scaling

Sebastian Litzinger, Jörg Keller
Faculty of Mathematics and Computer Science
FernUniversität in Hagen
58084 Hagen, Germany
First.Last@FernUni-Hagen.de

Christoph Kessler
Dept. of Computer and Information Science (IDA)
Linköping University
58183 Linköping, Sweden
Christoph.Kessler@liu.se

***Abstract*—We extend static scheduling of parallelizable tasks to machines with multiple core types, taking differences in performance and power consumption due to task type into account. Next to energy minimization for given deadline, i.e. for given throughput requirement, we consider makespan minimization for given energy or average power budgets. We evaluate our approach by comparing schedules of synthetic task sets for big.LITTLE with other schedulers from literature. We achieve an improvement of up to 33%.**

***Index Terms*—static scheduling, energy-efficient execution, streaming tasks, heterogeneous platform**

I. Introduction

Signal processing applications are often implemented by a set of streaming tasks. Each task does a specific job, input tasks take input and follow-up tasks are provided with results from predecessor tasks. All tasks are activated repeatedly, as the input data repeatedly arrives, i.e. forms a data stream. Consider for example an image processing algorithm where an initial task enhances sharpness of an input image, while two follow-up tasks apply different filtering algorithms to this image, and a final task combines the filtered variants into one image again. As data streams typically are to be processed with a given throughput requirement such as number of processed images per second, this leads to a maximum time span for every execution unit for executing its assigned tasks once.

In order to fulfill the throughput requirement, it might be necessary to parallelize tasks to reduce their runtime. This is especially true for low task count, so that a multicore processor could not be completely filled if the tasks remained sequential.

At the same time, energy consumption has become a first-class design constraint. The tasks should be run in a manner that, while meeting the deadline requirement, they consume as little energy as possible per round. Given the throughput requirement, this also incurs a minimum average power consumption. This is especially important if a higher power consumption would necessitate changes to the device itself, such as adding an opening with a fan to the chassis, which in turn might mean higher operational and maintenance cost. Thus, an additional goal might also be to maximize throughput, i.e. minimize makespan of one scheduling round, for a given (average) power budget. Controlling power and energy consumption can be achieved by running the tasks at a frequency (resp., frequency-voltage) level as low as possible. If task runtimes are long enough, it is even possible to switch the frequency level between tasks without noticeable overhead.

As a device might have to execute only one or few fixed applications throughout its lifetime, and execution of this application comprises many rounds of task invocation, it pays off to compute an optimized static schedule. Crown scheduling [1] is a static scheduling approach for this situation, i.e. it provides a static schedule to execute a set of tasks on a parallel platform until a deadline, parallelizes the tasks if necessary, and scales the frequencies for energy efficiency. Crown scheduling assumes a homogeneous platform. Current platforms are often heterogeneous, e.g. ARM's big.LITTLE platform comprises two types of cores with identical instruction set architecture but different speed and power consumption profiles. Moreover, the behavior on different core types, e.g. how much faster a task is on a big core compared to execution on a LITTLE core, or the concrete power consumption of that core while executing the task, depends on the instruction mix. [2] characterize tasks into a small number of categories and provide speed and power profiles for big.LITTLE architectures.

We try to bring both worlds together and present a scheduling algorithm based on crown scheduling for heterogeneous platforms and tasks with task types. The scheduling algorithm is based on solving an integer linear program (ILP) or mixed integer linear program (MILP), depending on optimization goal. As the number of tasks for a signal processing application is often moderate, this is no disadvantage, as scheduling problems for task sets of moderate size can still be optimally solved. As the scheduling is only done once before deploying the application, the savings during execution of the application — possibly for years and possibly in thousands of installations — far outweigh the energy invested in the scheduling. Our contributions thus comprise:

- We present a static scheduling algorithm for a set of tasks on a heterogeneous platform with frequency scaling, to meet a deadline and minimize energy consumption, given that the tasks are of different types and thus have different power and speed profiles on this platform.
- We extend the scheduling algorithm to situations where an energy budget per round or an average power budget is given, and the makespan for this round is minimized.

978-9-0827-9703-9/19/$31.00

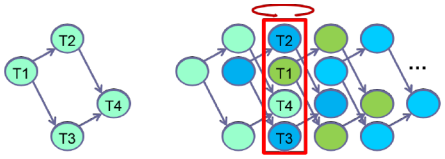

Fig. 1. Left: A streaming task graph. Right: The steady state of the streaming pipeline (rectangle) consists of n independent (instances of) streaming tasks.

- We perform experiments with profiles of ARM's big.LITTLE architecture that have been tested to be accurate. We compare our results with Crown Scheduling, already extended for a heterogeneous architecture but without task types, and with the scheduling from [2], where tasks remain sequential. We achieve improvements of up to 33%. To do so, we extend the original Crown Scheduling MILP, and modify both MILPs to address the additional target functions.

The remainder of this paper is organized as follows. In Section II, we provide technical background information and discuss related work. In Section III we present the heterogeneous crown scheduling algorithm for tasks with types, and extend this algorithm to situations where a fixed energy budget per round or a fixed average power budget is given and the makespan is minimized. In Section IV we report on experiments to evaluate our scheduling algorithm. In Section V we give our conclusions and an outlook to future work.

II. Background and Related Work

A. Scheduling Streaming Applications

Streaming computations (such as Kahn Process Networks [3]) model signal processing applications [4] by a graph of streaming tasks that repeatedly process and forward packets of data flowing along buffered communication channels. Applied to a large input data stream, pipelining the task executions for subsequent data results in a cyclically executed steady-state pattern, also referred to as a round, where task instances are independent (see Fig. 1). A required minimum throughput thus results in a deadline for the round on every execution unit.

A static schedule is computed once prior to execution, which tells each processor the list of tasks it has to execute in one round. The execution is done by a user-level scheduler, where each core simply calls its assigned tasks in round-robin order. A barrier per round might be necessary, but when tasks are parallelized the synchronization of cores happens implicitly. As the tasks are executed on a multicore platform, communication between tasks is via the shared memory, and thus fast. We do not model the dependencies between tasks explicitly, so it might happenthat processing one input data packet might take t rounds in the worst case, where t is the maximum number of nodes on a path through the graph, e.g. $t = 3$ in Fig. 1. However, usually part of the dependencies can be covered even within one round, if dependent tasks are scheduled later than their predecessors, thus reducing processing latency without affecting throughput.

B. Speed and Energy on Heterogeneous Platforms

The execution speed of a task on a processor core depends on many factors. Most prominent among them is the operating frequency of the core, as it determines the length of one processor cycle. The microarchitecture of the core also influences the runtime of a task, because it determines the number of cycles needed for each instruction, and how the instructions can be overlapped by pipelining or superscalar execution. For two cores with identical instruction set architectures but different microarchitectures, the same code will lead to different power consumptions or different runtimes or both. While the "stronger" core typically leads to shorter runtime, it often cannot be predicted which core leads to lower energy consumption. ARMs big.LITTLE[1] is an example for such a heterogeneous architecture. It comprises multiple cores with identical instruction set architecture but with different microarchitecture (e.g. A7 and A15). LITTLE cores have a simpler architecture but are less power-hungry than big cores.

A core's power consumption depends on its operating frequency, if we assume that the supply voltage is always set at the minimum possible value and the temperature is controlled, on the instructions that are executed, and on the structure of the core's execution units to execute these instructions. Assuming that a task's instruction mix is stable over its execution, i.e. the power consumption is stable, too, then the energy to execute the task is the product of power consumption and execution time. Thus, there are complex relationships between execution time, power consumption and energy consumption for a task. It may e.g. not be energy-efficient to execute a task on a low frequency instead of a higher, as the reduction in power consumption may not make up for the increase in runtime.

C. Related Work

Pruhs et al. [5] provide a static scheduler to minimize makespan for a task set on a parallel machine with frequency scaling for a given energy budget. Their tasks are sequential, have no task types, have dependencies, the frequency f can be scaled to an arbitrary continuous value, the power function is restricted to f^α, and the machine is homogeneous.

Melot et al. [1] and Xu et al. [6] propose static schedulers to minimize energy consumption for a task set on a parallel machine with discrete frequency levels, given a deadline. While their tasks are parallelizable, they do not have task types, and the machine is homogeneous. Also, they do not address energy or power budgets.

Holmbacka and Keller [2] investigate power profiles of ARM big.LITTLE for tasks of different types, and provide a static scheduling for task sets given a deadline. However, their tasks are sequential, and their scheduler enumerates solutions. Also, they do not address power or energy budgets.

Kuang and Bhuyan [7] optimize latency and throughput for network packet processing under a given power budget. However, their tasks are sequential, have no task types, have dependencies, and use a given mapping of tasks to processors.

[1] https://developer.arm.com/technologies/big-little

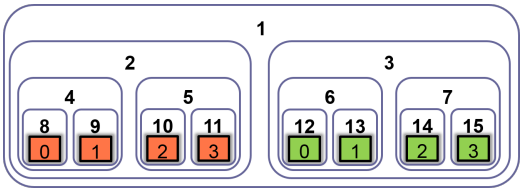

Fig. 2. A binary crown for $p = 8$ cores of 2 different types, where the core types are given by the color coding (orange = A15-cores (big), green = A7-cores (LITTLE)). The boldface numbers $1, \ldots, 15$ show the processor group indices of the crown.

Sarood et al. [8] present a dynamic scheduler for HPC cluster to increase performance under a strict power budget. In contrast, we target static schedules and average power budget.

Zahaf et al. [9] present a static scheduler for task sets of different task types on a heterogeneous parallel machine, in particular ARM big.LITTLE. In contrast to our work, their tasks have individual deadlines and periods, they use a non-linear program, and do not address energy or power budgets.

III. Scheduling for Energy and Power Efficiency

A. Original Crown Scheduler

Crown scheduling considers a set of n independent, parallelizable (i.e. moldable) tasks t_j, each with given workload τ_j (given in number of cycles) and parallel efficiency $e_j(w_j)$ when executing the task on $w_j \leq W_j$ processors, where W_j is the maximum degree (width) of parallelism. The runtime (in sec.) for a task executed at frequency f (cycles per second) is

$$\frac{\tau_j}{f \cdot w_j \cdot e_j(w_j)} \ .$$

The machine to execute the task set has p cores, each scalable to a finite number s of discrete frequency levels f_k. For each frequency level, the power consumption of the core $Pow(f_k)$ is known. Thus, the energy consumption of a task running at f_k is the product of runtime, power consumption per core and the number of cores

$$\frac{\tau_j}{f_k \cdot w_j \cdot e_j(w_j)} \cdot Pow(f_k) \cdot w_j = \frac{\tau_j \cdot Pow(f_k)}{f_k \cdot e_j(w_j)} \ .$$

Each task is allocated a width, mapped to a set of cores, and assigned an operating frequency. This is to be done such that all tasks are executed until a given deadline M is reached and that the energy consumption is minimized.

To simplify the scheduling complexity, the number of possible allocations and mappings is greatly reduced. A task can only be allocated a width that is a power of 2. The mapping of a task to a set of cores can only be done according to the hierarchy of core groups i, each of size p_i, as depicted in Fig. 2. Thus, there are only $2p - 1$ processor groups for mapping: one with p cores, two with $p/2$ cores, and so on.

Integrated Crown Scheduling considers allocation, mapping and frequency together as an optimization problem in an integer linear program (ILP). The ILP uses $ns(2p-1)$ binary decision variables $x_{i,j,k}$ where $x_{i,j,k} = 1$ if task j is mapped to processor group i at frequency level f_k. Two constraints to be fulfilled are that each task j is only mapped once, i.e.

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

and that no task's mapping supersedes its maximum width:

$$\forall j : \sum_{i:p_i > W_j} \sum_{k} x_{i,j,k} = 0 \ . \tag{2}$$

The remaining constraints and target function are presented when adapting Crown scheduling to a heterogeneous platform.

B. Heterogeneous and Task type-Aware Crown Scheduling

To fit Crown scheduling to a heterogeneous multicore machine, we assume that the heterogeneous machine consists of 2^q core types, each with $p/2^q$ cores. All processor groups that span more than one type of core remain empty, as each task is only parallelized on one type of core. These are groups $i = 1$ to $2^q - 1$, so that only groups of size at most $p/2^q$ remain. This can e.g. be achieved by restricting the task widths $w_i \leq p/2^q$. In Fig. 2, $2^q = 2$ core types are present, so that group $i = 1$ will remain empty by setting the maximum widths to at most 4. The processor groups in Crown scheduling can be adapted to architectures with numbers of core types that are no power of two, and different core counts for the different core types.

To correctly compute runtime and energy of a task depending on the core type the task runs on, we scale the workload with a core type-dependent constant factor r_i that expresses the relative performance of different core types. For big.LITTLE, the factor for LITTLE cores is set to 1, and the factor for big cores will normally be less than 1 as the runtime on a big core will be shorter than on a LITTLE. Additionally, there must be a power profile for each core type. In the equations, we index relative performance and power profile by the processor group index i, which however uniquely defines the core type.

To introduce task-type awareness, relative performance and power profiles must additionally be indexed by task index j (actually the task's type, but that is uniquely determined by the task index). We obtain optimization problem $\min E$ with

$$E := \sum_{i,j,k} x_{i,j,k} \cdot \frac{\tau_j \cdot r_{i,j} \cdot Pow(f_k, i, j)}{f_k \cdot e_j(p_i)} \tag{3}$$

under the constraint that for each core l, the runtime of that core must not exceed the deadline, i.e. $T_l \leq M$ where

$$T_l := \sum_{i \in G_l, j, k} x_{i,j,k} \cdot \frac{\tau_j \cdot r_{i,j}}{f_k \cdot p_i \cdot e_j(p_i)} \ . \tag{4}$$

Here G_l denotes the set of all groups that comprise core l. Constraints (1) and (2) remain as in the Crown scheduler. Variables E and T_l are no variables in the ILP but only abbreviations for expressions (3) and (4). Besides the variables $x_{i,j,k}$, the symbols in the formulae like τ_j or $Pow(f_k, i, j)$ are *constants* in the ILP representing the taskset or the platform.

Please note that the approach by Keller and Holmbacka can be considered a special case of our scheduling where all maximum task widths are artificially set to 1, i.e. the tasks remain sequential.

C. Scheduling for Fixed Energy and Power Budgets

The ILP from Subsec. III-B is quite general with respect to optimization target. When a fixed energy budget E_{max} is given instead of a fixed deadline M to be met, we just switch the target function for energy and the time constraint and obtain:

$$\min T_{max}$$

under constraints

$$\forall l: \; T_l \leq T_{max} \; ,$$

$$E \leq E_{max} \; .$$

We must introduce an additional variable T_{max} because the time constraint from the previous subsection is really a constraint for each core, which cannot directly be transferred to a target function. Thus, the ILP becomes a mixed ILP (MILP). Constraints (1) and (2) remain as they are.

If an average power budget P_{avg} is given, we again use $\min T_{max}$ as optimization problem with the accompanying constraint $\forall l : T_l \leq T_{max}$. We relate time, average power and energy in the usual way and obtain the constraint:

$$E \leq P_{avg} \cdot T_{max} \; .$$

Because of the target function, the ILP solver tries to make the inequality's right hand side, i.e. T_{max}, as small as possible. However, this increases the energy E, i.e. the left hand side, so that the ILP solver strives towards bringing both sides as close as possible. Again, constraints (1) and (2) remain as they are. For convenience, all MILPs are summarized in the appendix.

IV. Experiments

In our experiments, we consider synthetic task sets with 10, 20, 40, and 80 tasks as in [1]. For each cardinality, 10 task sets are chosen and scheduled. The workloads are taken from [1], in units of 10^6 cycles on LITTLE. Thus, the performance factors $r_{i,j}$ can be taken from [2, Table 4]. Each task is randomly and uniformly assigned one of the types MEMORY, BRANCH, FMULT, SIMD, or MATMUL as in [2]. To ensure that task sets are suitable for parallelization, the joint workload of BRANCH tasks is limited to 10% of the total workload. A task's maximum width is set with regard to its type as follows:

$$W(j) = \begin{cases} 1, & \text{if } j \text{ is of type BRANCH,} \\ w_j \in \{2,4\}, & \text{if } j \text{ is of type MEMORY or FMULT,} \\ 4, & \text{if } j \text{ is of type SIMD or MATMUL.} \end{cases}$$

For task types SIMD and MATMUL, considerable parallelization potential can be expected, whereas tasks mainly consisting of branch instructions may have to be executed sequentially. For MEMORY and FMULT task types, some parallelization seems realistic, so the respective widths are either 2 or 4 (randomly determined based on a uniform distribution).

The big cores' highest frequency level is not available on the LITTLE cores and is therefore ignored during scheduling, hence the set of possible operating frequencies is $\{0.6, 0.8, 1.0, 1.2, 1.4\}$ GHz. As workload is in 10^6 cycles, runtimes will be in milliseconds.

The parallel efficiency functions for all tasks j are defined as $e_j(1) = 1.0$, $e_j(2) = 0.9$ and $e_j(4) = 0.86$. Thus, we account for the initial overhead incurred by any kind of parallelization as well as the diminishing efficiency due to the increasing number of processing elements involved in the computations.

We derive our power profile $Pow(f_k, i, j)$ from [2, Table 3]. The interested reader can find details in the appendix. As the power is given in Watt, and runtimes in milliseconds, energy values are computed in milli Joule.

The experiments comprise three scheduling approaches: the task type-aware approach for sequential tasks (TAS) from [2], the task type-ignorant crown scheduler for parallelizable tasks (TIP) from [1] (adapted to the heterogeneous platform) and our task-type aware crown scheduler for parallelizable tasks (TAP). We investigate three optimization targets, of which we will only present the first scenario (given deadline constraint, minimize energy) in detail for space restrictions, and summarize the other scenarios (for details cf. appendix).

We derive the deadline for each task set by computing the average between running all tasks utilizing all processors on highest and lowest frequency, respectively:

$$M = 0.6 \cdot \frac{\frac{\sum_j \tau_j}{p \cdot f_1} + \frac{\sum_j \tau_j}{p \cdot f_s}}{2}. \tag{5}$$

The factor 0.6 helps to emphasize the differences between sequential and parallel scheduling, as some task sets might not be schedulable in TAS under tight deadline constraints while parallel scheduling still is. Energy and power budgets are also derived from the task set characteristics. The interested reader can find details in the appendix.

All schedulers were implemented in Python utilizing the gurobipy module. Accordingly, the Gurobi 8.1.0 solver was deployed. It was executed on an AMD Ryzen 7 2700X in 16 threads (8 physical cores). A 5 minute timeout for solving each individual (M)ILP was in effect.

Table I provides information on scheduling time[2], number of timeout occurrences, and number of infeasible models for each of the combinations of scenario and scheduling approach. Scheduling tasks sequentially under a deadline (scenario 1) or energy (scenario 2) constraint does not permit a valid schedule in some small task sets. Apart from that, runtimes are heavily influenced by the number of ILPs which cannot be solved to optimality within the 5 minute time limit. Most of the remaining ILPs are solved within seconds.

Table II shows average makespan and average energy consumption for each task set cardinality, relative to task-aware crown scheduling (TAP) as well as the number of deadline violations for task type-ignorant scheduling (TIP) when optimizing for energy consumption. Scheduling larger task sets for sequential execution does not yield a significant difference with regard to the parallel scheduler, neither concerning makespan nor energy consumption. However, for some small task sets, no feasible sequential schedule exists, while

[2]All runtimes are sums of user and system times, whereas the time limit is set in real (wall clock) time.

TABLE I
RUNTIME, TIMEOUT OCCURRENCES AND NUMBER OF INFEASIBLE MODELS FOR ALL SCENARIOS AND SCHEDULING APPROACHES

scenario	scheduling	runtime [min]	#timeouts	#infeasible
	TAP	563	6	0
1	TAS	637	7	4
	TIP	254	2	0
	TAP	764	9	0
2	TAS	797	9	2
	TIP	1383	15	0
	TAP	683	8	0
3	TAS	733	9	0
	TIP	1653	18	0

TABLE II
RESULTS FOR SCENARIO 1, RELATIVE TO TAP

scheduling	task set card.	makespan	energy	#deadline viol.
	10	1.000	1.046	
	20	1.000	1.001	
TAS	40	1.000	1.000	
	80	1.000	1.000	
	total	1.000	1.008	
	10	1.246	1.259	7
	20	1.225	1.316	8
TIP	40	1.157	1.313	9
	80	1.109	1.341	8
	total	1.184	1.307	32

parallel execution can be completed prior to the deadline in any case. Task type-ignorant scheduling leads to an increase in both makespan (more pronounced for small task sets) and energy consumption (more pronounced for larger task sets). It should be noted that in 80% of all cases, task type-ignorant scheduling brings about a deadline violation.

In scenario 2 (given energy budget, minimize makespan), the task type-ignorant scheduler again performs considerably worse than the other two for all task set cardinalities, the difference being yet more noticeable than in the first scenario. Moreover, roughly 25% of the task type-ignorant schedules do not comply with the energy budget. Only for small task sets, sequential scheduling leads to an increase in makespan compared to parallel scheduling. In medium and large task sets, most of the tasks are executed sequentially in any case. Thus, no deviation from the sequential schedule is to be expected even if scheduling tasks in parallel is permitted. Again, for some small task sets no feasible solution of the respective ILP can be obtained under sequential scheduling, while scheduling in parallel always yields a valid schedule. Result details can be found in the appendix.

For makespan optimization under an average power budget (scenario 3), the schedulers' relative performance is very similar to the one in scenario 2. The sequential scheduler manages to narrow the gap to TAP for small task sets. Due to the nature of the ILP constraints, a feasible solution can always be found: to lower the power, just stretch the runtime by lower frequency. Interestingly, the task type-ignorant scheduler overestimates energy consumption and consequently does not fully exploit the given power budget — at the cost of an increased makespan. Result details can be found in the appendix.

V. CONCLUSIONS

We have presented a static scheduling algorithm for signal processing applications modelled as a set of streaming tasks. The algorithm either minimizes energy consumption for a given throughput or maximizes throughput for a given energy budget per input or for a given average power budget. The scheduling algorithm allows parallelization of tasks and takes heterogeneity of the platform and different execution characteristics of tasks into account. Our scheduling algorithm is derived as an extension of the Crown Scheduler [1].

We have compared the results for the power and performance profile of an ARM big.LITTLE architecture with restricted versions, i.e. a task type-unaware Crown scheduler (already adapted to the heterogeneous platform) and scheduler from [2], where tasks cannot be parallelized. We achieve advantages for each target function, up to 33%.

Future work will comprise modeling of communication cost between tasks, as e.g. communication between caches of different core types is more expensive than communication within one core type. Also, we will explore explicit modeling of task dependencies to include input-to-output latency as an optimization goal. Finally, we plan experiments with a real ARM board to confirm the effect of our optimization also by measurement.

APPENDIX

We provide an appendix with further details and results under https://e.feu.de/ii .

REFERENCES

[1] N. Melot, C. Kessler, J. Keller, and P. Eitschberger, "Fast Crown scheduling heuristics for energy-efficient mapping and scaling of moldable streaming tasks on manycore systems," *ACM Trans. Archit. Code Optim.*, vol. 11, no. 4, pp. 62:1–62:24, Jan. 2015.

[2] S. Holmbacka and J. Keller, "Workload type-aware scheduling on big.LITTLE platforms," in *Algorithms and Architectures for Parallel Processing*, S. Ibrahim, K.-K. R. Choo, Z. Yan, and W. Pedrycz, Eds. Cham: Springer International Publishing, 2017, pp. 3–17.

[3] G. Kahn, "The semantics of a simple language for parallel programming," in *Proc. IFIP Congress on Information Processing*. North-Holland, 1974, pp. 471–475.

[4] I. Bacivarov, W. Haid, K. Huang, and L. Thiele, "Methods and tools for mapping process networks onto multi-processor systems-on-chip," in *Handbook of Signal Processing Systems, third edition*, S. S. Bhattacharyya et al., Ed. Springer, 2019, pp. 685–719.

[5] K. Pruhs, R. van Stee, and P. Uthaisombut, "Speed scaling of tasks with precedence constraints," *Theory of Computing Systems*, vol. 43, no. 1, pp. 67–80, 2008.

[6] H. Xu, F. Kong, and Q. Deng, "Energy minimizing for parallel real-time tasks based on level-packing," in *Proc. 18th Int. Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2012, pp. 98–103.

[7] J. Kuang and L. N. Bhuyan, "Optimizing throughput and latency under given power budget for network packet processing," in *INFOCOM 2010. 29th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 15-19 March 2010, San Diego, CA, USA*, 2010, pp. 2901–2909.

[8] O. Sarood, A. Langer, A. Gupta, and L. Kale, "Maximizing throughput of overprovisioned hpc data centers under a strict power budget," in *Proc. International Conference for High Performance Computing, Networking, Storage and Analysis (SC '14)*, 2014, pp. 807–818.

[9] H.-E. Zahaf, A. E. H. Benyamina, R. Olejnik, and G. Lipari, "Energy-efficient scheduling for moldable real-time tasks on heterogeneous computing platforms," *Journal of Systems Architecture*, vol. 74, pp. 46–60, 2017.