Scheduling Moldable Parallel Streaming Tasks on Heterogeneous Platforms with Frequency Scaling

Sebastian Litzinger [1] Jörg Keller [1] Christoph Kessler [2]

FernUniversität in Hagen

[1]Faculty of Mathematics and Computer Science, FernUniversität in Hagen, Germany
[2]Dept. of Computer and Information Science (IDA), Linköping University, Sweden

LINKÖPINGS UNIVERSITET

Motivation

- **Signal processing** applications are often implemented by a set of *streaming tasks*
- **Throughput requirement** gives execution unit maximum time span for single execution of assigned tasks (one scheduling round)
- **Low energy consumption** and low average power consumption are desirable with regard to purchasing, operational, and maintenance costs
- High throughput is desirable but power and energy consumption are often **constrained**
- Tasks may have to be **executed in parallel** (if possible) to facilitate **low makespan** of round
- Core operating frequency influences energy consumption as well as runtime
- Architecture might be **heterogeneous**, complicating scheduling
- Tasks may differ in execution speed or power consumption, e. g. due to instruction mix
- **Static scheduling** pays off since application runs for years in a large number of devices
- For **optimal schedule**, solve (mixed) **integer linear program** (MILP/ILP)

Contributions

- 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.
- We perform experiments with accurate profiles of ARM's big.LITTLE architecture

Streaming Task Graph

Each task does a specific job, input tasks take input, follow-up tasks are provided with results from predecessors. All tasks are activated repeatedly, as the input data repeatedly arrives, i. e. forms a data stream.

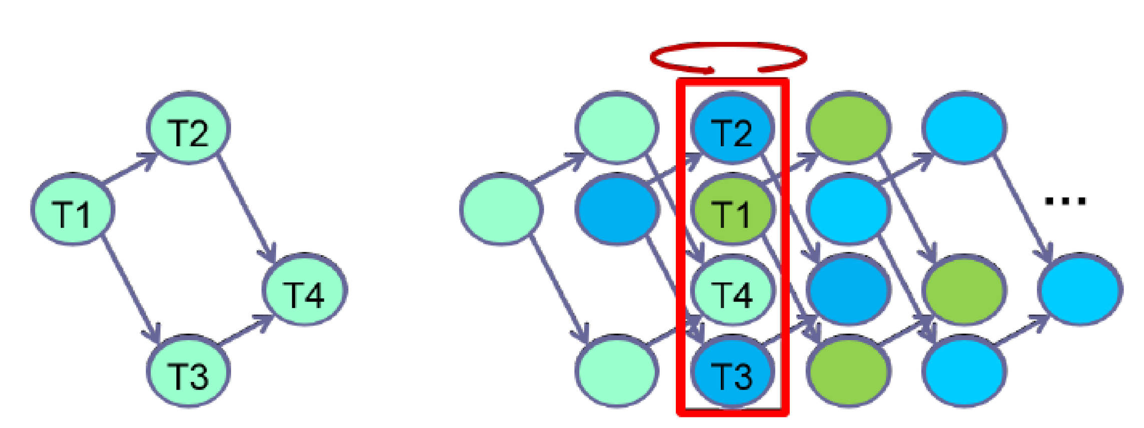

Figure 1. Left: A streaming task graph. Right: The steady state of the streaming pipeline (rectangle) consists of n independent (instances of) streaming tasks.

Further reading

N. Melot et al., "Fast Crown Scheduling Heuristics for Energy-Efficient Mapping and Scaling of Moldable Streaming Tasks on Many-Core Systems," *ACM TACO*, vol. 11, no. 4, pp. 62:1–62:24, 2015.

N. Melot et al., "Co-optimizing Core Allocation, Mapping, and DVFS in Streaming Programs with Moldable Tasks for Energy-Efficient Execution on Manycore Architectures," *Proc. ACSD 2019*, to appear June 2019.

Crown Scheduling

In crown scheduling, a task is mapped to a particular processor group, which lowers scheduling complexity. Possible allocations thus are powers of 2. Moreover, each task is assigned an operating frequency during scheduling.

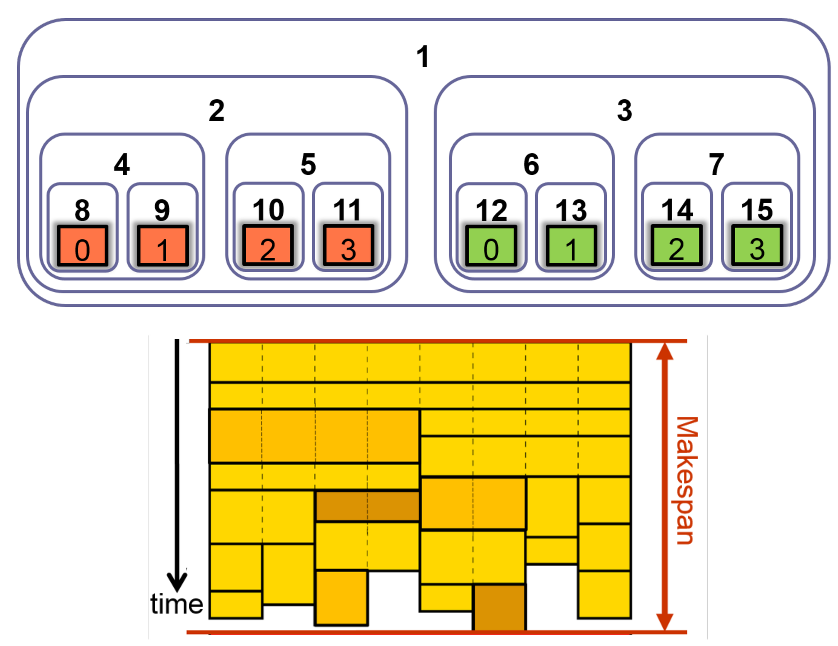

Figure 2. **Top:** A binary crown for $p = 8$ cores of 2 different types, where the core types are given by the color coding (orange = A15-cores (big), green = A7-cores (LITTLE)). The boldface numbers $1, \ldots, 15$ show the processor group indices. **Bottom:** Example crown schedule for an 8-core machine

Optimization problems for scheduling n tasks to p cores with s discrete frequency levels ($x_{i,j,k} = 1$ if task j is mapped to core group i at frequency level f_k):

Variables:
binary $x_{i,j,k}, i = 1..2p-1, j = 1..n, k = 1..s$
real T_{max}

(1) Min. energy E for given deadline M
$\min E$
$\forall l: \ T_l \leq M$

(2) Min. makespan T_{max} for energy budget E_{max}
$\min T_{max}$
$\forall l: \ T_l \leq T_{max}$
$E \leq E_{max}$

(3) Min. makesp. T_{max} for av. power budget P_{avg}
$\min T_{max}$
$\forall l: \ T_l \leq T_{max}$
$E \leq P_{avg} \cdot T_{max}$

Additional constraints for all targets
$\forall j: \ \sum_{i,k} x_{i,j,k} = 1$
$\forall j: \ \sum_{i:p_i > W_j} \sum_k x_{i,j,k} = 0$

Figure 3. (M)ILPs for different optimization targets ("scenarios"). T_l signifies the runtime of core l.

Experiments

- 40 synthetic task sets of varying cardinality (10–80 tasks), 5 different task types
- Real frequencies and power consumption values for the ARM big.LITTLE architecture
- **TAS**: Task type-aware approach for sequential tasks (Keller & Holmbacka 2017)
- **TIP**: Task type-ignorant crown scheduler for parallelizable tasks (Melot et al. 2015)
- **TAP**: Task-type aware crown scheduler for parallelizable tasks
- Implementation in Python with Gurobi solver, 5 minute wall clock timeout for each (M)ILP

Results

Table 1. Runtime, timeout occurrences and number of infeasible models for all scenarios and scheduling approaches

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

Table 2. Results for scenario 1, relative to TAP

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

Scenario 1 (min E, M given):

- TAP vs. TAS: advantage TAP for small task sets (feasible schedule in any case), tasks executed sequentially anyways for larger task sets
- TAP vs. TIP: lower makespan (more pronounced for small task sets), lower energy consumption (more pronounced for larger task sets), TIP: deadline violation in 80% of all cases

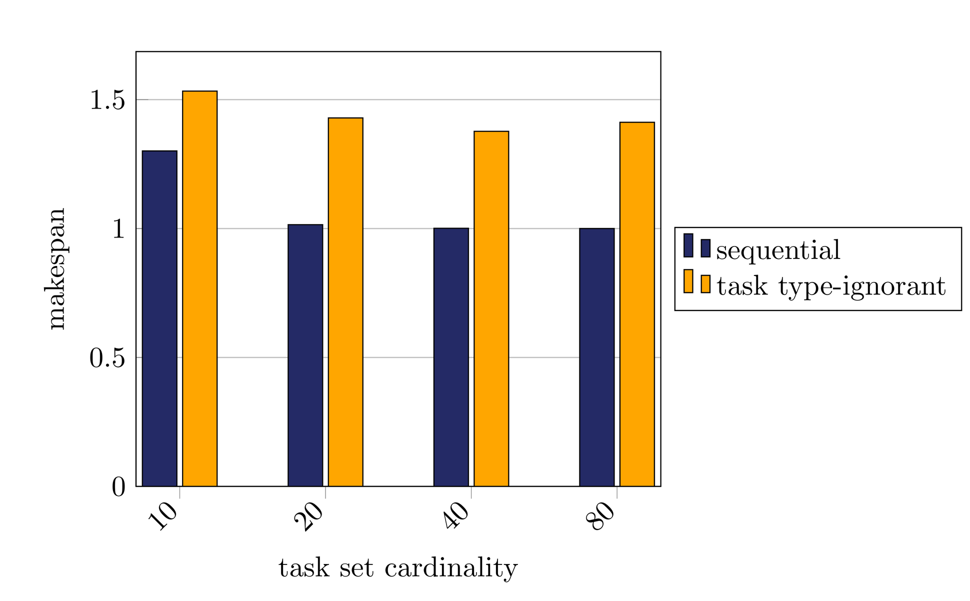

Figure 4. Average makespan for sequential and task type-ignorant scheduling relative to average makespan for parallel scheduling in scenario 2

Scenario 2 (min makespan, E given):

- TAP vs. TAS: same behavior as for scenario 1, relative performance of TAP better
- TAP vs. TIP: TAP's relative performance even better than for scenario 1

Scenario 3 (min makespan, P_{avg} given):

- TAP vs. TAS: TAP still better for small task sets, feasible solution can always be found (due to nature of constraints)
- TAP vs. TIP: lower makespan due to TIP overestimating energy consumption and thus not exploiting power budget