Scheduling Moldable Parallel Streaming Tasks on Heterogeneous Platforms with Frequency Scaling

Sebastian Litzinger, Jörg Keller, **Christoph Kessler**

FernUniv. in Hagen, Germany

Linköping University, Sweden

27th European Signal Processing Conference (EUSIPCO'19), La Coruna, Spain, Sep. 2019. IEEE, Nov. 2019.

Also presented at 12th Nordic Multicore Computing Workshop (MCC-2019), Karlskrona, Sweden.

Multi- / Manycore CPUs

- Many small energy-efficient cores + on-chip memory units
- Can be used for low-power, high-throughput computing, e.g. processing image frame sequences
- Dynamic discrete voltage and frequency scaling (DVFS) of cores or small core groups

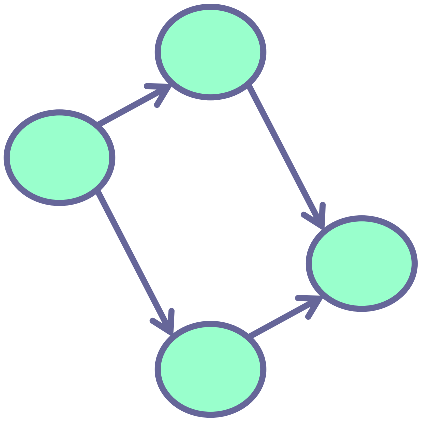
Streaming Computations

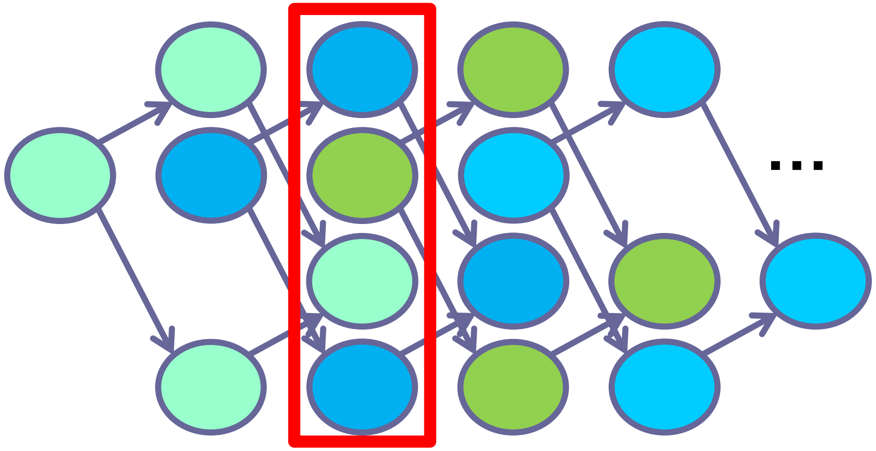
Streaming computations

- Software pipelining
- Concurrent execution of all streaming tasks in steady state

Streaming task graph (= actor network)

- (Acyclic) pipeline graph of streaming tasks
- Producer-consumer communication
- Cf. Kahn Process Networks



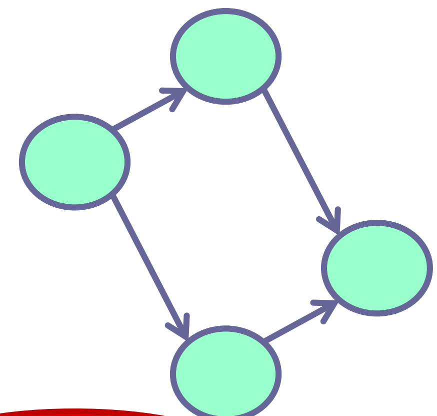


Streaming Computations

Streaming computations

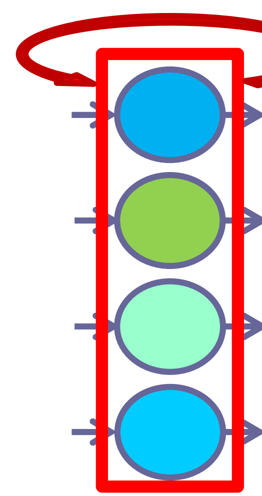
- Software pipelining
- Concurrent execution of all streaming tasks in steady state

Streaming task graph (= actor network)



- (Acyclic) pipeline graph of streaming tasks
- Producer-consumer communication
- Cf. Kahn Process Networks

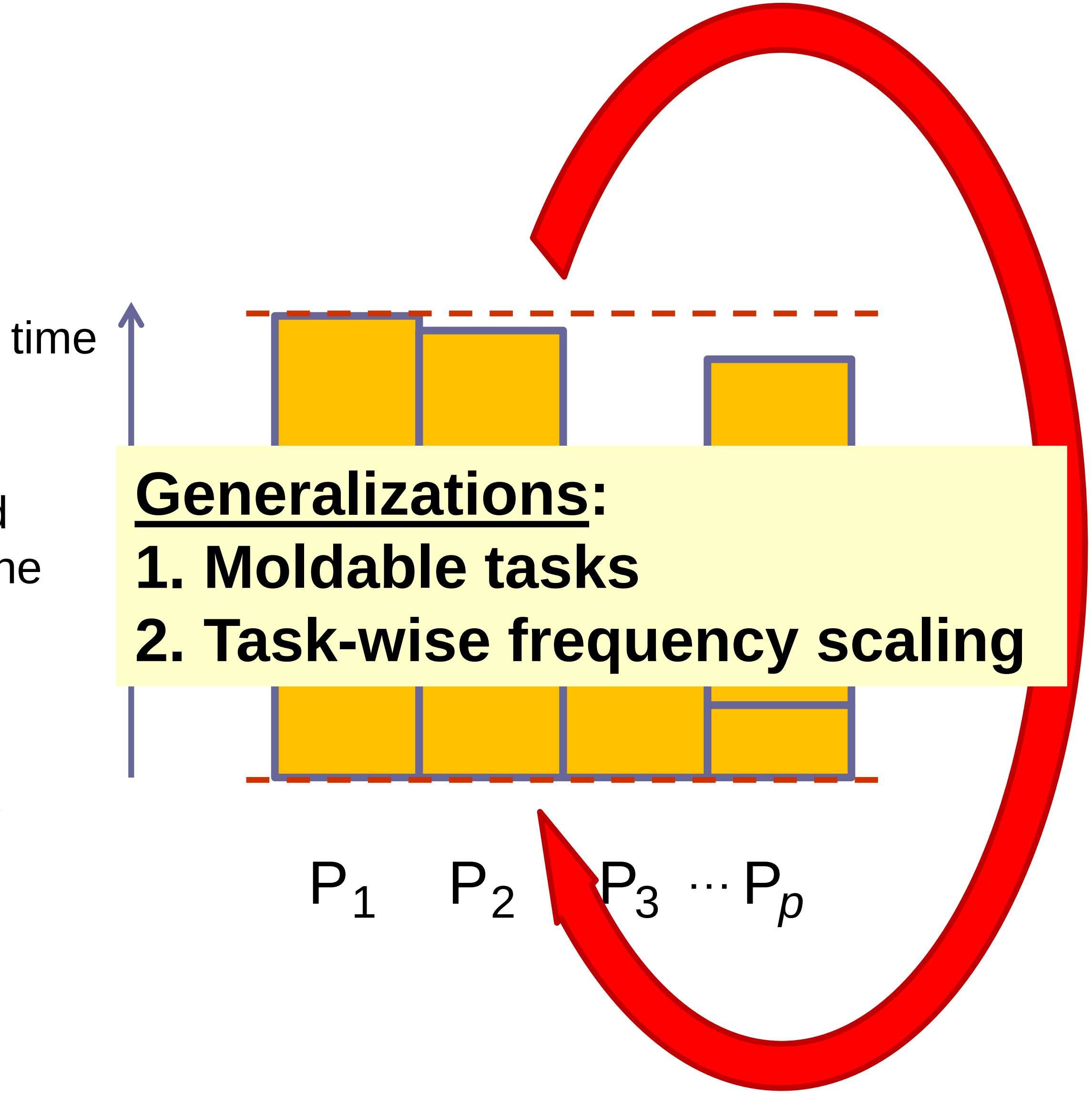
Steady state: Streaming task collection



- Independent streaming task instances
- Balance workload over all cores
  to minimize the makespan
  for one round of the steady state

Mapping Tasks for the Steady State

One round
of the pipeline
kernel



<u>Generalizations</u>:

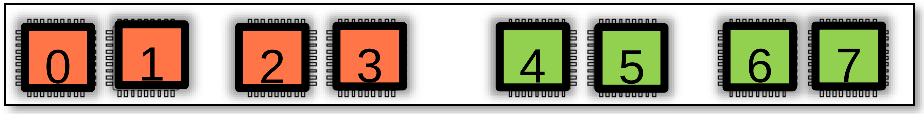
1. Moldable tasks

2. Task-wise frequency scaling

User-level
round-robin
scheduler
on each core

Generic Heterogeneous Multicore CPU Model

Resources



- *p* cores $P_1, \ldots, P_p$ of 2 types
  - e.g., A7 (LITTLE) and A15 cores (big) in ARM big.LITTLE, same ISA

Discrete Voltage and Frequency Scaling

- *s* discrete frequency levels F = { $f_1 = f_{min}, \ldots, f_s = f_{max}$ } (ordered)
  - Includes voltage scaling by auto-co-scaling
  - big, LITTLE: 0.6, 0.8, 1.0, 1.2, 1.4GHz. (top frequency of A15 not used)
  - Assuming: task execution time scales linearly in *f*,
    - with measured performance coefficient $r_{i,j}$ = 1 for big, <1 for LITTLE
- Core can change frequency dynamically at task switching

Power model: for ARM big.LITTLE [Holmbacka, Keller 2017]

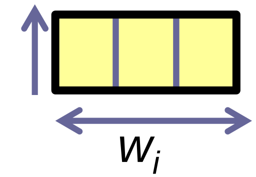
- Task-type specific power values $Pow_j(f, i, j)$ for big and for LITTLE
  - Types: MEMORY, BRANCH, FMULT, SIMD, MATMUL
  - determined by measurements on target

Moldable (Parallelizable) Tasks

Moldable tasks $i = 1,\ldots,n$

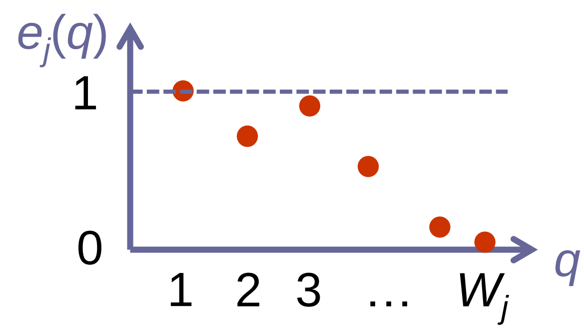
- A task j performs fixed **work** τ_j
- Can be run with (integer) $w > 1$ cores (fixed before the task starts)
  - Resource allocation → task **width** w_j
- <u>Arbitrary</u> scalability functions:

  efficiency $0 < e_j(w) \leq 1$ for $1 \leq w \leq W_j$



time

$$tt_j(w, f) = \frac{\tau_j}{f \cdot e_j(w) \cdot w}$$

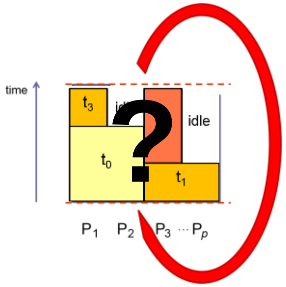


Mixed task model

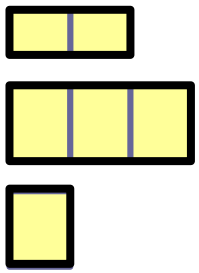
Streaming tasks $j = 1,\ldots,n$ can be

- Inherently sequential (max. width $W_j = 1$)
- Moldable, limited scalability (given maximum width $W_j > 1$)
- Moldable, unlimited scalability ($W_j \geq p$)

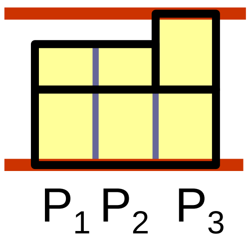
Steady-State Task Scheduling



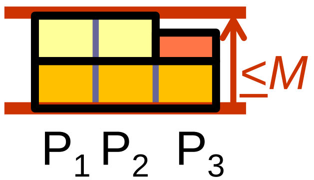
3 subproblems to solve (off-line):



- **Resource allocation**
- **Mapping**
  - Map each task *i* to w_i specific cores



- **Discrete frequency scaling**
  - Select for each task *i* a frequency F_i in $\{f_1, \ldots, f_s\}$



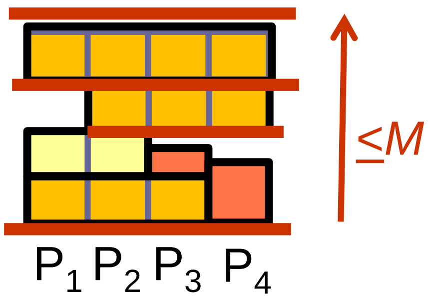
given a **throughput constraint** (round makespan),

to minimize overall energy usage.

Problem – Complexity!

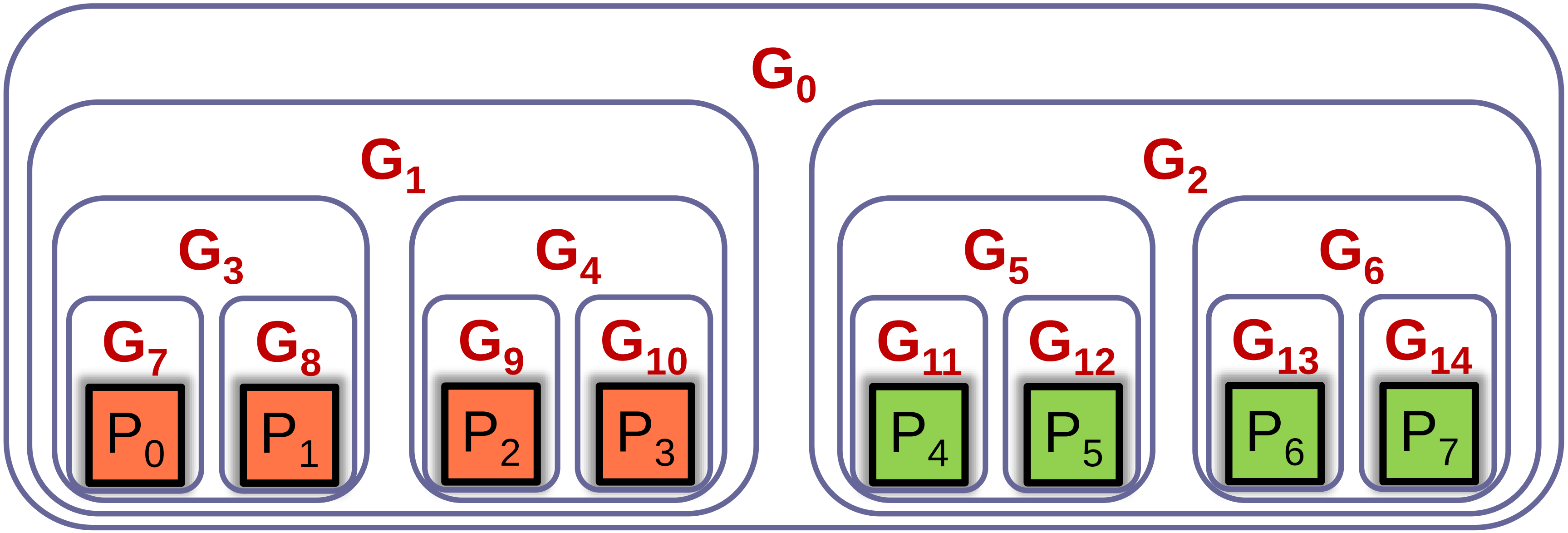
- A *parallel task* should start on all its assigned cores simultaneously
- Scale frequency on all cores of a parallel task equally

→ Idle times within the round that might not be scaled away (internal fragmentation)



Idea: Constrain Resource Allocation

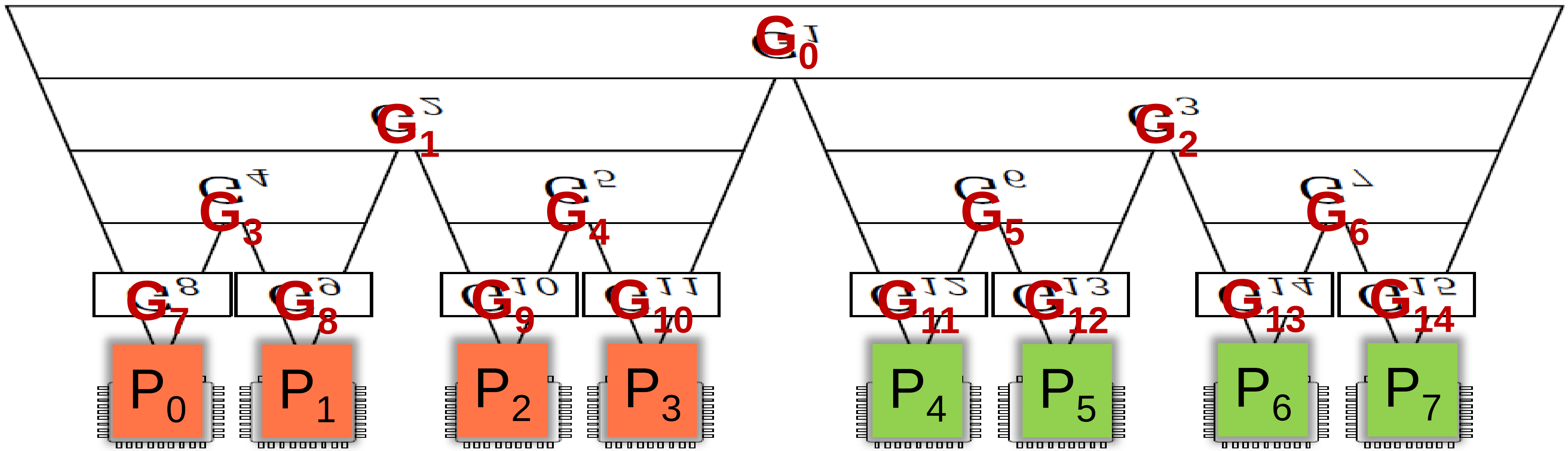
- Define a hierarchy of processor groups
  - **The Crown**
    - Example: A balanced binary crown over 8 cores



    - Tasks' core allocations must be whole groups (here, powers of 2)
    - Reduces #**possible mapping targets** (here, from 2^p-1 to $2p$-1)

Idea: Constrain Resource Allocation

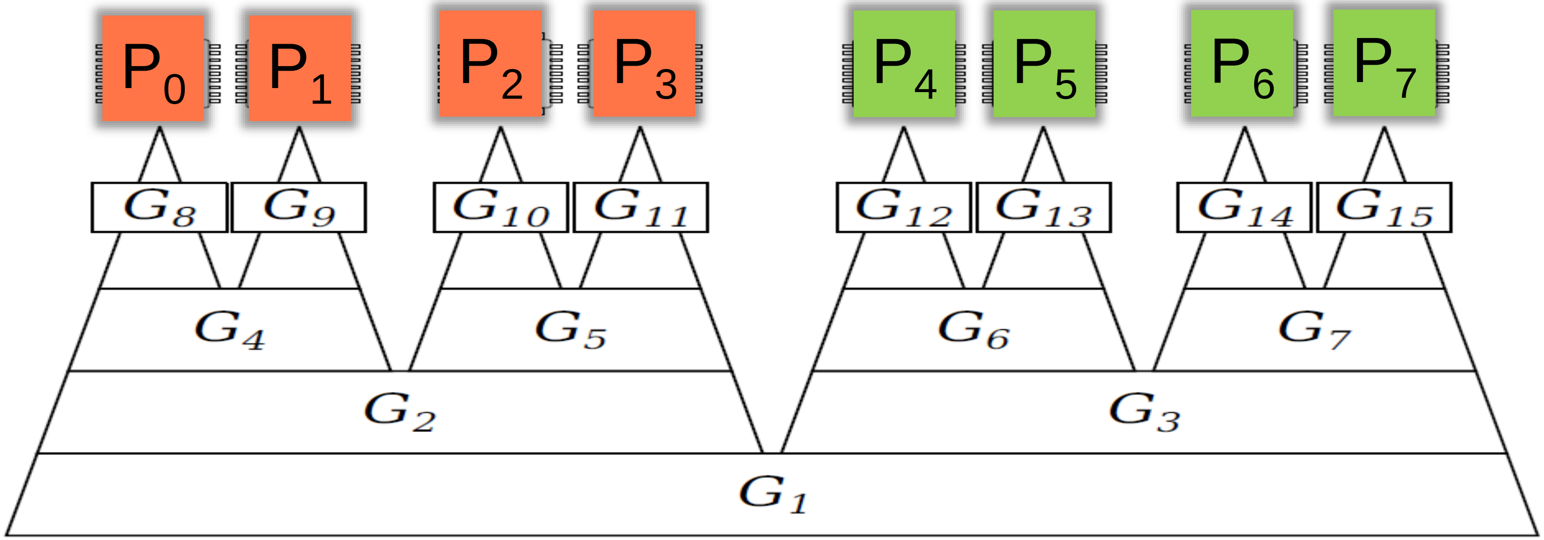
- Define a hierarchy of processor groups
  - **The Crown**
    - Example: A balanced binary crown over 8 cores



- Tasks' core allocations must be whole groups (here, powers of 2)
- Reduces #**possible mapping targets** (here, from 2^p-1 to 2p-1)

Idea: Constrain Resource Allocation

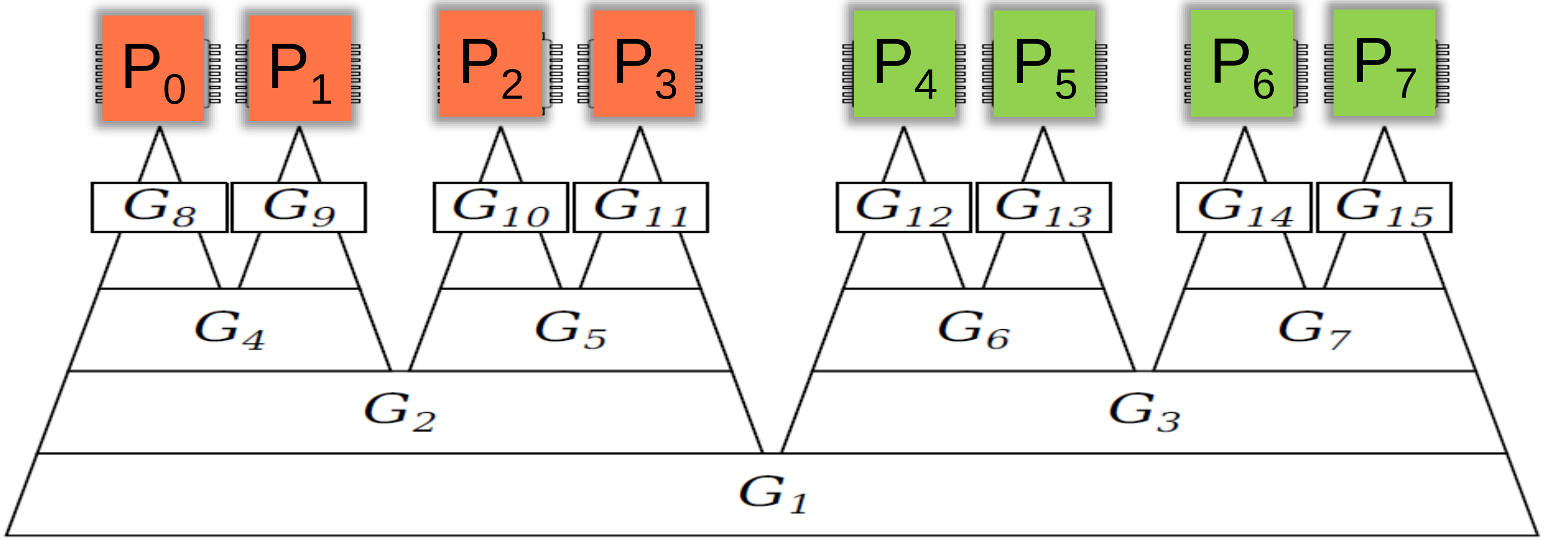
- Define a hierarchy of processor groups
  - **The Crown**
    - Example: A balanced binary crown over 8 cores



    - Tasks' core allocations must be whole groups (here, powers of 2)
    - Reduces #**possible mapping targets** (here, from 2^p-1 to 2p-1)

Idea: Constrain Resource Allocation

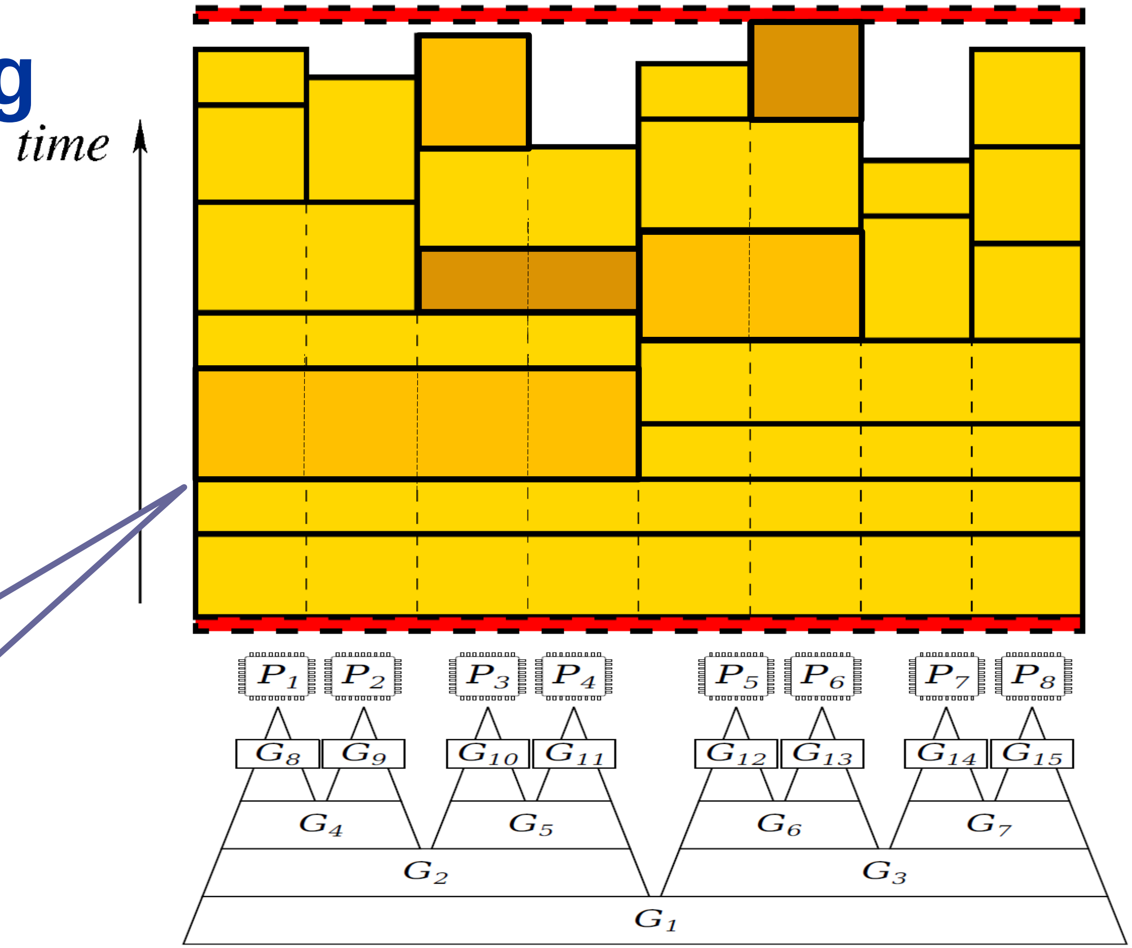
- Define a hierarchy of processor groups
  - **The Crown**
    - Example: A balanced binary crown over 8 cores



    - Tasks' core allocations must be whole groups (here, powers of 2), only root group is heterogeneous
    - Reduces #**possible mapping targets** (here, from 2^p-1 to $2p$-1)

Crown Scheduling

- Crown Allocation
- Crown Mapping
- Crown Scaling



Crown Schedule: one round of the steady state of the pipeline

- *Same order* and *non-increasing widths* of tasks executed "down-crown" on each core within the round
  - Minimizes global interferences for parallel task scaling
  - Only one barrier-effect at the beginning of a new round
  - Also, eases dynamic rescaling if tasks are dropped for a round

MILP Constraints for Crown Scheduling

- $x_{i,j,k}$ = 1 iff task j mapped to core group i at frequency level k
- Constraints:

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

Each task mapped exactly once

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

Maximum width not exceeded

- Calculate:
  - Time (G_l)

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

G_l = all groups containing core *l*

  - Energy

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

3 MILPs

for different optimization goals

- **Scenario 1:**
  Given makespan *M*,
  min *E*
- **Scenario 2:**
  Given E_{max},
  min makespan
- **Scenario 3:**
  Given avg. power P_{avg},
  min T_{max}

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
$\forall j: \ \sum_{i,k} x_{i,j,k} = 1$
$\forall j: \ \sum_{i:p_i > W_j} \sum_k x_{i,j,k} = 0$

3 Scheduling Approaches

- Task type aware, sequential tasks (**TAS**) – special case (W_j=1)
  = [Holmbacka, Keller 2017]
- Task type insensitive, parallel (**TIP**) – special case,
  = [Melot *et al*. 2015] adapted for heterogeneity
- Task type aware, parallel (**TAP**)
  (this work)

Experimental setup

- MILP solver: Gurobi 8.1 on AMD Ryzen 8 cores (16 HWT), 5 min. timeout per problem instance
- Synthetic task sets with 10, 20, 40, 80 tasks each, 10 instances each
- Parallelization efficiencies $e_j(1) = 1.0$, $e_j(2) = 0.9$, $e_j(4) = 0.86$
- Task max-widths (max. #cores) W_j chosen depending on task type:

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

- Deadline $$M = 0.6 \cdot \frac{\frac{\sum_j \tau_j}{p \cdot f_1} + \frac{\sum_j \tau_j}{p \cdot f_s}}{2}$$
- Power and time coefficients from [Holmbacka, Keller 2017]

Optimization Time / Feasibility

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

Results: Schedule Quality

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*, given makespan *M*

- TAP vs. **TAS**:
  - advantage TAP for small task sets (feasible schedule in any case),
  - tasks executed sequentially anyway 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

Results (on-line appendix)

RESULTS FOR SCENARIO 2, RELATIVE TO TAP

scheduling	task set card.	makespan	energy	#budget transgr.
TAS	10	1.301	1.081	
	20	1.015	1.009	
	40	1.001	0.999	
	80	1.000	1.000	
	total	1.068	1.019	
TIP	10	1.533	1.158	4
	20	1.429	1.190	3
	40	1.377	1.148	2
	80	1.412	1.212	2
	total	1.438	1.177	11

Scenario 2 (**min** makespan, given *E* budget):

- 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

Results (on-line appendix)

RESULTS FOR SCENARIO 3, RELATIVE TO TAP

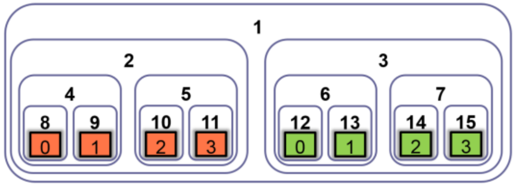
scheduling	task set card.	makespan	energy	av. power
TAS	10	1.155	1.077	0.945
	20	1.003	1.002	0.999
	40	1.000	1.000	1.000
	80	1.000	1.000	1.000
	total	1.039	1.020	0.986
TIP	10	1.369	1.059	0.787
	20	1.502	1.104	0.738
	40	1.507	1.183	0.791
	80	1.369	1.191	0.873
	total	1.437	1.134	0.797

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

- TAP still better than TAS 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

Summary

- Actor networks with **moldable tasks** on multi-/many-core architectures
- **Co-optimize** core allocation, mapping, DVFS
  - Optimize for **total energy**, given a throughput requirement, or **throughput** (round makespan) given an energy budget,
- **Crown-Scheduling**: group hierarchy, restricts core allocations
  - Integrated exact solution by ILP becomes feasible
  - In this work **generalized** for **heterogeneous** multicores a la big.LITTLE
- 3 optimization scenarios – 3 **MILP** models
- Experiments using big.LITTLE power profiles and time coefficients measured for different task types



- **Task-type-awareness helps**: improvements in both time and energy,
  up to 53% speedup,
  more feasible, fewer time-outs,¨faster opt.
- **Task parallelization helps**: Up to 8% energy TAP/TAS for smaller task sets
  Up to 30% time (TAP can avoid deadline violations
- **Future work**:
  - Modeling the communication cost between tasks
    - e.g., cache misses where mapped to different core types
    - Modeling dependencies for latency optimization
  - Real ARM board measurements for validation

APPENDIX

Abstract

Scheduling Moldable Parallel Streaming Tasks on Heterogeneous Platforms with Frequency Scaling

Sebastian Litzinger, Jörg Keller, Christoph Kessler

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%.

References

This work:

- Sebastian Litzinger, Jörg Keller, Christoph Kessler: Scheduling Moldable Parallel Streaming Tasks on Heterogeneous Platforms with Frequency Scaling. Proc. 27th European Signal Processing Conference (EUSIPCO 2019), A Coruna, Spain, Sep. 2019, IEEE. DOI: 10.23919/EUSIPCO.2019.8903180. On-line appendix: https://e.feu.de/ii

Previous work on Crown Scheduling:

- Christoph Kessler, Nicolas Melot, Patrick Eitschberger, Jörg Keller: **Crown Scheduling: Energy-Efficient Resource Allocation, Mapping and Discrete Frequency Scaling for Collections of Malleable Streaming Tasks.** In: Proc. 23rd Int. Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS 2013), Karlsruhe, Sept. 2013, pp. 215–222. © IEEE, 2013. DOI: 10.1109/PATMOS.2013.6662176
- Nicolas Melot, Christoph Kessler, Jörg Keller, Patrick Eitschberger: **Fast Crown Scheduling Heuristics for Energy-Efficient Mapping and Scaling of Moldable Streaming Tasks on Many-Core Systems.** *ACM Trans. on Architecture and Code Optimization (TACO)*, Vol. 11(4), Art. 62, Jan. 2015. DOI: 10.1145/2687653
- Nicolas Melot, Christoph Kessler, Jörg Keller: **Improving Energy-Efficiency of Static Schedules by Core Consolidation and Switching Off Unused Cores.** ParCo-2015 conference, Edinburgh, UK, 1-4 Sep. 2015. Published in: Gerhard R. Joubert, Hugh Leather, Mark Parsons, Frans Peters, Mark Sawyer (eds.): *Advances in Parallel Computing, Volume 27: Parallel Computing: On the Road to Exascale*, IOS Press, April 2016, pages 285-294. DOI 10.3233/978-1-61499-621-7-285.
- Nicolas Melot, Christoph Kessler, Jörg Keller, Patrick Eitschberger: **Co-optimizing Core Allocation, Mapping and DVFS in Streaming Programs with Moldable Tasks for Energy Efficient Execution on Manycore Architectures.** In: Proc. 19th International Conference on Application of Concurrency to System Design (ACSD-2019), Aachen, Germany, June 23-28, 2019. IEEE. DOI: 10.1109/ACSD.2019.00011

Previous work on task type aware scheduling of sequential streaming tasks on big.LITTLE:

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