Static Scheduling of Moldable Streaming Tasks with Task Fusion for Parallel Systems with DVFS

Christoph Kessler
Linköping University
Sweden

Sebastian Litzinger, Jörg Keller
FernUniversität in Hagen
Germany

ACM SIGBED International Conference on Embedded Software (EMSOFT'20)
ESWEEK, September 2020

LINKÖPING UNIVERSITY

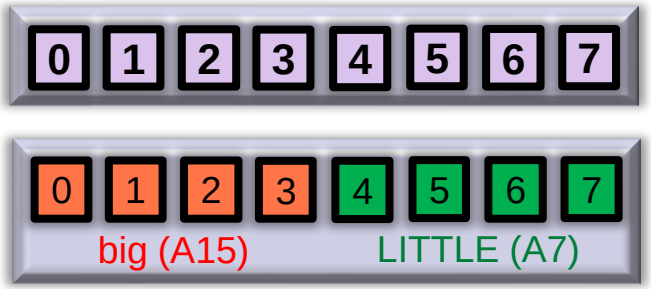
FernUniversität in Hagen

Motivation

- **Multi-core / Many-core CPUs**
  - Continued growth in #cores

- **Energy-efficient software**
  - Embedded system: Given application, dedicated hardware
  - **Static** configuration and mapping
    - Leverage both **inter-task** and **intra-task parallelism**
    - + **DVFS** + **Heterogeneity** + ...
    - In this work: ... + **Task fusion**

Hardware Model:
Generic Multicore CPU with DVFS



Resources

- *p* cores $P_0, \ldots, P_{p-1}$
- Can be heterogeneous

Discrete Dynamic Voltage and Frequency Scaling (DVFS)

- *s* discrete DVFS levels, ordered by frequency $F = \{ f_1 = f_{min}, \ldots, f_s = f_{max} \}$
  - Includes voltage scaling by auto-co-scaling
  - Example: ARM Cortex A15: 0.6, 0.8, 1.0, 1.2, 1.4, 1.6 GHz
- Can change DVFS level dynamically at task switching
- Ideal: **Core-wise DVFS**
- **DVFS islands**: Groups of several cores that share a DVFS regulator
  - cores in a DVFS island will run at same DVFS level at the same time

Power model:

- Power values $P(f_k)$ measured on target for each DVFS level *k* (and task type)
  - e.g. for A15, A7 in [Holmbacka, Keller 2017]

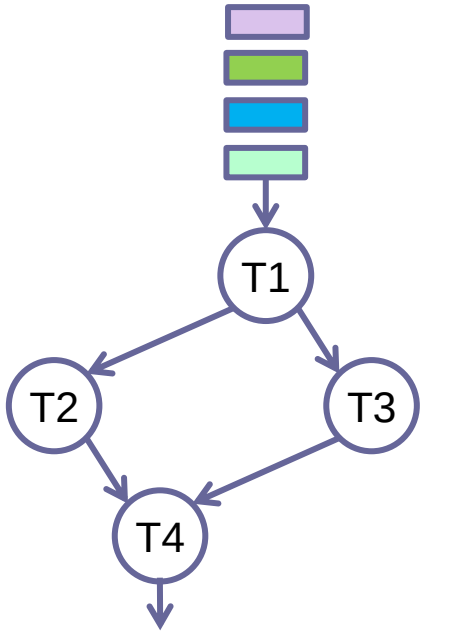
Software Model:
Streaming Computations

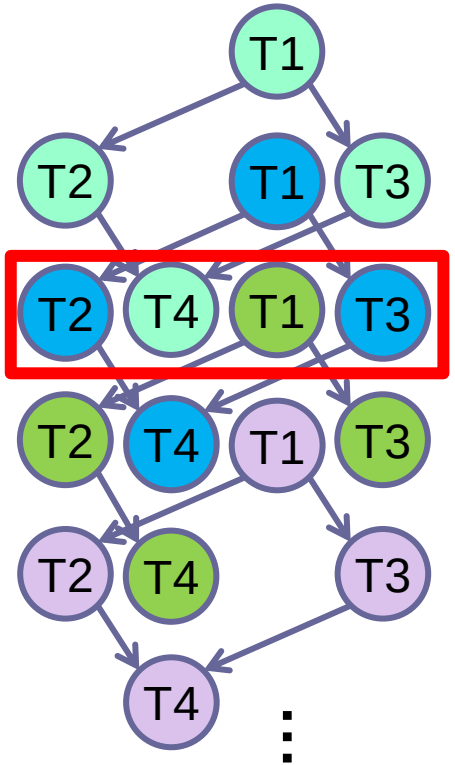
Streaming task graph (= actor network)

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

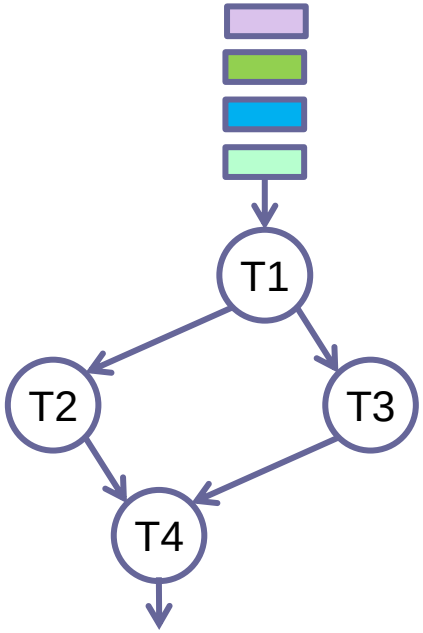
Software pipelining

- Steady state





Software Model:
Streaming Computations

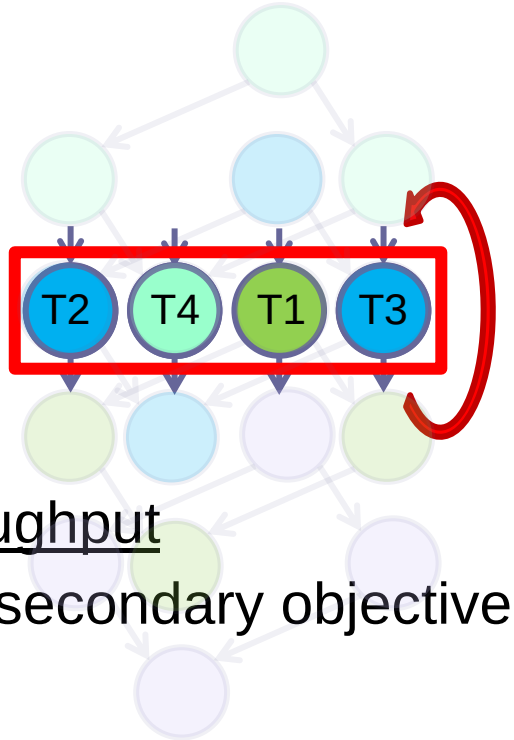


Streaming task graph (= actor network)

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

Software pipelining

- Steady state



One round in steady state:

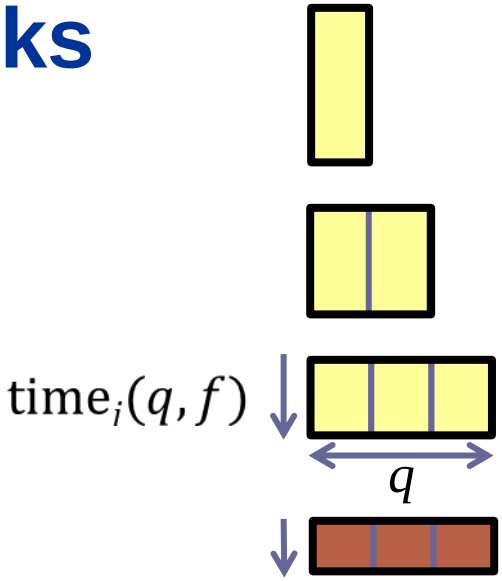
- Independent streaming task instances
- Balancing workload over all cores
  → minimizes makespan → maximizes throughput
- Reducing single-packet latency might be a secondary objective

Moldable (Parallelizable) Tasks

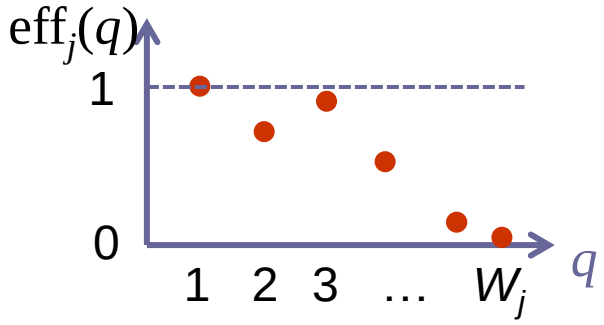
Moldable tasks $j = 1,\ldots,n$

- A task j performs fixed **work** workload_j (per round)
- Can be run with (integer) $q \geq 1$ cores (fixed before the task starts)
  - internally using a **parallel algorithm**
- Arbitrary scalability behavior model:

  parallel efficiency $0 < \text{eff}_j(q) \leq 1$
  - predicted or measured on target system for $1 \leq q \leq W_j$



$$\text{time}_j(q, f) = \frac{\text{workload}_j}{f \cdot q \cdot \text{eff}_j(q)}$$



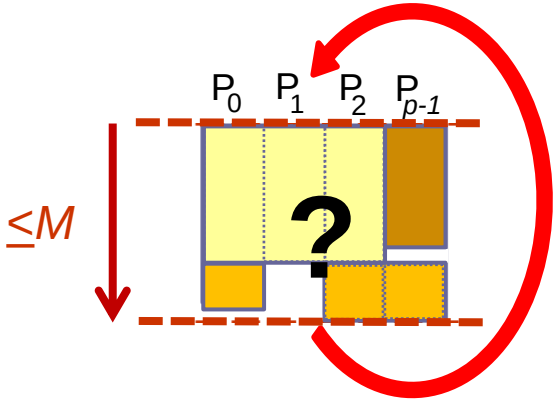
Mixed task model

- Inherently sequential: max. parallelism $W_j = 1$
- Moldable, limited scalability: fixed $W_j > 1$
- Moldable, unlimited scalability: $W_j \geq p$

Scheduling for the Steady-State

Multiple interdependent subproblems to solve (off-line):

- **Core allocation**
  for each moldable task *j*
- **Mapping**
  - Map each task *j* to *specific* core(s)
- **DVFS level selection**
  - Select for each task *j* a frequency level in $\{f_1, \ldots, f_s\}$

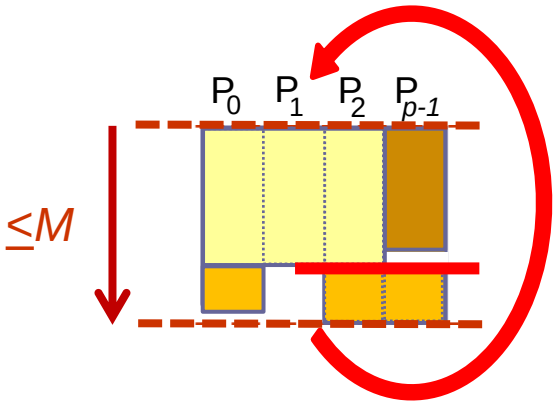


given a **throughput constraint** (round makespan),

to minimize overall energy usage.

Problem – Complexity!

- Combinatorial explosion of options if unrestricted
  - ILP [Melot *et al*. ACSD'19] (for allocation+mapping+scaling)

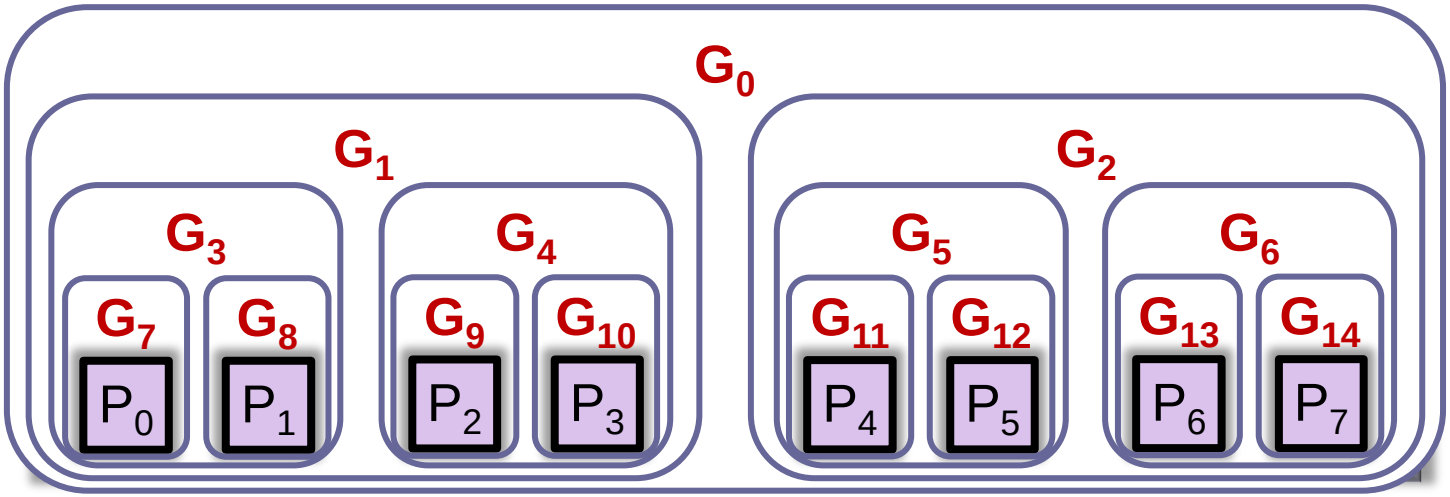


- A *parallel task* should start on all its assigned cores simultaneously
- Scale frequency on all cores of a parallel task equally

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

Idea: Constrain Resource Allocation

- Define a hierarchy of processor groups
  - **The Crown** [Kessler *et al*. 2013]
    - Example: A balanced binary crown over 8 cores

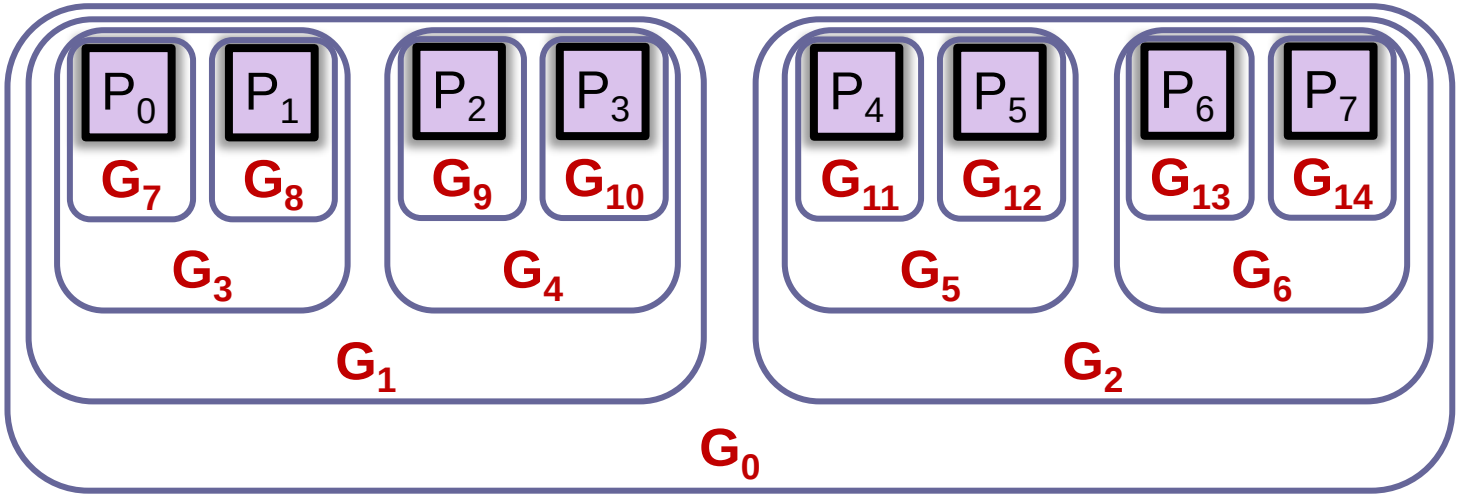


- Tasks' core allocations must be whole groups (here, powers of 2)
- Reduces #**possible mapping targets** (here, from 2^p-1 to $2p$-1)

Why "Crown"?

Idea: Constrain Resource Allocation

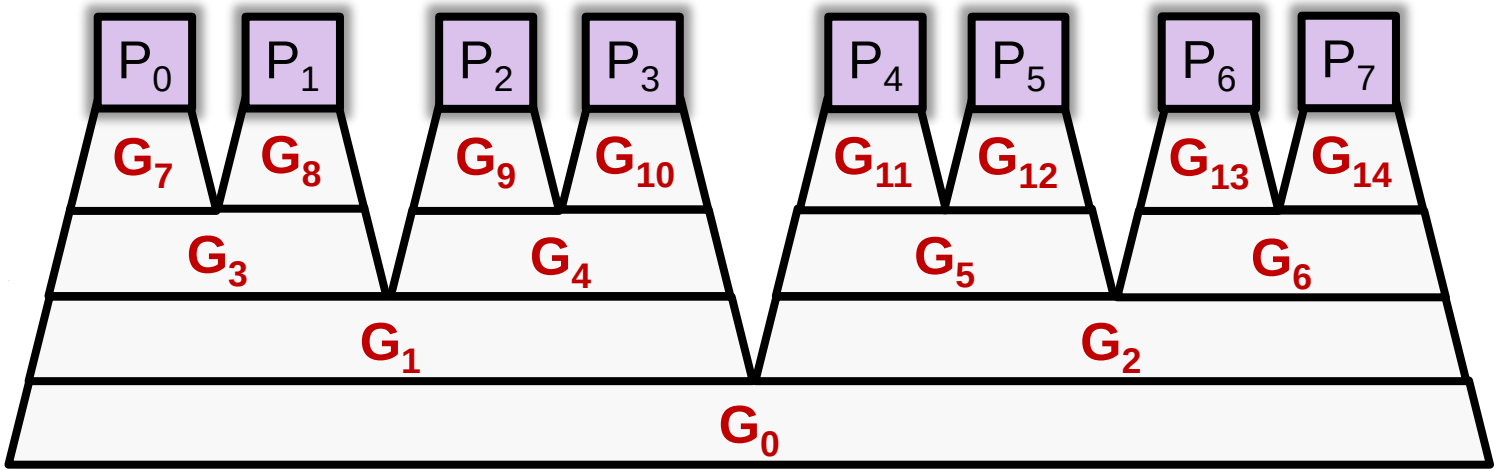
- Define a hierarchy of processor groups
  - **The Crown** [Kessler *et al*. 2013]
    - Example: A balanced binary crown over 8 cores



- 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

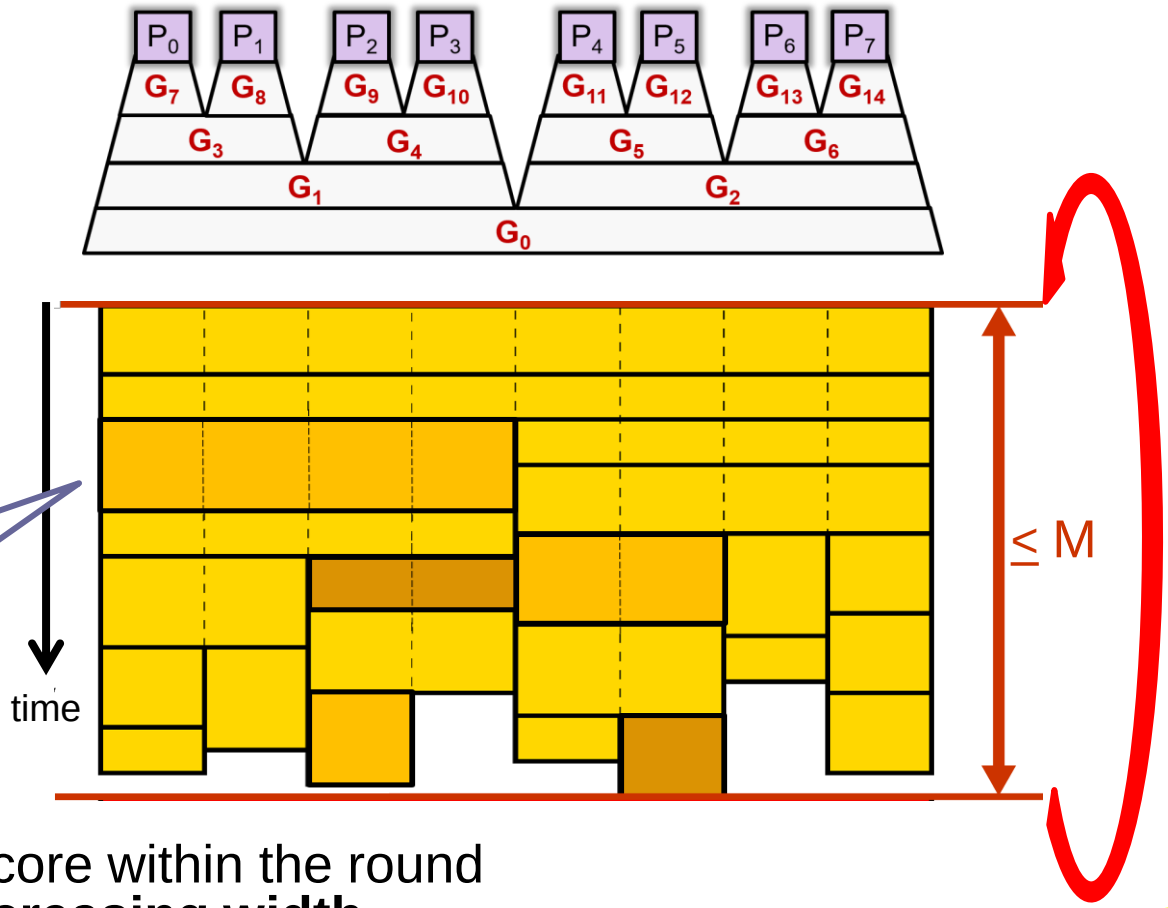
- Define a hierarchy of processor groups
  - **The Crown** [Kessler *et al*. 2013]
    - Example: A balanced binary crown over 8 cores



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

Crown Schedules

- Crown-restricted
  - Allocation
  - Mapping
  - Scaling



Crown Schedule for one round of the steady state

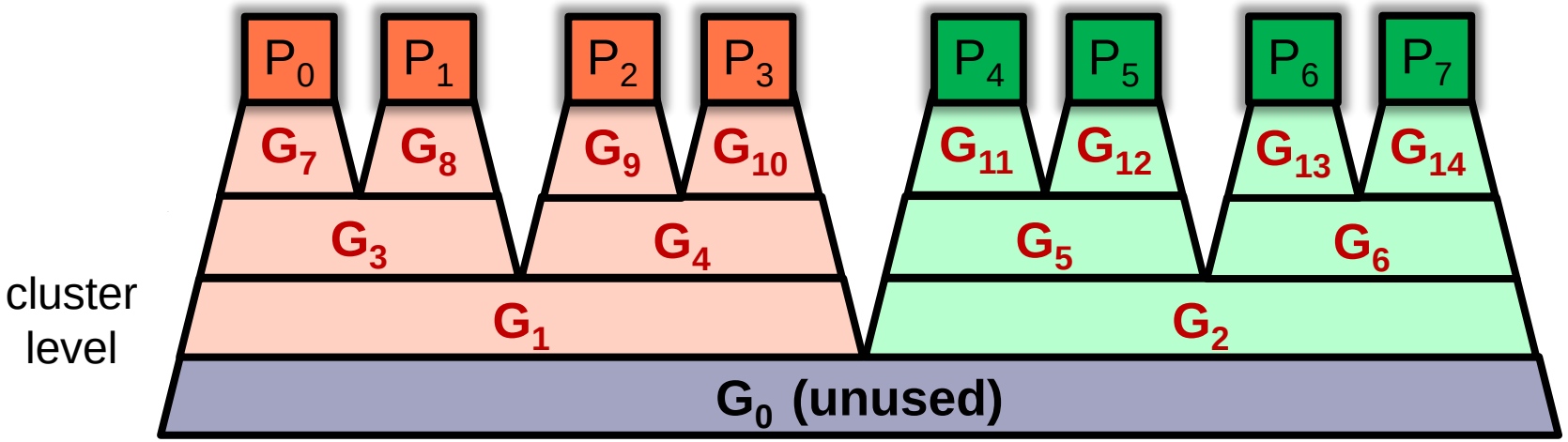
- Tasks executed on each core within the round in **same order** by **non-increasing width**
  - Within a group keep a topological order
  - Reduces global interferences e.g. in scaling tasks
  - Idle times only at end of a round

Exact solution by ILP feasible [Melot *et al.* TACO'15]

Generalization: Heterogeneous Crown

- Example: ARM big.LITTLE CPU
- Different core types in different subcrowns

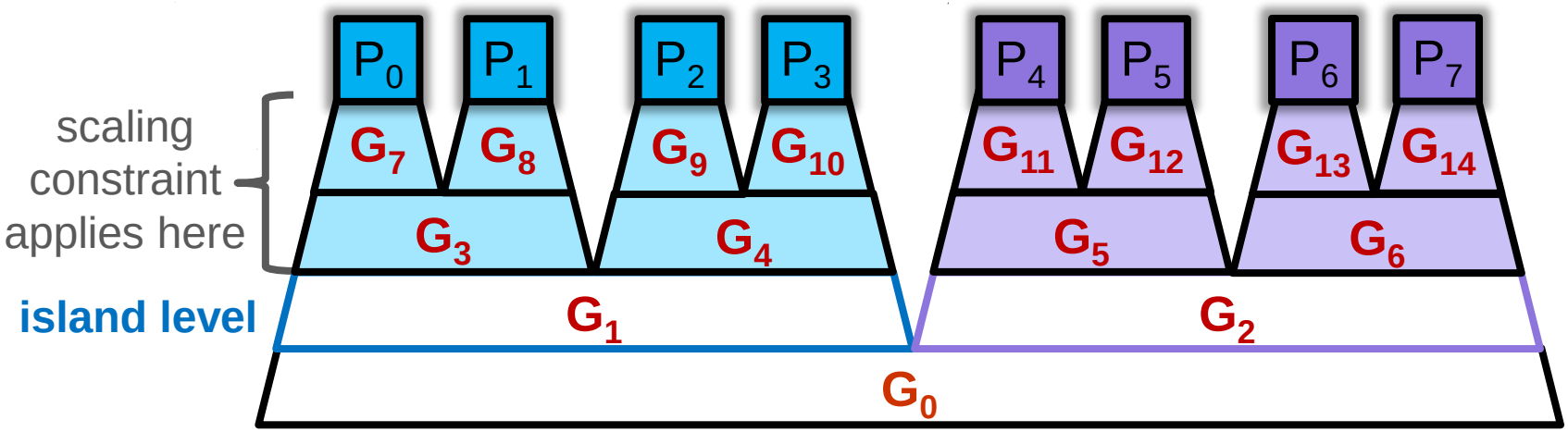
  [Litzinger *et al*. EUSIPCO'19]



Generalization: Crown with DVFS Islands

- Example:
  CPU with 2 DVFS islands of 4 cores each

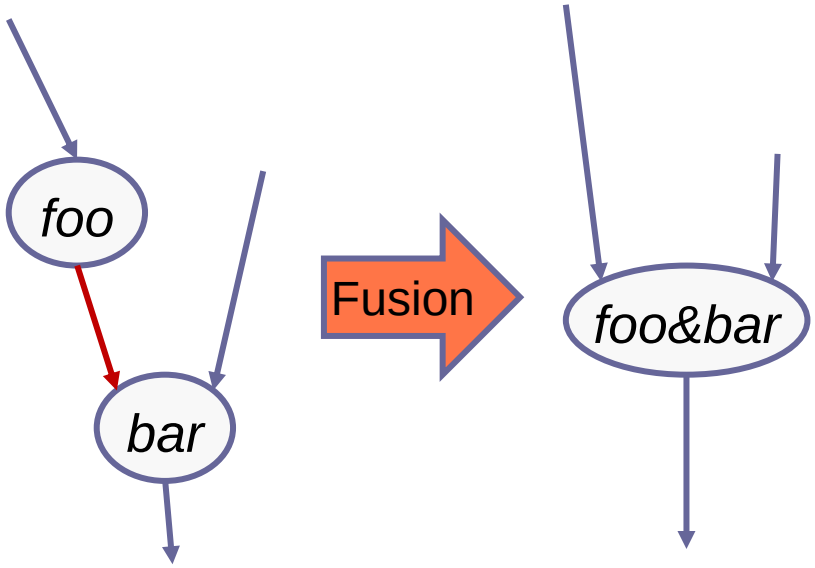
- Crown structure aligned with DVFS islands [Melot *et al*. PDP'20]
- ILP constraint on DVFS level selection
  - All tasks with allocations < island size mapped to cores of the same island need to run at *same* DVFS level



Task Fusion

Here, for groups of **dependent tasks**

- ☺ Fusion internalizes data flow
  - ☺ reduces communication overhead
- ☺ Writes and reads of intermediate data get closer to each other in time
  - ☺ data locality, cache hits, use of registers or core-local memory
  - ☺ need less temporary storage
- ☺ May even use a different algorithm



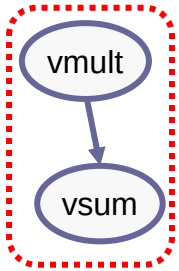
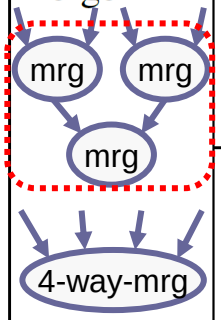
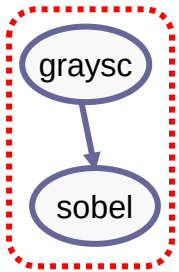
Fusion Gains for Microbenchmarks

- Times measured on Odroid XU+E board ARM **big.LITTLE** (Cortex A15, A7) CPU

Fusion speedup ca. **15% up to and beyond 2x**

fscale: inverse speedup – lower is better

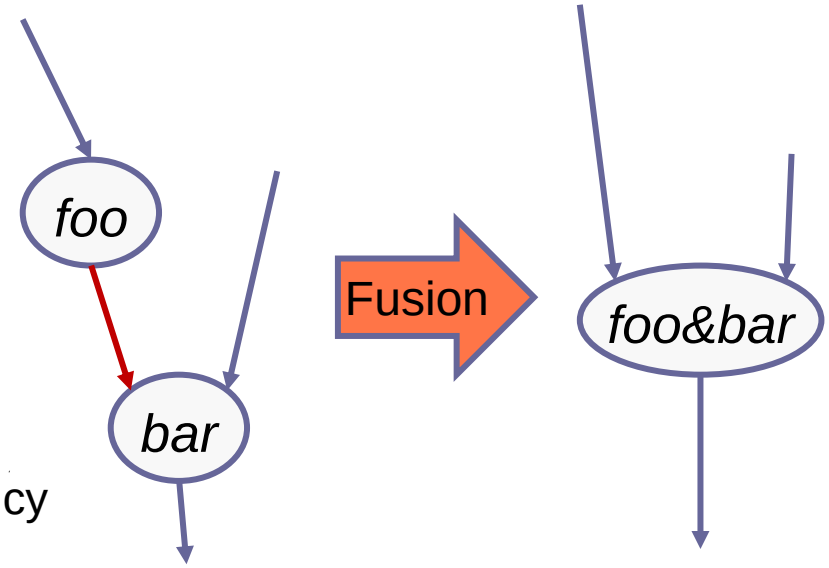
- Similarly, also for **energy** savings
- Similarly, also on a **Xeon** E5-2620 CPU

Benchmark	Core type	Nr. of cores	Avg. fscale	Avg. fused par. time rel. to seq.	Avg. non-fused time rel. to seq.
dotproduct 	LITTLE	1	0.636		
		2	0.640	0.521	0.518
		3	0.639	0.362	0.361
		4	0.624	0.308	0.315
	big	1	0.411		
		2	0.438	0.566	0.529
		3	0.450	0.414	0.377
		4	0.455	0.407	0.370
merge 	LITTLE	1	0.457		
		2	0.456	1.001	1.002
		3	0.456	1.002	1.003
		4	0.488	1.083	1.015
	big	1	0.455		
		2	0.452	0.995	1.002
		3	0.451	1.000	1.008
		4	0.452	1.015	1.021
graysobel 	LITTLE	1	0.864		
		2	0.869	0.508	0.506
		3	0.868	0.344	0.343
		4	0.865	0.262	0.262
	big	1	0.820		
		2	0.826	0.510	0.507
		3	0.824	0.344	0.343
		4	0.827	0.268	0.265

Task Fusion

Here, for groups of **dependent tasks**

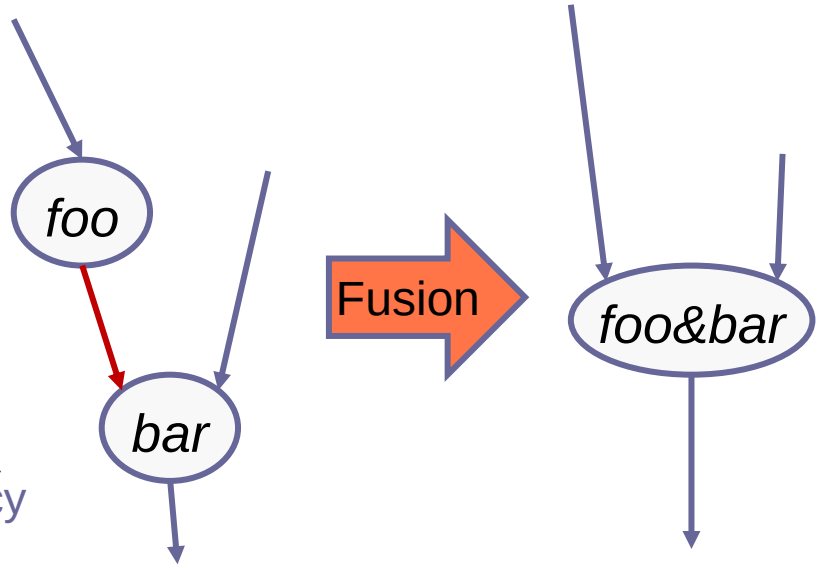
- ☺ Fusion internalizes data flow
  - ☺ reduces communication overhead
- ☺ Writes and reads of intermediate data get closer to each other in time
  - ☺ data locality, cache hits, use of registers or core-local memory
  - ☺ need less temporary storage
- ☺ May even use a different algorithm
- ☺ Might benefit both throughput, energy and latency



Task Fusion

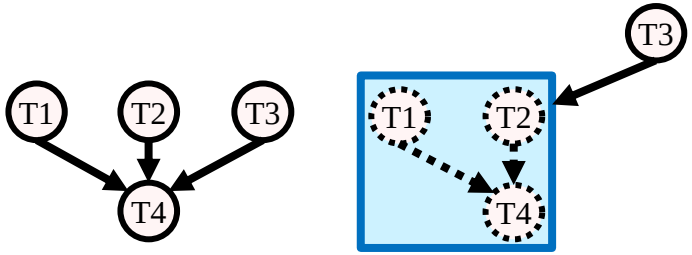
Here, for groups of **dependent tasks**

- ☺ Fusion internalizes data flow
  - ☺ reduces communication overhead
- ☺ Writes and reads of intermediate data get closer to each other in time
  - ☺ data locality, cache hits, use of registers or core-local memory
  - ☺ need less temporary storage
- ☺ May even use a different algorithm
- ☺ Might benefit both throughput/energy and latency



BUT:

- ☹ Loss of pipeline (inter-task) parallelism
- ☹ Higher stress on mapping and allocation
- ☹ Constrained max-parallelism of fused task
- ☹ Might also *increase* critical path length for latency

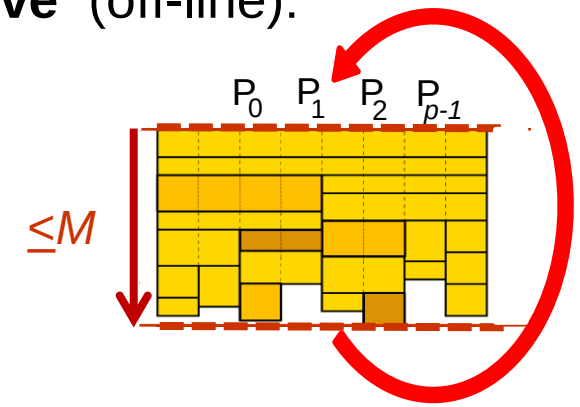


→ Fusion should be decided **together** with allocation, mapping and scaling!

Crown-Scheduling for the Steady-State With Fusion

Multiple interdependent subproblems to solve (off-line):

- **Crown allocation** and
- **Crown mapping**
  - Map each task *j* to *some group*



- **DVFS level selection**
  - Select for each task *j* a frequency level in $\{f_1, \ldots, f_s\}$
- **Task fusion**
  - Merge dependent tasks or not

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

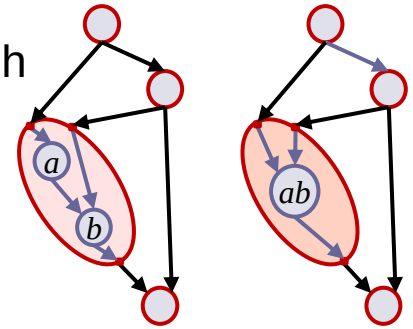
to minimize overall energy usage.

Two Methods – Details in the paper ...

- **Fully integrated Crown Scheduling including Task Fusion**
  - Mixed ILP (MILP) model
    - Energy-optimal fusion + core allocation + mapping + scaling,
      - given throughput goal (maximum makespan)
    - maximum latency parameter (→ Pareto-optimization)
    - Homogeneous / heterogeneous multicore platform, DVFS islands
    - Constraints to avoid that fusion creates artificial cycles in the task graph
  - Solution is *not* constrained by a detached pre-fusion phase.

- **Two-step approach**
  - **Step 1: Pre-fusion**
    - Inductively generate promising fusion candidates
    - Heuristic pruning of generated options
    - then greedy fusion
  - **Step 2:** classical Crown Scheduling (ILP) on pre-fused task graph

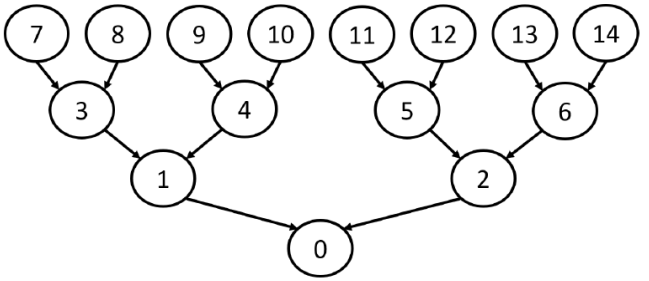
- **Generalization** (→ paper)**:**
  Crown Scheduling ILP on a ***hierarchical multi-variant task graph***
    - **supertasks** with **variants** (here: fused, nonfused)

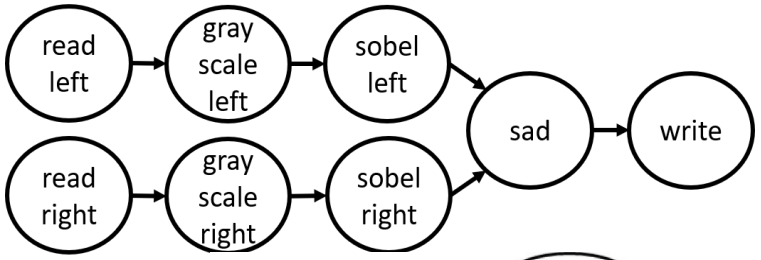


Experimental Evaluation

Target Platform

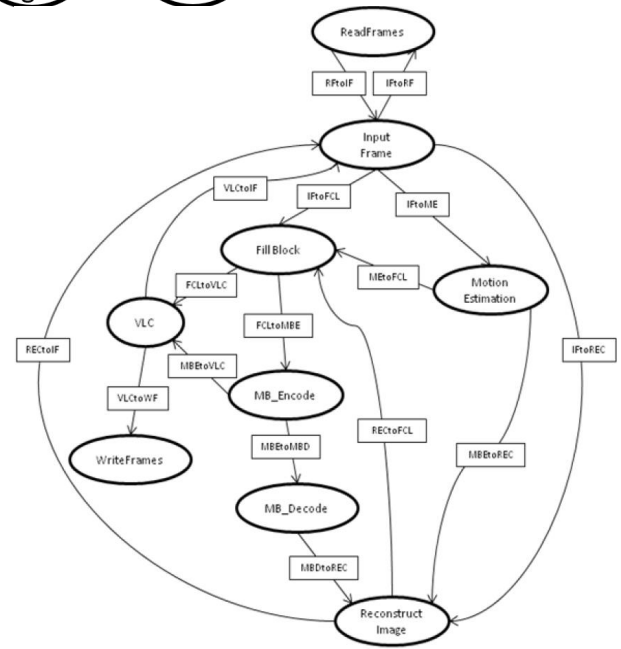
- Heterogeneous: big.LITTLE
- Homogeneous: A15 cluster





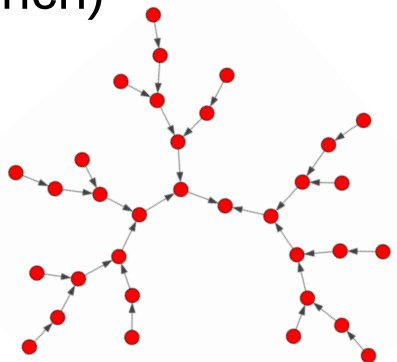
Applications

- Parallel mergesort: binary merger trees
- SDE stereo depth estimation
  - 1x1 (8), 2x2 (32), 3x3 (72) partitioning
- H.263 encoder (DFbench)



Synthetic Task Graphs

- Irregular topology; scaling up



Pareto-Optimization (Energy, Latency)

- **Baseline (1.0):** Two-step method: **Pre-fusion + Crown scheduling ILP**
- **Crown scheduling with fully integrated fusion**
  - Iterate Max-Latency parameter in MILP from upper bound until infeasible

task set	start lat.	lowest lat.	rel. E at latency reduction by 0	1	2	3	4
integrated scheduling and task fusion							
H.263	7	3	0.998	0.998	0.998	1.030	1.066
SDE 1x1	5	2	0.560	0.560	0.560	0.599	–
SDE 2x2	5	3	0.989	0.989	0.989	–	–
SDE 3x3	5	3	0.989	0.989	0.989	–	–
merge 7	3	2	1.000	1.000	–	–	–
merge 15	4	1	1.000	1.000	1.000	1.919	–
merge 31	5	2	0.996	1.000	1.000	1.000	–

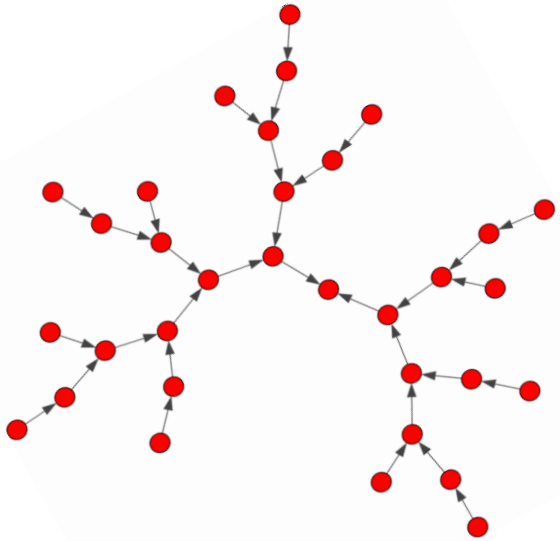
Trade-off: higher energy for lowest possible latency

Where > 1, fusion-integrated CS MILP did not find an optimal solution and the two-step heuristic was better.

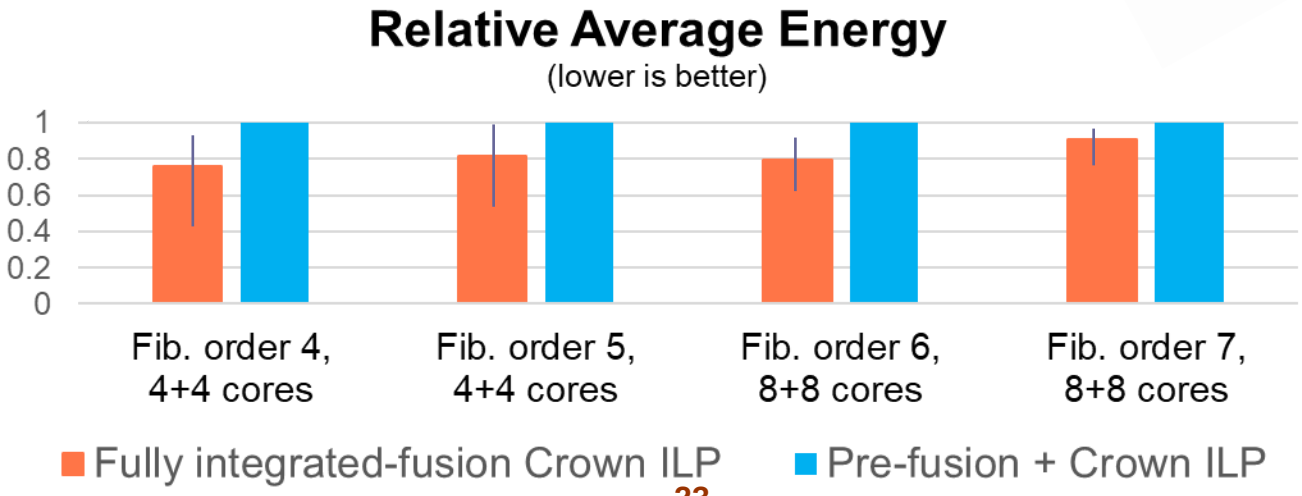
- Further results (**Crown scheduling with supertasks)** in the paper

Scaling Up: Fully Integrated vs. Two-Step Method

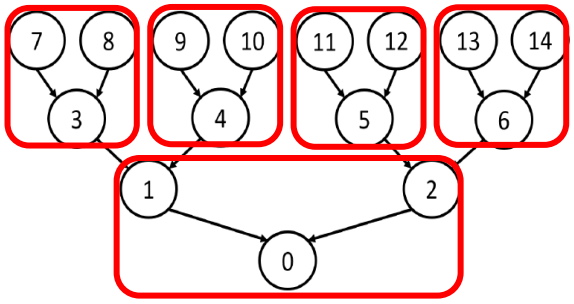
- 200 random task graphs
  - based on Fibonacci trees of order 4, 5, 6, 7
- Target: generic heterogeneous multicore CPU based on big.LITTLE with 4+4, 8+8 cores



- **Fully integrated fusion** Crown scheduler leads to **10% to 25% lower energy** on average compared to two-step pre-fusion + crown scheduling
  - Two-step method gets closer for larger task sets and core numbers



Real Overall Gains
in Throughput, Energy



- **Mergesort** application, 4-level merger tree
  - Integrated scheduler: Use 2 fully fused levels of 4-way mergers
- Runtime system for the A15 cluster of XU+E big.LITTLE

- Predicted vs. measured time and energy, averaged over 100 rounds:

	makespan (s)	E (J)
fused predicted	22.56	32.78
nonfused predicted	23.88	86.60
real fused average	22.58	35.39
real fused std. dev.	0.17	0.30
real nonfused average	24.22	85.12
real nonfused std. dev.	0.03	0.45
real fused vs. predicted	0.99	1.08
real nonfused vs. predicted	1.01	0.98
real fused vs. real nonfused	**0.93**	**0.42**

scheduler's runtime prediction within 2% of real

Throughput gain 7.5%

Energy savings 58%

Summary

- Streaming task graphs with **moldable tasks**
- **Static** scheduling to multi-/many-core CPUs with discrete DVFS
  - Minimize **total energy**, given a throughput requirement
  - Packet **latency** as secondary objective
- **Co-optimize** core allocation, mapping, DVFS, **task fusion**
- **Crown-Scheduling**: group hierarchy, restricts core allocations+mapping
  - Integrated exact solution by ILP becomes feasible
  - Can model DVFS islands and heterogeneous multicore
- **Fully integrating task fusion in Crown scheduling MILP**
  - Pareto-optimization (energy, latency)
- **Two-step method**: Heuristic pre-fusion + ILP crown scheduler (with variants)
- **Evaluation** on big.LITTLE based target platforms (model + real)
  - Using synthetic and realistic task graphs / applications
  - Crown scheduler better than alternative static scheduler (Xu *et al.*'12), see paper
  - MILP / ILP methods *are* computationally feasible
    - Pre-fusion + crown ILP with variants generally < 0.3s (Gurobi 8.1.0)
    - Fully integrated crown scheduler needs typ. 10..100x longer, yet << Xu-scheduler
  - Fully integrated task fusion: less energy than with two-step method
  - Can trade energy for latency improvements

Static Scheduling of Moldable Streaming Tasks with Task Fusion for Parallel Systems with DVFS

Christoph Kessler
Linköping University
Sweden

Sebastian Litzinger, Jörg Keller
FernUniversität in Hagen
Germany

ACM SIGBED International Conference on Embedded Software (EMSOFT'20)
ESWEEK, September 2020

Thanks for listening!
See the paper for details

LINKÖPING UNIVERSITY

FernUniversität in Hagen

Questions?

APPENDIX

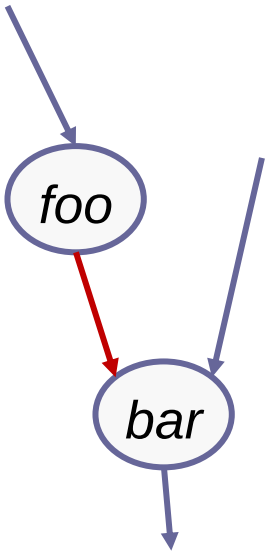
Backup Slides, Abstracts,
References to previous work

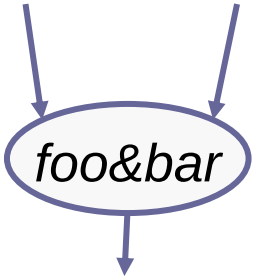
Task Fusion Example

```
// producer task:
... read in1[0:B-1] from buffer
for (i=0; i<B; i++)
  out1[i] = foo(in1[i],...);
... write out1[0:B-1] to buffer
```

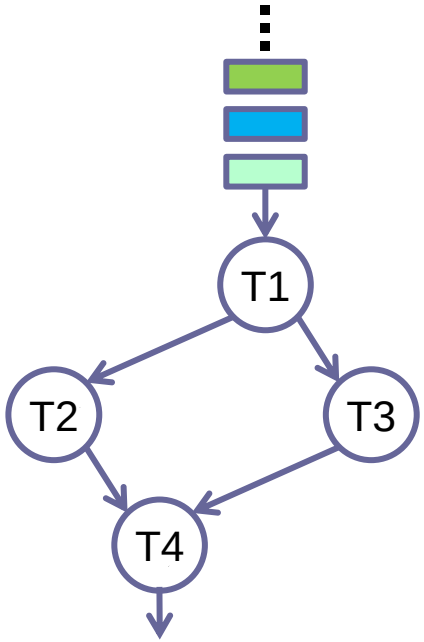
```
// consumer task:
...read out1[0:B-1] from buffer
for (i=0; i<B; i++)
  out2[i] = bar(out1[i],...);
...write out2[0:B-1] to buffer
```


```
// fused task:
... read in1[0:B-1] from buffer
for (i=0; i<B; i++)
  out2[i] = bar(foo(in1[i],..).);
... write out2[0:B-1] to buffer
```

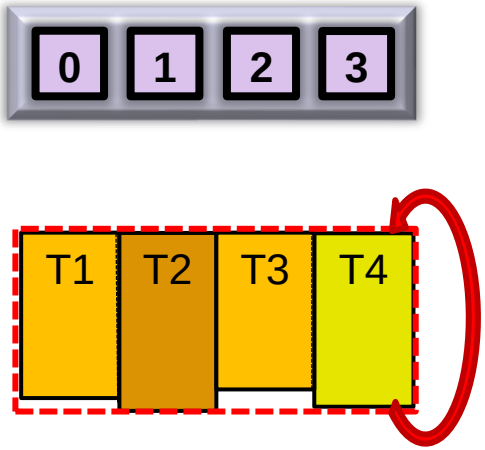




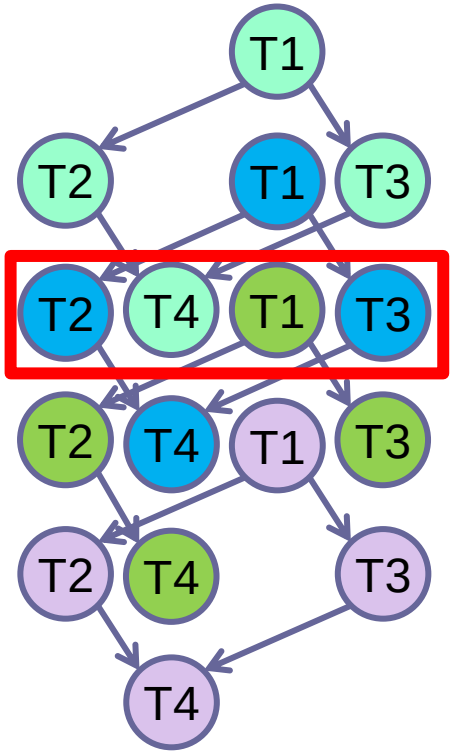
Example
Packet latency depends on mapping (1)



Streaming task graph

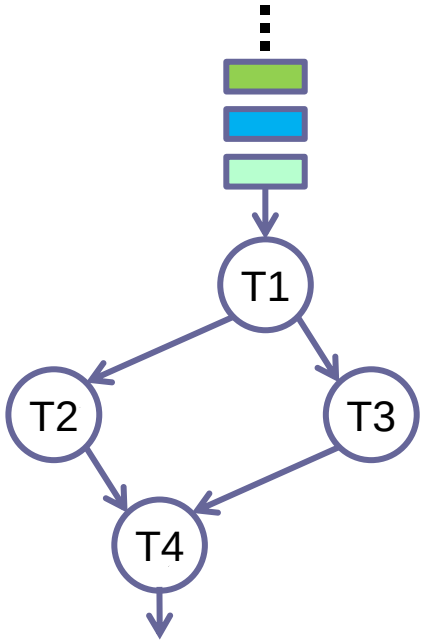


A schedule for 4 cores

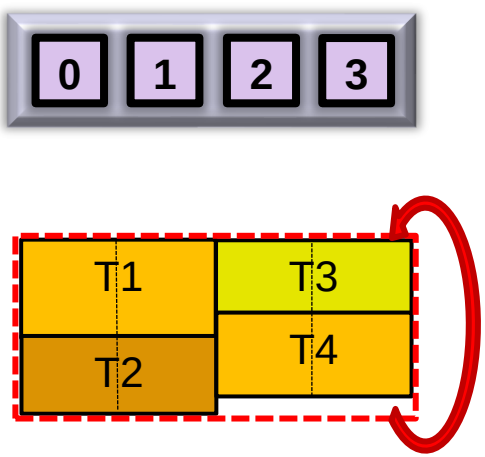


Need **3 rounds** to process an input packet completely

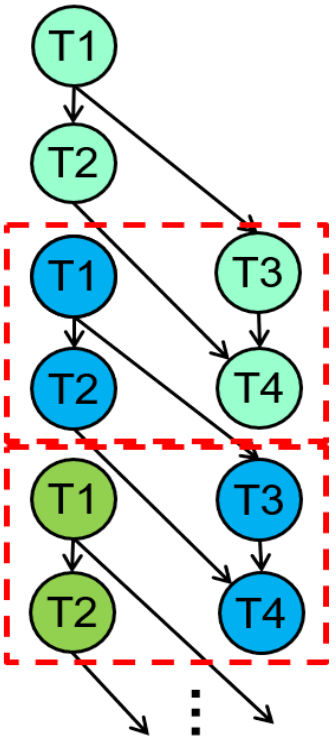
Example
Packet latency depends on mapping (2)



Streaming task graph

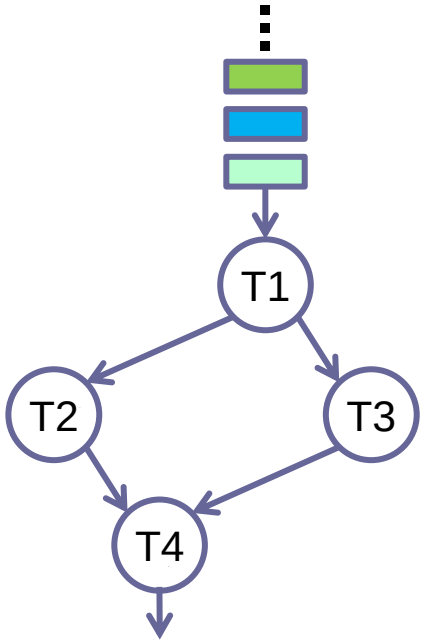


A schedule for 4 cores

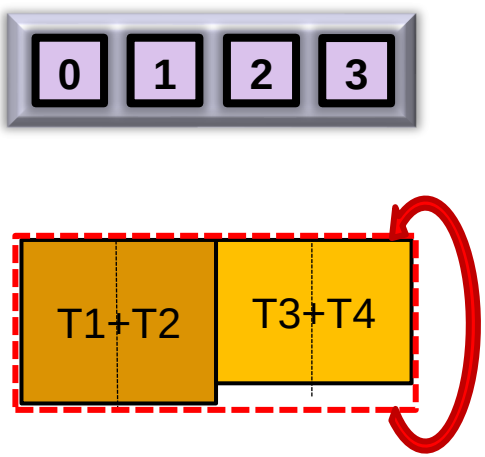


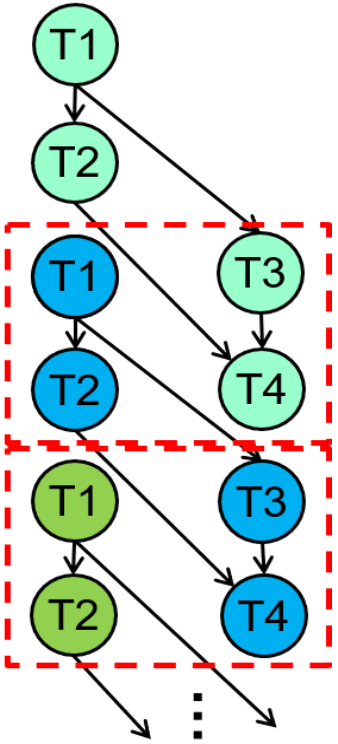
Need **2 rounds** to process an input packet completely

Example
Packet latency depends on mapping (3)



Streaming task graph

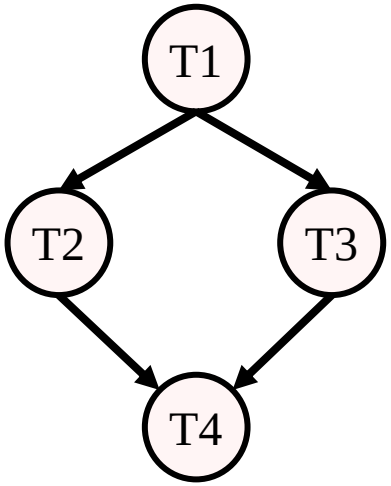


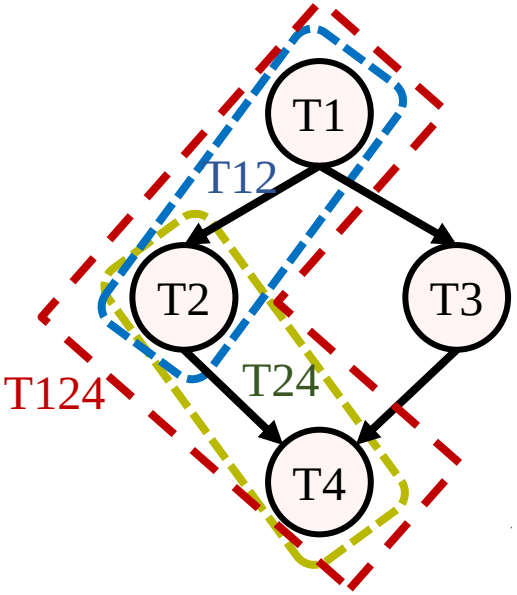


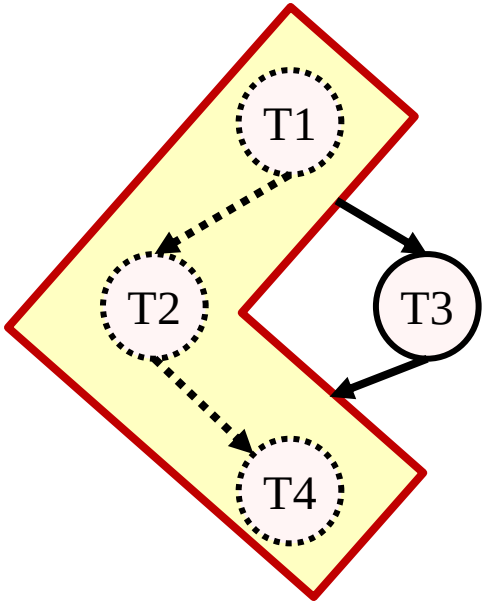
A schedule for 4 cores
similarly, after fusing T1+T2, T3+T4

Need **2 rounds** to process an input packet completely

Avoiding Artificial Cycles by Fusion







T1+T2+T4 **not** fusable
– creates artificial cycle

Crown ILP vs. Xu ILP schedulers
Effect of Fusion on Energy and Latency

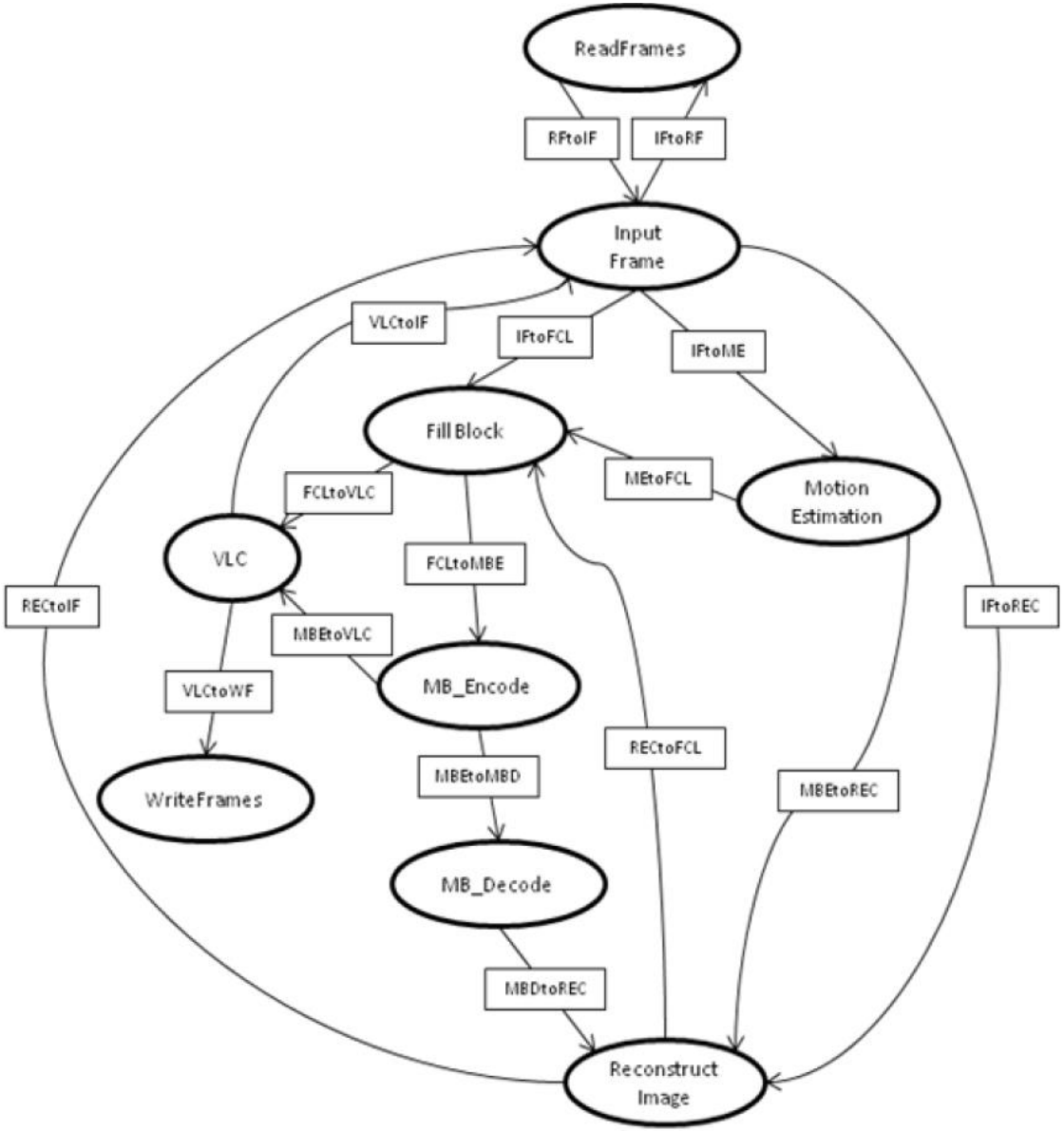
- For homogeneous 4-core (A15) target
- **Baseline (1.0):** Pre-fusion heuristic + regular Crown Scheduler
  - Lower is better
- **Regular crown:** No fusion + Regular Crown scheduler
- **Xu:** No fusion + Energy-optimizing moldable task scheduler by Xu *et al*. '12
- **Pre-fused Xu**: Pre-fusion heuristic + scheduler by Xu et al.
- **Crown scheduler outperforms Xu scheduler on energy (... and on scheduling time)**

task set	regular crown		regular Xu		pre-fused Xu	
	E	lat.	E	lat.	E	lat.
H.263 encoder	1.00	1.00	1.60	0.83	1.60	0.83
SDE (1 × 1)	0.63	1.00	0.74	1.00	–	–
SDE (2 × 2)	1.05	1.67	1.05	1.33	1.00	1.00
SDE (3 × 3)	1.05	1.67	1.05	1.33	1.00	1.33
mergesort (7)	1.59	1.50	2.24	1.50	1.22	1.00
mergesort (15)	2.46	2.00	2.97	2.00	1.17	1.00
mergesort (31)	1.94	1.67	2.29	1.67	1.11	1.00
average	1.39	1.50	1.71	1.38	1.18	1.03

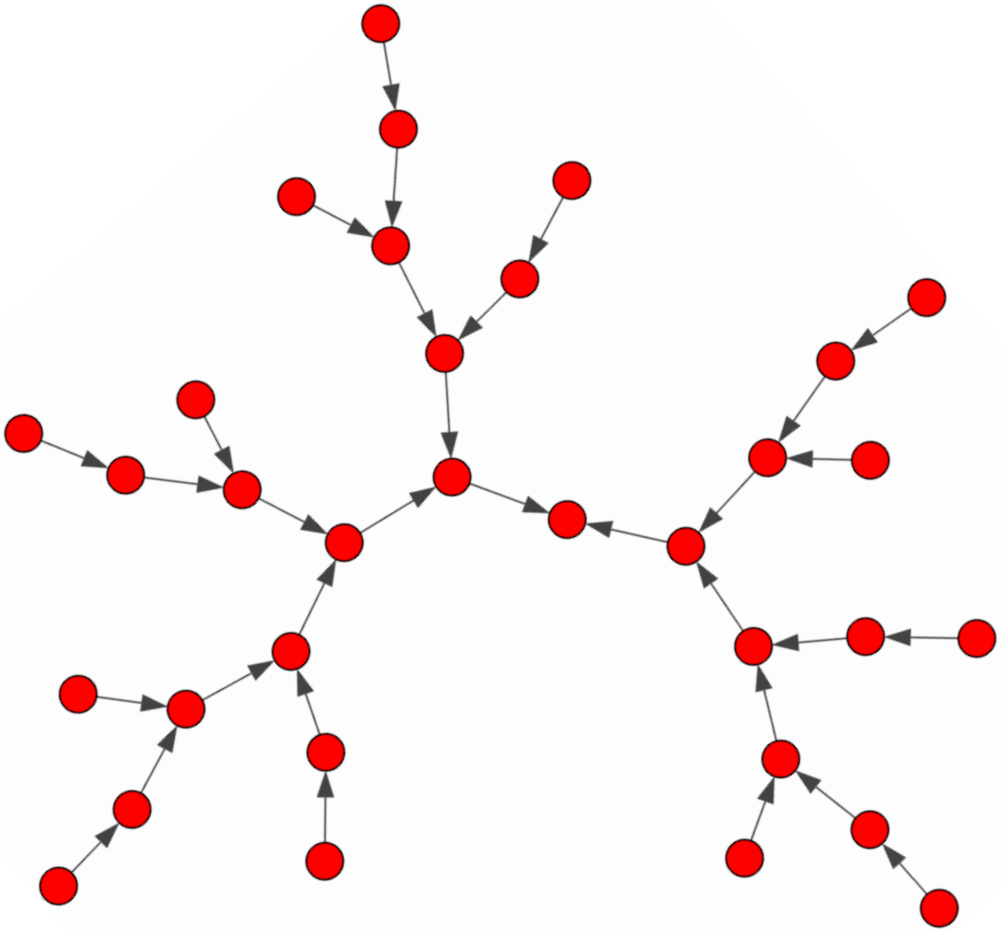
Higher energy for SDE 1x1: Pre-fusion too aggressive, too few tasks remain; schedule becomes sequential at high DVFS level

Xu scheduler (ILP) with pre-fusion does not find a feasible solution

H.263 Encoder from DFbench



Fibonacci Tree of Order 7



Abstract

Static Scheduling of Moldable Streaming Tasks With Task Fusion for Parallel Systems With DVFS

Christoph Kessler, Sebastian Litzinger, Jörg Keller

Abstract: We consider the problem of statically scheduling a task graph of moldable streaming tasks (i.e., the actor network) to a multicore or many-core CPU with discrete dynamic voltage and frequency scaling (DVFS). We employ an integer linear programming (ILP) approach that combines allocating cores to tasks, mapping tasks to core subsets, selecting a DVFS level for each task, and considering all options for task fusion as provided by a cost model, given data throughput and latency requirements and targeting low energy consumption. We also propose a partly decoupled approach that applies greedy prefusion before running an ILP-based scheduler considering the other three subproblems together. We use microbenchmarking on an ARM big.LITTLE architecture to quantify the advantage of task fusion in the above setting, and evaluate the use of task fusion in terms of energy savings, latency improvement, and scheduling time for three real-world applications. We confirm the scheduling results by running the applications with and without task fusions on the ARM big.LITTLE. Results indicate that streaming applications can profit from task fusion, as we achieve a significant reduction of energy consumption in most cases, while scheduling time is only moderately increased.

Referenced Previous Work

This work:

- Christoph Kessler, Sebastian Litzinger, Jörg Keller: **Static Scheduling of Moldable Streaming Tasks with Task Fusion for Parallel Systems with DVFS.** *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, vol. 39 no. 11, S.I. on ESWEEK 2020. DOI: 10.1109/TCAD.2020.3013054

Our 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, Apr. 2016, pp. 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
- 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
- Nicolas Melot, Christoph Kessler, Jörg Keller: **Voltage Island-Aware Energy-Efficient Scheduling of Parallel Streaming Tasks on Many-Core CPUs..** Proc. 28th Euromicro Conference on Parallel, Disributed and Network-Based Processing (PDP'20), Västerås, Sweden, March 2020. IEEE, 2020. DOI: 10.1109/PDP50117.2020.00030

Our 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

The Xu *et al*.'12 ILP scheduler used for comparison:

- H. Xu, F. Kong, Q. Deng: **Energy Minimizing for Parallel Real-Time Tasks Based on Level-Packing**. Proc. 18th Int. Conf. on Emb. and Real-Time Comput. Syst. and Appl., 2012, pp. 98–103.