Adaptive crown scheduling for streaming tasks on many-core systems with discrete DVFS

C. Kessler, S. Litzinger, J. Keller

AutoDASP @ Euro-Par 2019

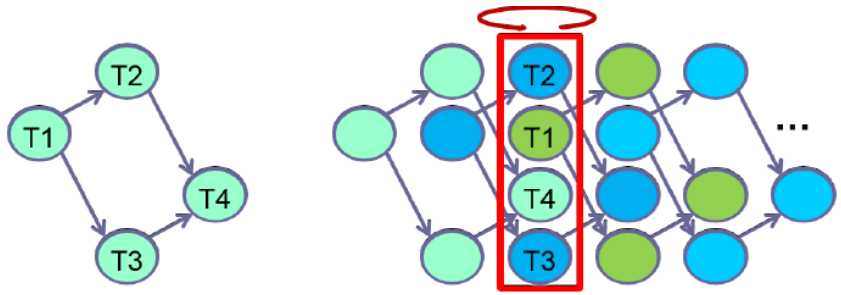
Parallelism and VLSI Research Group
Sebastian Litzinger

Overview

- motivation
- background
- schedule adaptation
- adaptive schedule selection
- experimental results
- summary

Motivation

- data stream processing: important computation paradigm for IoT
- example: continuous preprocessing of camera or vehicle sensor data
- various device constraints, data must be compressed
- low-power designs: manycores, moderate frequencies



- stream processing application: graph of stream tasks
  - read packet of data from input channel
  - process data packet
  - write packet of data to output channel
- with sufficient buffering capacity: pipeline enables concurrent execution within round

Motivation

- stream task itself may run in parallel
- static scheduling problem: steady state of pipeline
- throughput requirement imposes deadline constraint on steady state round
- target: minimize energy consumption
- core allocation, mapping, core operating frequency selection
- *crown scheduling*: all at the same time
- frequency selection: large impact on energy consumption due to small set of discrete voltage/frequency levels
- approach: provide *two* schedules: conservative and relaxed
- dynamic control mechanism switches between schedules

Background: processor and task model

- generic multi/manycore architecture
- each core can be set to any frequency level independently at runtime
- runtime of task scales with frequency (for computational loads)
- each task τ_j performs certain amount of work λ_j
- tasks can be moldable, partially moldable, sequential, depending on maximum width W_j
- each task is of one of two possible task types tt_j: memory-bound, computation-bound
- each task has individual parallel efficiency function e_j

Background: scheduling

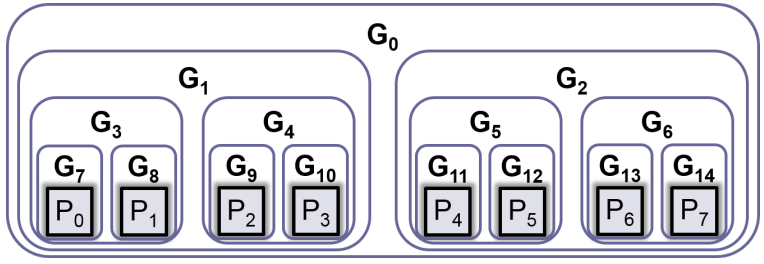
- optimization problem:
  - core allocation: assign to each task τ_j a number of cores w_j
  - mapping: assign to τ_j a core subset R_j, where $|R_j| = w_j$
  - assign to τ_j a frequency f_j
  - all tasks shall complete before the deadline M
  - energy consumption is minimized
- average power consumption: micro-benchmarks in Holmbacka & Keller (2015)
- per-core runtime can be computed as

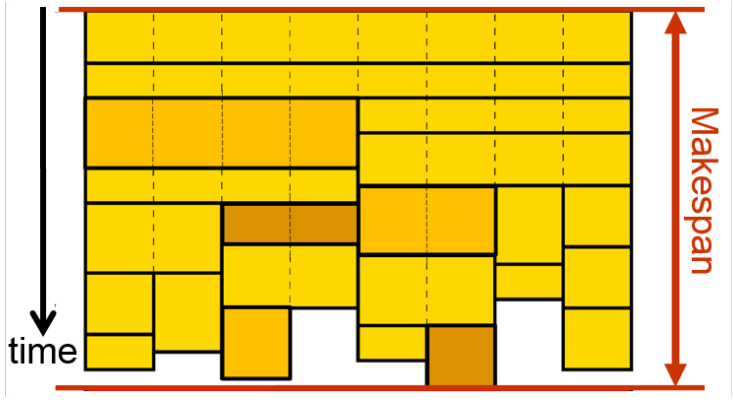
$$t_j(w_j, f_j) = \frac{\lambda_j}{f_j \cdot e_j(w_j) \cdot w_j}$$

- energy consumption for execution of τ_j:

$$E_j = t_j(w_j, f_j) \cdot w_j \cdot P(f_j, tt_j)$$

Background: crown scheduling





- structural constraint on core allocation to make joint optimization via ILP feasible
- core set partition
- map each task to *core group*, select frequency
- execute tasks in order of non-increasing width
- problem: small number of tasks and/or small number of frequency levels: negative impact on energy efficiency

Schedule adaptation

- conservative schedule: makespan probably lower than deadline, especially for applications with few tasks
- first minimize energy consumption (result: E^*), then minimize makespan for energy budget E^*, result: M^*
- relaxed schedule: keep core allocations and mapping for low switching overhead
- compute smallest makespan M' when decreasing frequency for a single task by one level
- compute energy-optimal crown schedule for deadline M'
- alternative: greedy heuristic

Schedule adaptation: greedy heuristic

- what makes a task attractive for frequency reduction?
- energy reduction:

$$\Delta E_j = t_j(w_j, f_j) \cdot P(f_j, tt_j) - t_j(w_j, f_{j-1}) \cdot P(f_{j-1}, tt_j)$$

- several properties:
  - number of cores w_j used by τ_j
  - type tt_j of τ_j
  - current frequency level f_j
  - workload λ_j of τ_j
- cumulative preference score can be computed
- sort tasks in decreasing order of energy reduction or preference score
- greedy algorithm: in that order, lower frequency level by one task-wise, when execution time exceeds extended deadline, stop
- for negative ΔE_j, do not scale down
- most likely, makespan constraint is violated but energy consumption reduced

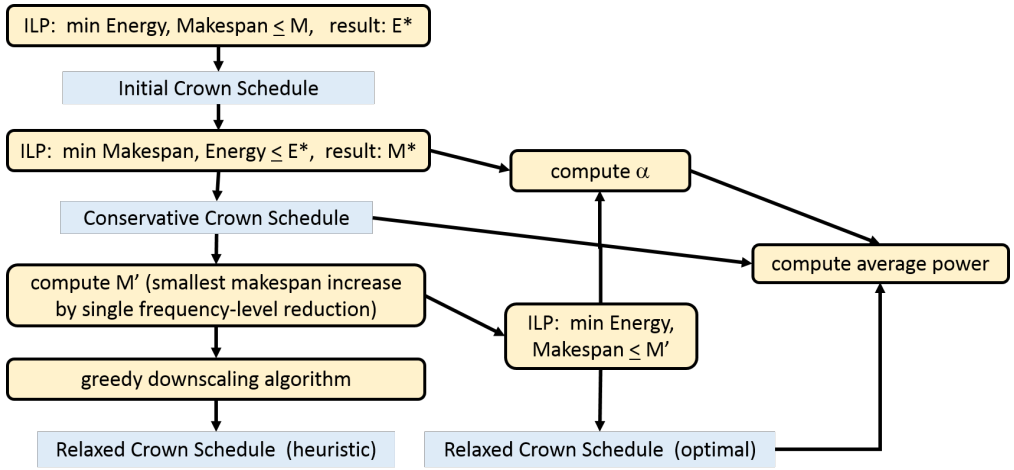
Adaptive schedule selection

- goal: switch between conservative and relaxed schedule s. t. average throughput is sufficient ($1/M$)
- if below $1/M - \delta$: switch to conservative schedule
- if above $1/M + \delta$: switch to relaxed schedule
- fraction of rounds α where conservative schedule is used:

$$\alpha = \frac{M - \widehat{M}}{M^* - \widehat{M}}$$

- from E^*, M^*, $\widehat{E}$, $\widehat{M}$, and α, average power consumption can be computed
- control mechanism can be extended to temperature: when too high, switch to relaxed, regardless of throughput
- can accommodate for further adversities: sunlight exposure, cooling issues

Experiments



- synthetic task sets, $n \in \{2, 3, 4\}$
- $p = 8$, for $n = 3$ also $p = 16$, $p = 32$
- for each combination: 10 sets memory-intensive, 10 sets other
- $W_j = p$ for feasible schedules under tight deadlines
- tight deadlines for high operating frequencies
- toolchain implemented in Python, for ILPs Gurobi 8.1.0 solver and gurobipy, execution on AMD Ryzen 7 2700X (8 cores, SMT)

Experimental results

- here: second ILP does not decrease the schedules' makespan: conservative schedule determined by first ILP
- increasing machine size has no impact on energy consumption (due to $\forall j : W_j = p$) for optimal schedules
- heuristic scheduler: only one task set treated differently for varying p
- non-memory-intensive task sets: same results for energy reduction and cumulative preference score
- conduct whole set of experiments with 8-core machine and ΔE_j ranking criterion
- interesting: average power consumption for alternating conservative and relaxed schedule vs. conservative schedule alone
- compare heuristic and optimal solution

Experimental results

- average power consumption when switching between optimal schedules vs. conservative schedule only:

# tasks	Task types	Avg. power ratio	Exec. ratio cons.
2	other	0.883	0.471
	memory	0.932	0.700
3	other	0.927	0.464
	memory	0.961	0.655
4	other	0.956	0.603
	memory	0.976	0.762

Experimental results

- Number of relaxed schedules which equal the respective conservative schedule:

# tasks	task types	optimal	heuristic
2	other	2	6
	memory	5	5
3	other	0	9
	memory	3	6
4	other	1	10
	memory	3	9
total	other	3	25
	memory	11	20
	total	**14**	**45**

Summary

- scheduling of stream processing application
- observation: makespan oftentimes less than deadline
- idea: max out available time span by switching between conservative and relaxed schedule, maintaining required throughput in the long run
- dynamic control mechanism monitors throughput, triggers switch between schedules
- can as well be employed to mitigate temperature-related issues
- tools to derive relaxed schedule from conservative schedule (determined via crown scheduling) and compute reduction in average power consumption
- neither malleability of tasks nor preemption are required
- experiments with energy profiles of real multicore platform show 2–12% reduced power consumption for small task sets of up to 4 tasks