Deterministic Scheduling in Multicore Environments using Evolutionary Algorithms

Zorana Banković

IMDEA Software Institute

ENTRA Workshop
Málaga, May 6-7, 2015

Problem Statement

Given a set of n jobs (tasks) $J := J_1, J_2, ..., J_n$*, where each job* J_i *has*:

- Release time r_i
- Deadline d_i
- Processing volume ω_i (number of cycles)

and a *multicore environment* given by:

- Number of cores, and threads per core.
- Possible (V, f) levels for each core.

Find:

- A task scheduling.
- A task-core assignment.
- (V, f) levels for each core.

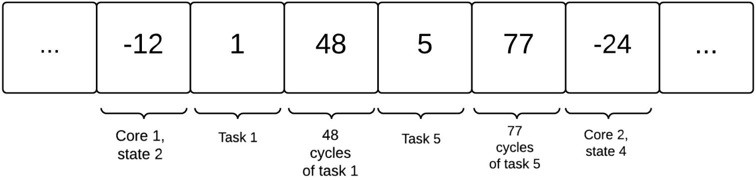
so that the **total energy** is **minimised** and **all task deadlines** are met.

Solutions

- *Evolutionary Algorithm (EA):*
  - Custom solution created.
  - Can fail in finding a viable solution when deadlines are too tight.
- *YDS:*
  - Adapted for multicore environments.
  - Solved the problem of energy increase due to static power when deadlines are loose.
- *EA + Loop Perforation:*
  - For application which can permit accuracy loss.
- *Testing environment*: XMOS one core chips with eight threads, where all the threads have the same V and f.

Deterministic Scheduling: EA

- Supports task preemption and migration.
- State $\equiv$ (V, f)



- Energy estimation:
  - The latest energy model by U. of Bristol.
    - Introduces overhead, having in mind it is instruction-based and it has to be performed for each individual in each generation.
  - *Static analysis*: total energy equal to the sum of separate programs.
- If the deadlines are too tight, it cannot always find a viable solution starting from the random initial population.

Deterministic Scheduling: The YDS Algorithm

*Frances **Y**ao, Alan **D**emers, and Scott **S**henker, "A Scheduling Model for Reduced CPU Energy", FOCS, 1995.*

Definitions:

- Set of n jobs (tasks) $J := J_1, J_2, \ldots, J_n$, where each job J_i has:
  - release time r_i
  - deadline d_i
  - processing volume ω_i (number of cycles)
- I : time interval (defined with release times and deadlines)
- $S_I \in I$: set of jobs to be processed in I, i.e. $[r_i, d_i] \in I$
- Work density in I: $\Delta_I = \frac{1}{|I|} \sum_{J_i \in S_I} \omega_i$

Algorithm:

While $J \neq \{\}$

1. Determine the time interval I of maximum density Δ_I
2. In I process the jobs of S_I at speed Δ_I according to EDF
3. Remove S_I from the set of jobs $J := J \setminus S_I$
4. Remove I from the time horizon and update the release times and deadlines of unscheduled jobs accordingly.

End While

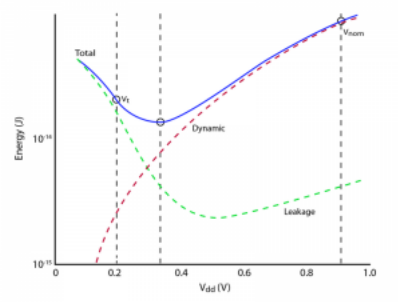
YDS: Pros and Cons

Pros:

- Very fast.
- Always finds a viable solution, i.e., all the dealines are met (if the hardware can support the processing volume).

Cons:

- It does not take into account the static power, which becomes significant if the deadlines are too loose.



Does not use information about energy, only time. *Q: Pro or Con?*

Adaptation of YDS to a Multicore Environment

Implemented:

- Two choices for optimal task-core assignment:
  1. Assign a task to the core with the least load at that moment, so the processing volume of each core is (approximately) equal.
  2. Assign a task to the core with the least work density during its active period, i.e., $[r_i, d_i]$, so its addition assumes minimal density increase.

  As the number of tasks increases, the second one performs better.
- Run YDS for each core.
- If frequency f calculated by YDS is not supported by the system, supported frequencies f_1 and f_2 are assigned in the following way:

$$\frac{\omega_i}{f} \approx \frac{\omega_{i1}}{f_1} + \frac{\omega_{i2}}{f_2}$$

$$f_1 \leq f \leq f_2$$

$$\omega_i = \omega_{i1} + \omega_{i12}$$

YDS for Multicores: Dealing with Static Power

A. Miyoshi, C. Lefurgy, E. Van Hensbergen, R. Rajamony, and R. Rajkumar. ***Critical power slope: Understanding the runtime effects of frequency scaling***.

Slope:

$$m^{f_x} = \frac{P_{f_x} - P_{f_{min}}}{f_x - f_{min}}$$

f_{min} - frequency when the core does not go in the idle state.

Critical slope, i.e. the slope when energy is equal for all the frequencies:

$$m^{f_x}_{critical} = \frac{P_{f_x} - P_{idle}}{f_x}$$

- If $m^{f_x} < m^{f_x}_{critical}$, then $E_{f_{x-\epsilon}} > E_{f_x} > E_{f_{x+\epsilon}}$, i.e., the energy increases as we decrease the frequency.
- If $m^{f_x} > m^{f_x}_{critical}$, then $E_{f_{x-\epsilon}} < E_{f_x} < E_{f_{x+\epsilon}}$, i.e., the energy decreases as we decrease the frequency.

YDS: Results After Applying the Slope Improvement

Energy savings obtained by improved YDS vs. original YDS (%)

	Scenario with tight deadlines		**Scenario with loose deadlines**	
Num.cores	*Allocation 1*	*Allocation 2*	*Allocation 1*	*Allocation 2*
1	4.18	4.18	6.21	6.21
2	1.50	4.26	14.67	14.67
3	-5.26	3.17	14.67	14.67
4	2.22	2.77	8.80	8.80
5	-3.28	3.47	11.18	11.18
6	0.95	4.34	11.82	11.82
7	4.80	3.03	10.90	10.90
8	19.36	5.61	10.56	10.56

Deterministic Scheduling: EA vs. YDS

	YDS	**EA**
Speed	Very fast	Slow
Viable solution	Always	Not always
Solution quality	Good	Solution found → better
Opt. num. of threads	Has to be set before	Intrinsically found

EA vs. YDS: Experimental Results

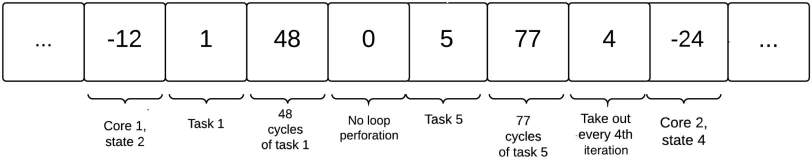
EA trained with static analysis input.

	EA En. by St. Analysis μJ	EA En. by Model μJ	YDS En. by Model μJ	En. Saving % $(Col.3 - Col.2)/Col.3$	En. Saving EA train. on Mod. %
A scenario with 22 small numeric tasks and loose deadlines					
Mean	26.3	14.3	33.1	56.8	76.57
A scenario with 22 small numeric tasks and tight deadlines					
Mean	36.9	14.6	34.8	60.92	69.83
A scenario with 16 tasks made of Biquad and FIR filters and loose deadlines					
Mean	11.25	4.38	35.3	87.59	NA
A scenario with 16 tasks made of Biquad and FIR and tight deadlines					
Mean	87	14.5	35.4	59.04	NA
A scenario with 32 tasks made of Biquad and FIR filters and loose deadlines					
Mean	165.33	17.85	68.16	73.81	NA
A scenario with 32 tasks made of Biquad and FIR filters and tight deadlines					
Mean	226.4	29.43	68.16	56.82	NA

Energy/Accuracy Trade-off: EA + Loop Perforation

Loop perforation: skip every *n*-th loop iterations.

Energy: static analysis - total energy equal to the sum of separate programs.



EA + Loop Perforation: Results

Obtained savings with different levels of minimal acceptable accuracy

Tested on 32 tasks, each implemented using either FIR or Biquad, starting at different moments

Case 1: loop perforation is applied.

Case 2: no loop perforation.

Max. Avg. Error	Case 1: Avg. En.(mJ)	Case 2: Avg. En.(mJ)	Savings(%)	
			Avg.	CI0.05
10^{-6}	0.487	0.721	16.18	0.93 - 31.42
$2 \cdot 10^{-6}$	0.461	0.597	18.21	3.54 - 32.87
$3 \cdot 10^{-6}$	0.434	0.666	31.04	13.72 - 48.37

Error: Euclidean distance between the outputs obtained with and without applying loop perforation

EA + Loop Perforation: Experimental Results

Tasks to which loop perforation has been applied. Max.error $= 10^{-6}$.

Task	Original num. of loop iterations	Final num. of loop iterations	N
FIR97-1	97	87	9
FIR85-1	85	76	9
FIR121-1	121	108	9
FIR109-1	109	104	21
FIR97-2	97	96	96
FIR85-2	85	84	84
FIR121-2	121	120	120
FIR109-2	109	108	108
FIR97-3	97	87	9
FIR85-3	85	76	9
FIR121-3	121	108	9
FIR109-3	109	97	9
FIR85-4	85	84	1
FIR121-3	121	81	3
FIR109-3	109	97	9

Conclusions

- Two algorithms with different characteristics implemented, complement each other.
- Static analysis introduced significant speed-up, although precision loss.
- EA coupled with loop perforation: if applications can permit accuracy loss, significant energy savings can be achieved.
- Possible improvements:
  - YDS: Optimal number of cores
    - For small number of cores, simply checking each possibility would be faster than introducing an additional optimisation process.
    - If the number of threads is bigger than the number of tasks, computationally extensive tasks can be further parallelised.
  - EA:
    - Additional operators, so it can always find a viable solution.
    - Techniques for speeding-up the training process.

Thank you!

Thank you for your attention!