A Unified Online Directed Acyclic Graph Flow Manager for Multicore Schedulers

Karim Kanoun (EPFL), David Atienza (EPFL), Nicholas Mastronarde (UB) and Mihaela van der Schaar (UCLA)

Karim.Kanoun@epfl.ch
Embedded Systems Laboratory
EPFL, Switzerland

ASP-DAC 2014
Outline

- Motivation
- Problem statement
- Proposed solution: Online DAG Flow Manager
- Experimental setup and results
- Conclusion
Modeling a task-graph application with a general DAG model

General Directed Acyclic Graph (DAG) model

Dependency inside a deadline tasks set

Dependency between 2 deadlines tasks set
From the application layer to the hardware layer

General Directed Acyclic Graph (DAG) model

Process and analyze the DAG of an application

Schedulers:
- Mapping tasks to cores
- Frequency selection
- Switching on/off cores

Multi-core platform
Limitation of existing DAG analysis solutions

- **Static solutions**
  - Unsuitable for applications with non-deterministic workload
  - Not applicable when the DAG model is determined at run-time

- **Online solutions**
  - Designed for their own specific schedulers (most of the time)
  - Limitation on the DAG model

![Diagram showing static and online solutions](image-url)
Problem statement

- Unified DAG analysis solution to assist schedulers:
  - Online, low-complexity and scheduler-independent
  - Process all possible applications (general DAG)
  - Provide detailed tasks dependencies to schedulers

Any external scheduler
DAG Flow Manager (DFM)
Input and key point

- Using a look-ahead window buffer to process the full DAG
- Dependencies in red color are managed with a separated list

DFM Input (for each $T_i$)
- Adjacency matrix
- Deadline value
- List of edges connecting $T_i$ with $T_{i+l}$ (with $l\neq 0$)
DAG Flow Manager (DFM): Overview

Initialization ($T_i$)
- DAG decomposition
- Dependency checking with other deadlines

Request the next $T_i$

Generate scheduler input

Multi-core platform

Scheduler

Finished task (with deadline $d_f$)

Update (only $T_f$)
- DAG decomposition
- Dependencies checking with other deadlines

Deadline Spec Table

Priority Table

$T_f$ finished?

yes

no
DFM
Tasks sets decomposition

- The decomposition is applied during the *initialization* and the *update* phases
  - Example of an H.264 video decoder DAG decomposition

\[ l_{i,0} = 0 \]
\[ l_{i,1} = 1 \]
\[ l_{i,2} = 2 \]
\[ l_{i,3} = 3 \]
\[ l_{i,4} = 4 \]
\[ l_{i,5} = 5 \]

\[ l_{i+1,0} = 0 \]
\[ l_{i+1,1} = 1 \]
\[ l_{i+1,2} = 2 \]
\[ l_{i+1,3} = 3 \]
\[ l_{i+1,4} = 4 \]
\[ l_{i+1,5} = 5 \]

\( l_{i,j} \) Depth level

\( T_j \) Group of tasks having the same deadline
DFM
Managing dependencies between the deadlines

Update phase

Initialization phase (\(T_{i+2}\))

- If task 3 finishes then edge [3-8] is cleared normally.
- If task 7 finishes then edge [7-10] is stored in the list of non-cleared dependencies.

- Edge [7-10] is detected in the list of non-cleared dependencies. It is then cleared and removed from the list.
Tasks in the Priority Table are sorted according to their:

1. Deadline
2. Depth level in $T_i$
3. Estimated workload

Scheduler input for each task:
- The total number of parent tasks that it still depends on
- Workload
- Earliest release time
- Critical path workload to its deadline

allow detecting if a task is ready to be scheduled
DFM
Prepared scheduler input: DeadlineSpec Table

- $T_i$ refers to the tasks set with deadline $d_i$
- The DeadlineSpec Table is used to track the overall progress of each tasks set $T_i$.

![Diagram]

- Total workload
- Executed workload
- Depth Table
- Scheduled workload
- #outgoing edges
- #ingoing edges

<table>
<thead>
<tr>
<th>DeadlineSpec Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_i$</td>
</tr>
<tr>
<td>$T_{i+1}$</td>
</tr>
<tr>
<td>$T_{i+2}$</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

| $l_{i,0}$          |
| $l_{i,1}$          |
| $l_{i,2}$          |
| ...               |
| ...               |

- Total workload
- Maximum number of allowed cores
- Scheduled workload
- Minimum amount of parallelizable workload
Experimental Setup

- H.264 video decoder tasks workload measured with MPARM simulator [Benini, et al., 2005] and using power figures of 90nm technology node
- Synthetic DAGs generated with GGEN tool [Cordeiro, et al., 2010]

DAG with dependent deadlines
- H.264 video decoder
- Synthetic DAG models

Existing DAG analysis solution [Li et al, 2013]

DAG Flow Manager - DFM (our solution)

Online energy-efficient scheduler ([Wei et al, 2010] adapted)

References:

[Li et al, 2013] does not consider dependencies between deadlines tasks sets

- Forced to convert the general DAG model to the fork-join DAG model and monitor then only one deadline at a time
Connecting an existing scheduler to our DFM
Scheduler overview

- Existing scheduler [Wei, et al. 2010]:
  - Tune the deadline
  - Largest Task First (LTF) on the tasks set with the earliest deadline $d_e$
  - Set up the minimum frequency based on the LTF and the tuned deadline value

Connecting an existing scheduler to our DFM
Adaptation to the general DAG model

- Adapting [Wei, et al. 2010] to the general DAG model by exploiting the output of our DFM

- Bonus: Thanks to our DFM, we can fill the generated gap (filling the gaps with $T_{e+1}$)
H.264 benchmarks

Energy consumption and deadline miss rates

- Variation of the number of deadlines in the buffer and the number of cores
- Up to 52% of energy reduction and over 80% reduction in deadline miss rates

![Graph showing energy consumption and deadline miss rates](image)

Synthetic DAG benchmarks
Frequency usage and deadline miss rates

- Deadlines values set 1% greater than the critical path workload (6 cores)
  → Simulate congested system
- 25 tasks per deadline; 40% workload variation; buffer size = 4 deadlines
- Up to 42% of energy reduction (using ARM9 power figures)

![Frequency usage](image)

![Deadline miss rates](image)

Computation overhead analysis

- Execution time measured on the iPhone 5
- Our H.264 DAG model has 300 tasks per 1 second
  → Our DFM with a buffer size of 4 deadlines needs only 650 µs to proceed the 300 tasks (<<1% overhead)

![Graphs showing execution time](image)

**H.264** - 20 tasks per deadline

**TGFF** - 4 deadlines in the buffer
Conclusion

We have proposed a unified DAG analysis solution

- Low complexity online solution
- No restrictions were imposed, covering general DAG models
- Providing detailed information about the execution status of tasks and deadlines within a look-ahead window

Significant reduction in energy consumption and deadline miss rates

- **H.264 video decoder** and **Synthetic DAGs**: up to 52% of energy reduction and over 80% reduction in deadline miss rates
- **Computation overhead**: 0.65% overhead (H.264 application)
Thank You

QUESTIONS?

Karim.Kanoun@epfl.ch
Embedded Systems Laboratory
EPFL, Switzerland