Hard-Real-Time Scheduling of Data-Dependent Tasks in Embedded Streaming Applications

Mohamed Bamakhrama and Todor Stefanov
Leiden University, The Netherlands

October 11, 2011
EMSOFT ’11 - Taipei, Taiwan

Trends in Embedded Streaming Systems Design

- Model-of-Computation (MoC) based methodologies [1]
  - Typically directed graph (e.g. SDF, CSDF)
- Platform-based design [2]
  - Heterogeneous MPSoCs
- Increasing complexity of applications [3]

[1] A. Gerstlauer et al., "Electronic System-Level Synthesis Methodologies", *IEEE TCAD*. 2009

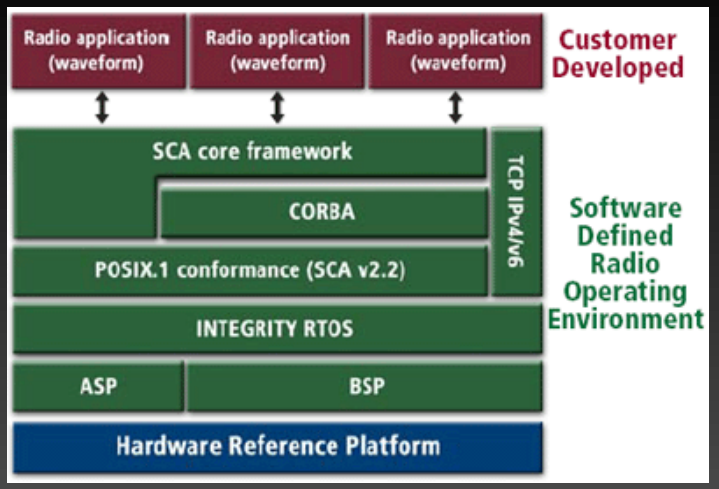
[2] A. Sangiovanni-Vincentelli and G. Martin., "Platform-based design and software design methodology for embedded systems", *IEEE D&T*. 2001

[3] L. Karam et al., “Trends in multicore DSP platforms”. *IEEE SPM*. 2009

It is all about applications!

- "Increasing complexity" means that many systems require now:
  - Hard-real-time execution on multiprocessor platforms (R1)
  - Running multiple applications on a single platform (R2)
  - Support for adding/removing applications at run-time (R3)

Interventional Radiology image filtering.
Source: Philips Healthcare



Software-Defined Radio Architecture.
Source: Green Hills Software Inc.

Current Approaches

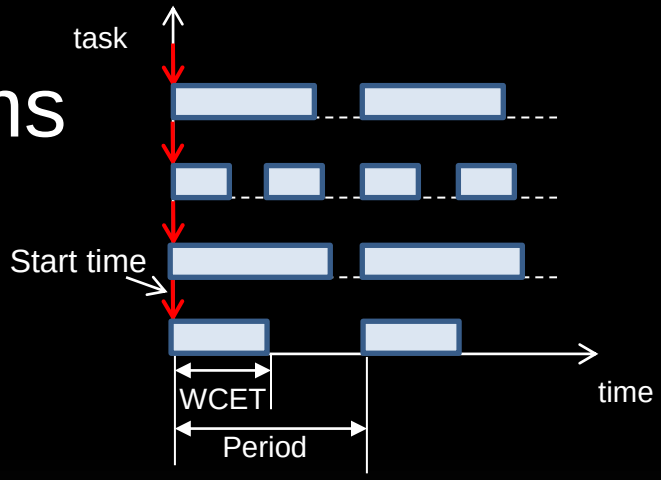
- Self-timed scheduling
  - Proven to achieve the maximum throughput and minimum latency
  - BUT no temporal isolation! ☹
- Time Division Multiplexing (TDM)
  - Provides temporal isolation ☺
  - Very similar to the Cyclic Executive approach

Our Approach

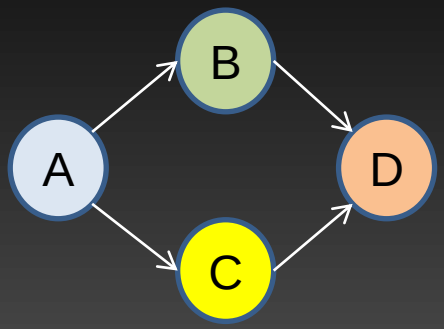
- Try to use hard-real-time multiprocessor scheduling theory
  - Many nice properties
    - **Proven** timing guarantees → (R1: HRT execution)
    - **Temporal isolation** → (R2+3: multiple apps + add/remove @ runtime)
    - **Fast** schedulability analysis → (admission control + platform sizing)
    - …
  - In the last two decades, many new developments and findings
    - Invention of **optimal** and **hybrid** algorithms, new task models, etc.
- Why this theory received little attention in the embedded streaming community?!!! ☹

The problem is…

- Most of HRT MP algorithms assume:
  - **Independent** periodic or sporadic tasks



- In contrast, MoC-based methodologies assume:
  - Tasks with **data dependencies**



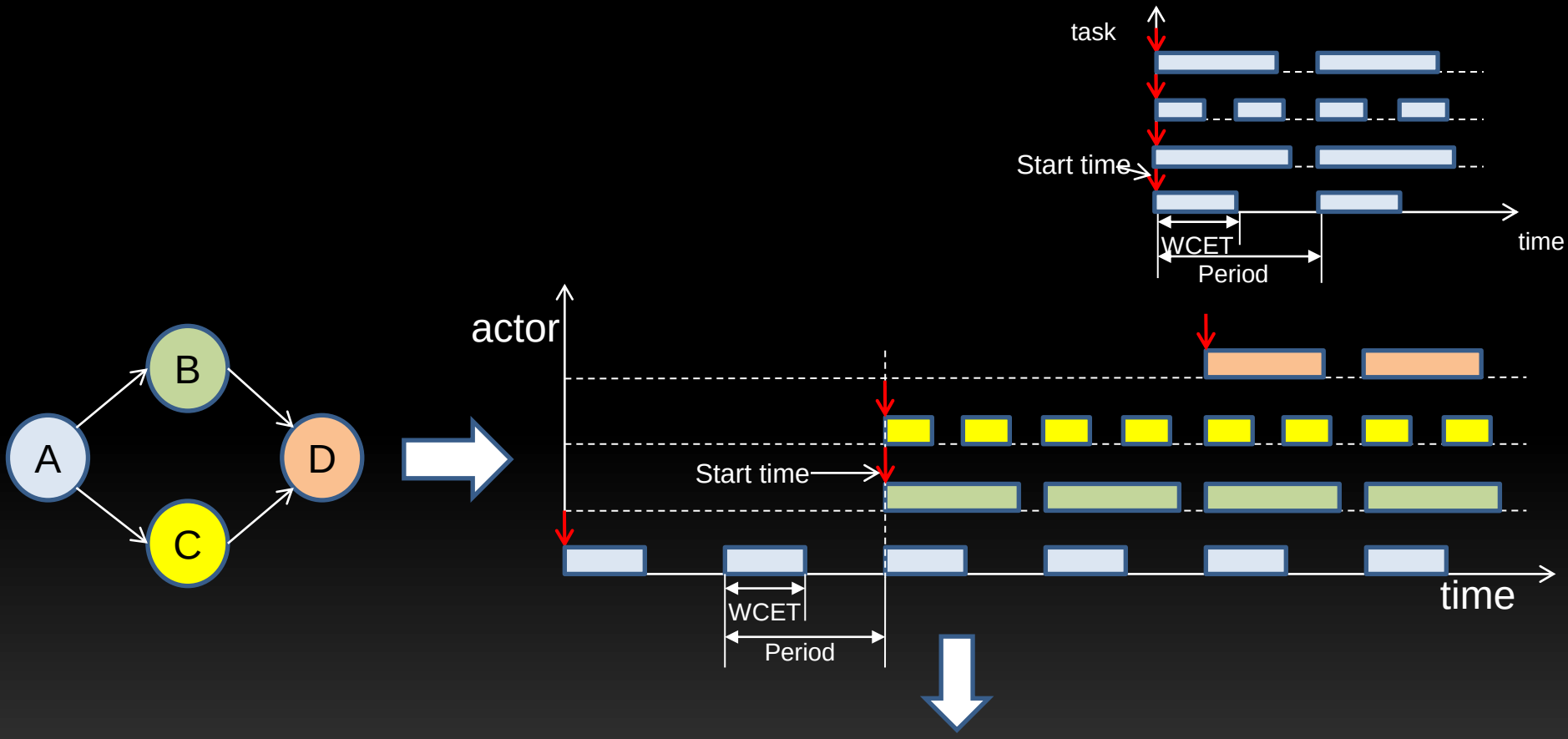
Problem Statement

- Given an application modeled as a Cyclo-Static Dataflow (CSDF) graph, can we schedule the actors as strictly periodic tasks?

- We consider applications:
  - That have *periodic* input streams
  - Modeled as *acyclic* CSDF graphs

[1] W. Thies and S. Amarasinghe. “An empirical characterization of stream programs and its implications for language and compiler design”. *In Proc. of PACT*. 2010

Contributions



Enable applying classical HRT scheduling theory to embedded streaming applications! ☺

Proving the Schedulability

- We prove the schedulability by:
  - Deriving the minimum period for each actor
  - Constructing a strictly periodic schedule for all the actors

Minimum Period Vector

- For a graph G, a period vector $\vec{\lambda}$ such that λ_i is the period of actor $v_i \in G$ is given by the solution to:

$$q_1\lambda_1 = q_2\lambda_2 = \ \ldots = q_N\lambda_N$$

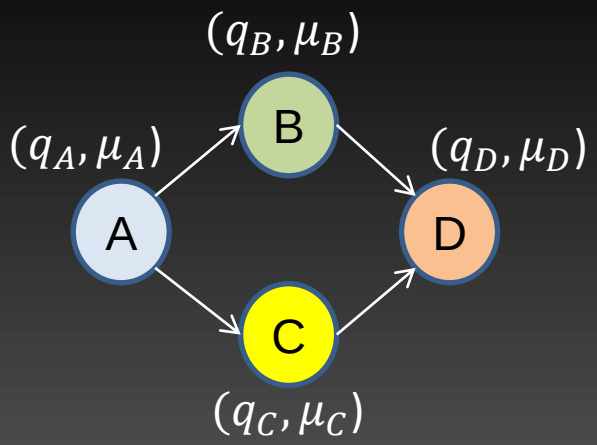
and

$$\vec{\lambda} - \vec{\mu} \geq \vec{0}$$

  - Where:

  q_i is the basic repetition of v_i

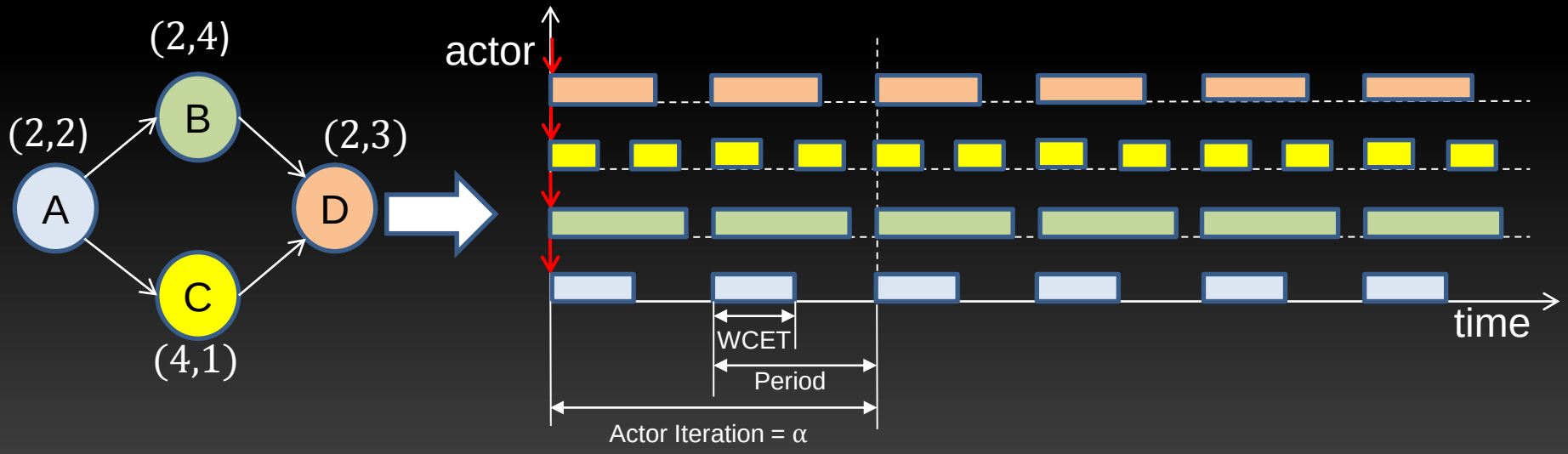
  $\vec{\mu}$ is the WCET vector of G



Understanding the Period Vector

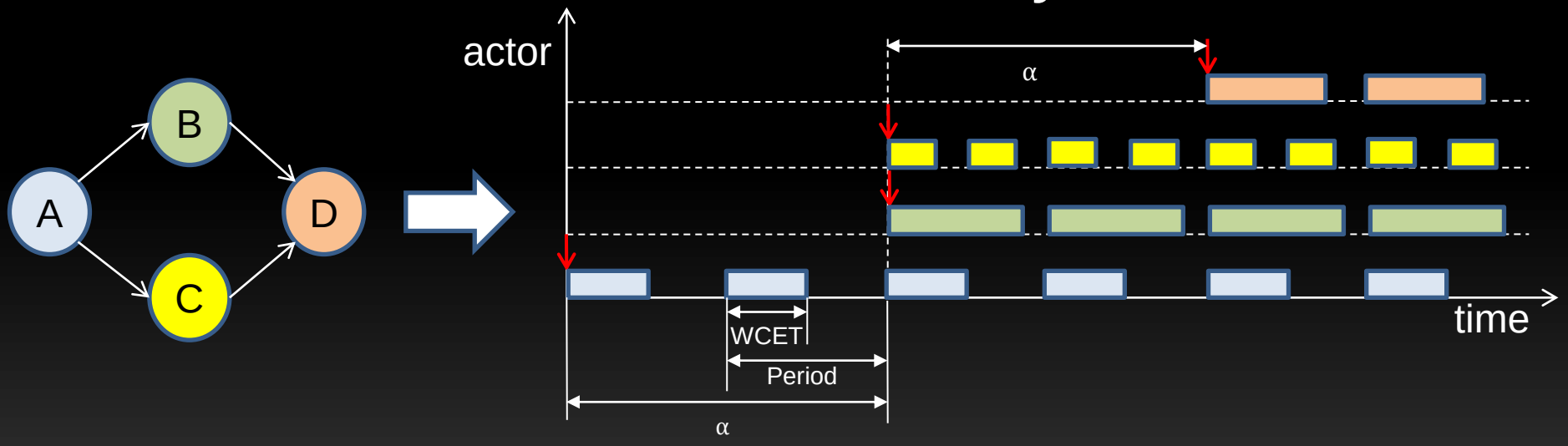
- Equalize the time needed to complete an actor iteration for all the actors

$$\alpha = q_i \lambda_i$$



Strictly Periodic Schedule

- To force strictly periodic execution, shift the start time of each actor by α time-units



A strictly periodic schedule exists! ☺

- However, do we have to shift by α?

Earliest start time and minimum buffer size

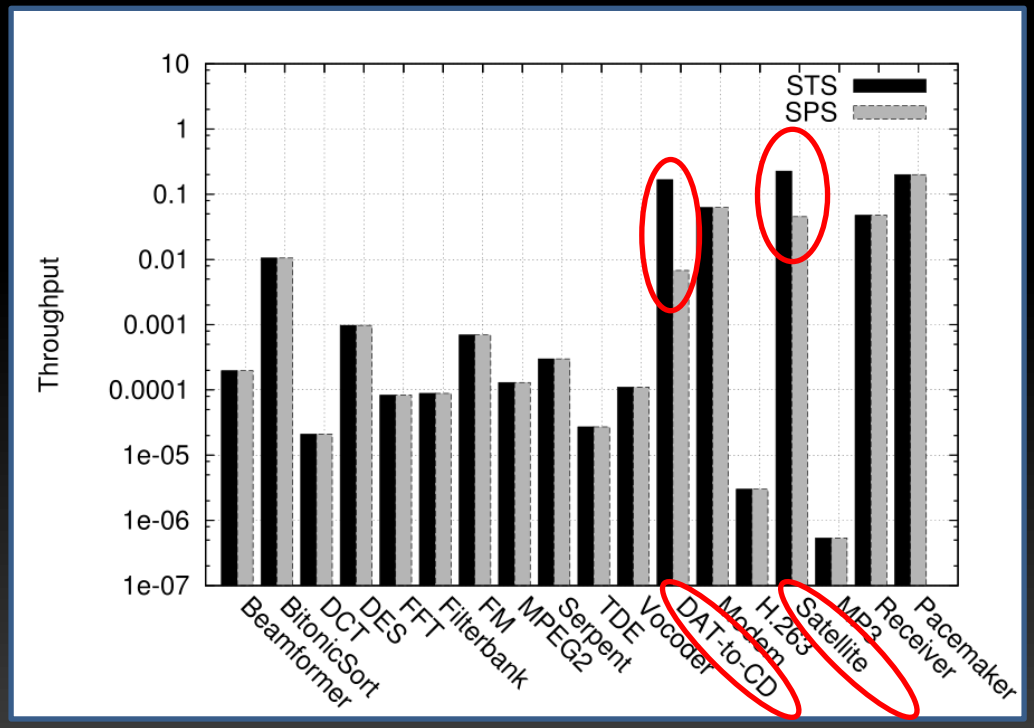
- Starting the actors earlier reduces initial response times and buffer sizes
- We provide two theorems to determine the minimum values for start times and buffer sizes

How Good is our Strictly Periodic Scheduling (SPS)?

- To measure the “goodness” of SPS, we evaluate it using a set of real-life applications (19 apps in total)
  - For each application, we compare the throughput resulting from SPS with the maximum achievable throughput
- We provide an analytical test to determine whether an application achieves the maximum throughput under SPS

Throughput Comparison

- SPS achieves the maximum throughput for 17 out of 19 apps
- Why these two apps do not achieve the maximum throughput?



Because these two apps have mismatched I/O rates

Matched vs. Mismatched

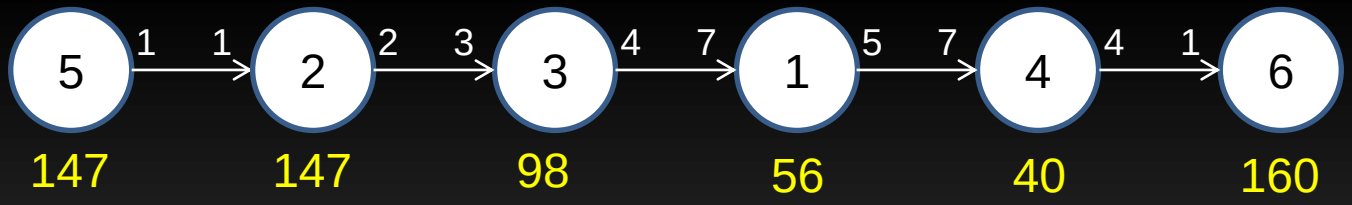
- Let:

  $\mathrm{L} = lcm(q_1, q_2, \ldots, q_N)$

  $\mathrm{H} = max(q_i \mu_i)$

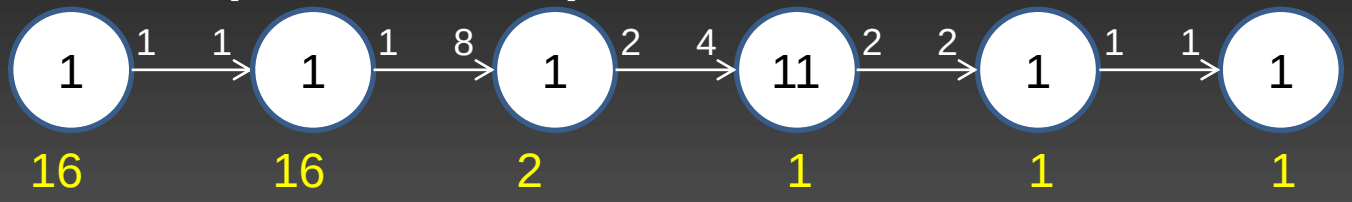
$\mathbf{H \bmod L = 0}$ → matched

- DAT-to-CD (mismatched)



L = 23520, H = 960

- Modem (matched)



L = 16, H = 16

Goodness of SPS

- If an application is matched I/O → it achieve the maximum throughput under SPS

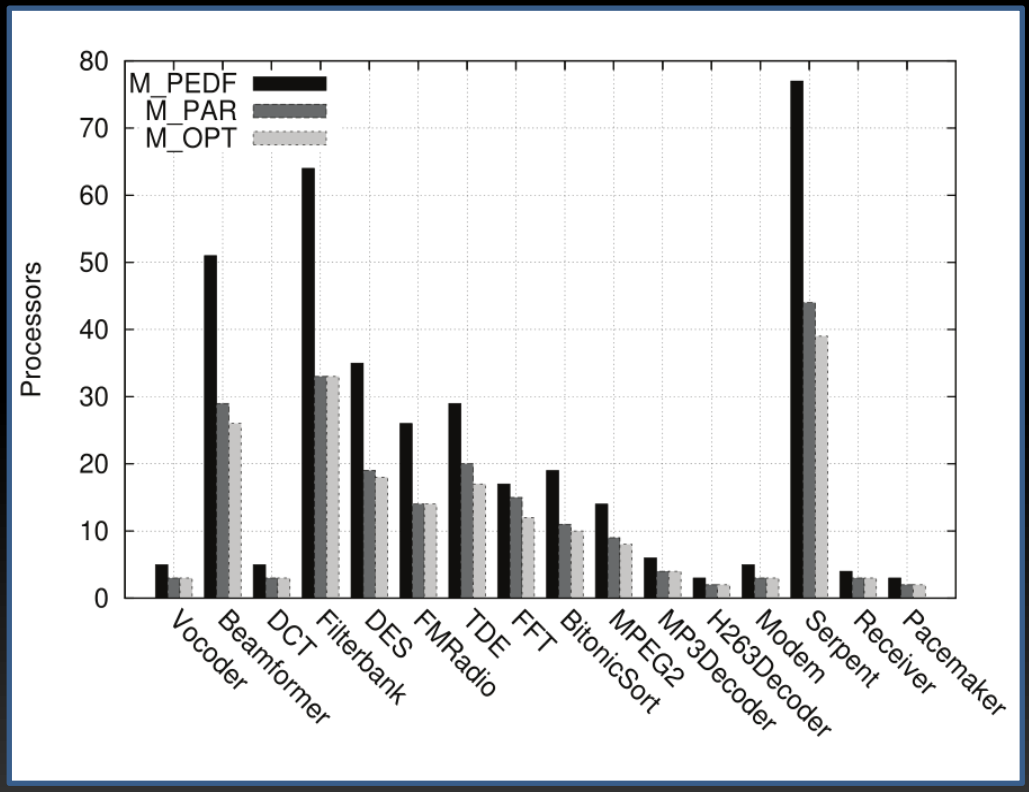
- What are the implications?
  - For matched I/O apps, schedulability analysis can be used to easily find the minimum required resources
  - Eliminates the need for complex Design Space Exploration (DSE)

DSE is not needed! ☺

- $M_{OPT} = \left\lceil \sum_{v_i \in G} \frac{\mu_i}{\lambda_i} \right\rceil$

  μ_i = WCET

  λ_i = Period



Conclusions

- We bridge the classical real-time scheduling theory and the embedded streaming theory
  - By enabling the use of HRT scheduling algorithms to schedule embedded applications with data dependencies
- We provide an analytical test to determine whether an application belongs to matched I/O category
  - Matched I/O applications are guaranteed to achieve the maximum throughput under SPS
- For matched I/O applications, we do not need DSE to find the minimum number of processors needed to guarantee the maximum throughput
  - Instead, use schedulability analysis

Thank you

?