Combining Dataflow Applications and Real-time Task Sets on Multi-core Platforms

Hazem Ismail Ali  Benny Akesson  Luís Miguel Pinho

CISTER Research Centre/INESC-TEC, Polytechnic Institute of Porto, Portugal

{haali, kbake, lmp}@isep.ipp.pt

SCOPES 2017
Sankt Goar, Germany
June 12-13, 2017
Overview

1. Introduction
2. Background
3. Problem
4. Proposed Solution
5. Contributions
6. Evaluation
7. Conclusion
8. Future Work
Overview

1. Introduction
2. Background
3. Problem
4. Proposed Solution
5. Contributions
6. Evaluation
7. Conclusion
8. Future Work
Introduction

- Embedded systems serve us in our daily life.
Embedded systems serve us in our daily life.

Trend of growing functionality, which means:
Introduction

- Embedded systems serve us in our daily life.
- Trend of growing functionality, which means:
  - incorporating **multi-core** platforms.
Embedded systems serve us in our daily life.

Trend of growing functionality, which means:

- incorporating multi-core platforms.
- running different applications with different requirements.
Embedded systems serve us in our daily life.

Trend of growing functionality, which means:

- incorporating **multi-core** platforms.
- running **different applications** with **different requirements**.

**Different requirements** may mean:
Introduction

- Embedded systems serve us in our daily life.
- Trend of growing functionality, which means:
  - incorporating multi-core platforms.
  - running different applications with different requirements.
- Different requirements may mean:
  - Computational demands.
Embedded systems serve us in our daily life.

Trend of growing functionality, which means:

- incorporating **multi-core** platforms.
- running **different applications** with **different requirements**.

**Different requirements** may mean:

- **Computational demands**.
- **Timing constraints**.
Introduction

- Embedded systems serve us in our daily life.
- Trend of growing functionality, which means:
  - incorporating multi-core platforms.
  - running different applications with different requirements.
- Different requirements may mean:
  - Computational demands.
  - Timing constraints.
Embedded systems serve us in our daily life.

Trend of growing functionality, which means:
- incorporating multi-core platforms.
- running different applications with different requirements.

Different requirements may mean:
- Computational demands.
- Timing constraints.
Introduction

Trend in embedded systems of running different application models with timing constraints.
Introduction

- **Trend** in embedded systems of running different application models with timing constraints.
- We are concerned with systems running applications represented as **Real-time Tasks** and **Dataflow Graphs**. Why?
Introduction

- **Trend** in embedded systems of running different application models with timing constraints.
- We are concerned with systems running applications represented as Real-time Tasks and Dataflow Graphs.
- Dataflow represents Digital Signal Processing (DSP), Streaming and Multimedia applications.
Introduction

- **Trend** in embedded systems of running *different application models* with timing constraints.
- We are concerned with systems running applications represented as **Real-time Tasks** and **Dataflow Graphs**.
- Dataflow represents Digital Signal Processing (DSP), Streaming and Multimedia applications.
- **Real-time tasks covers a wide range of Control applications.**
A task that releases its jobs periodically after fixed time interval, called Period.
A task that releases its jobs periodically after **fixed time interval**, called **Period**.

This work is concerned with the **Arbitrary-deadline task model**.
A task that releases its jobs periodically after **fixed time interval**, called **Period**.

This work is concerned with the **Arbitrary-deadline task model**.

A **task** $\tau_i$ is defined by $\tau_i = (a_i, C_i, T_i, D_i)$, where:
A task that releases its jobs periodically after **fixed time interval**, called **Period**.

This work is concerned with the **Arbitrary-deadline task model**.

A **task** $\tau_i$ is defined by $\tau_i = (a_i, C_i, T_i, D_i)$, where:

- $a_i$ is the offset.
- $C_i$ is the WCET.
- $T_i$ is the period.
- $D_i$ is the relative deadline.
Background

Synchronous Dataflow (SDF)

Figure: An example SDF graph.
**Background**

**Synchronous Dataflow (SDF)**

- Actors have a fixed production and consumption rates.

Figure: An example SDF graph.
Background
Synchronous Dataflow (SDF)

- Actors have a fixed production and consumption rates.
- Channels can have initial tokens.

Figure: An example SDF graph.
Background

Synchronous Dataflow (SDF)

- Actors have a fixed production and consumption rates.
- Channels can have initial tokens.
- Tokens are always consumed in a FIFO order.

Figure: An example SDF graph.
Background
Homogeneous Synchronous Dataflow (HSDF)

Figure: An example HSDF graph.
Background
Homogeneous Synchronous Dataflow (HSDF)

- A special case of SDF.

Figure: An example HSDF graph.
Background
Homogeneous Synchronous Dataflow (HSDF)

- A special case of SDF.
- When every actor is fired once, the distribution of tokens on all channels return to their initial state (graph iteration).

Figure: An example HSDF graph.
Background
Homogeneous Synchronous Dataflow (HSDF)

- A **special case** of SDF.
- When **every actor is fired once**, the distribution of tokens on all channels return to their initial state (**graph iteration**).
- **Other models can be converted** to an equivalent HSDF.

**Figure**: An example HSDF graph.
Background
Homogeneous Synchronous Dataflow (HSDF)

- A special case of SDF.
- When every actor is fired once, the distribution of tokens on all channels return to their initial state (graph iteration).
- Other models can be converted to an equivalent HSDF.
  - Pros. Parallelism.

Figure: An example HSDF graph.
Background
Homogeneous Synchronous Dataflow (HSDF)

- A **special case** of SDF.
- When **every actor is fired once**, the distribution of tokens on all channels return to their initial state (**graph iteration**).
- Other models can be converted to an equivalent HSDF.
  - Pros. **Parallelism**.
  - Cons. **Large Graph size**.

**Figure:** An example HSDF graph.
Overview

1. Introduction
2. Background
3. Problem
4. Proposed Solution
5. Contributions
6. Evaluation
7. Conclusion
8. Future Work
Problem Statement

Research Question?

How can real-time embedded systems safely incorporate mixed application models, dataflow and real-time tasks, with timing constraints onto multi-core platforms, such that their timing constraints are satisfied?
Problem Statement

How can real-time embedded systems safely incorporate mixed application models, dataflow and real-time tasks, with timing constraints onto multi-core platforms, such that their timing constraints are satisfied?
Problem Statement

Research Question?

How can real-time embedded systems safely incorporate mixed application models, dataflow and real-time tasks, with timing constraints onto multi-core platforms, such that their timing constraints are satisfied?
Problem Statement

Research Question?
How can real-time embedded systems safely incorporate mixed application models, dataflow and real-time tasks, with timing constraints onto multi-core platforms, such that their timing constraints are satisfied?
Problem Statement

Research Question?

How can real-time embedded systems safely incorporate mixed application models, dataflow and real-time tasks, with timing constraints onto multi-core platforms, such that their timing constraints are satisfied?
Overview

1. Introduction
2. Background
3. Problem
4. Proposed Solution
5. Contributions
6. Evaluation
7. Conclusion
8. Future Work
Problem Statement

Main Idea

- The real-time domain has a well developed analysis and scheduling techniques.
The real-time domain has a well developed analysis and scheduling techniques.

A **unified model** is required to transform dataflow applications into independent periodic real-time tasks to enable real-time analysis and scheduling techniques.
The real-time domain has a well developed analysis and scheduling techniques.

A **unified model** is required to transform dataflow applications into independent periodic real-time tasks to enable real-time analysis and scheduling techniques.

**Mapping algorithm** that utilize multi-core platform resources efficiently.
Overview

1. Introduction
2. Background
3. Problem
4. Proposed Solution
5. Contributions
6. Evaluation
7. Conclusion
8. Future Work
Contributions

Addressed Problems

System

Applications

Dataflow
represented as SDF graphs
WCET, P/C rates, Throughput, Latency

Traditional
Real-time Tasks
Arbitrary-deadline tasks
Contributions

Addressed Problems

- Dataflow represented as SDF graphs
  - WCET, P/C rates, Throughput, Latency

- Traditional Real-time Tasks
  - Arbitrary-deadline tasks

- System
- Applications

Arbitrary-deadline tasks
Contributions

Addressed Problems

System

Applications

Dataflow
represented as SDF graphs
WCET, P/C rates, Throughput, Latency

Traditional
Real-time Tasks
Arbitrary-deadline tasks

How to deal with large HSDF graphs?

Arbitrary-deadline tasks
Contributions

Addressed Problems

- Dataflow represented as SDF graphs
- WCET, P/C rates, Throughput, Latency
- How to deal with large HSDF graphs?
  - Slack-based Merging Heuristic

Arbitrary-deadline tasks

Traditional Real-time Tasks
- Arbitrary-deadline tasks
Contributions
Addressed Problems

System

Applications

Dataflow
represented as SDF graphs
WCET, P/C rates, Throughput, Latency

Traditional
Real-time Tasks
Arbitrary-deadline tasks

How to deal with large HSDF graphs?
Slack-based Merging Heuristic

Reduced-size
graph

Arbitrary-deadline tasks
Contributions
Addressed Problems

System
Applications
Dataflow
represented as SDF graphs
WCET, P/C rates, Throughput, Latency

Traditional
Real-time Tasks
Arbitrary-deadline tasks

How to deal with large HSDF graphs?
Slack-based Merging Heuristic

Reduced-size graph

How to transform HSDF into real-time tasks?

Arbitrary-deadline tasks
Contributions

Addressed Problems

System

Applications

Dataflow
represented as SDF graphs
WCET, P/C rates, Throughput, Latency

Traditional
Real-time Tasks
Arbitrary-deadline tasks

How to deal with large HSDF graphs?
Slack-based Merging Heuristic

Reduced-size
graph

How to transform HSDF intoeal-time tasks?
Timing Parameter Extraction Heuristic

Arbitrary-deadline tasks
Contributions

Addressed Problems

- System
- Applications
- Dataflow: represented as SDF graphs
  - WCET, P/C rates, Throughput, Latency
- Traditional Real-time Tasks
  - Arbitrary-deadline tasks

- How to deal with large HSDF graphs?
  - Slack-based Merging Heuristic

- Reduced-size graph

- How to transform HSDF into real-time tasks?
  - Timing Parameter Extraction Heuristic

- Arbitrary-deadline tasks

- How to map efficiently on multi-core platform?
Slack-based Merging

Motivation
Motivation

- Proposed solution works with HSDF graphs.
Slack-based Merging

Motivation

- Proposed solution works with HSDF graphs.
- Transformation to HSDF can lead to an exponential increase in the size of the application graph that significantly increases the run-time of the analysis.
Slack-based Merging

Motivation

- Proposed solution works with HSDF graphs.
- Transformation to HSDF can lead to an exponential increase in the size of the application graph that significantly increases the run-time of the analysis.

What is it?

The *Slack-based Merging* is a novel offline graph reduction technique that generates reduced-size HSDF graph, satisfying the throughput and latency constraints of the original HSDF graph.
Slack-based Merging

**Motivation**
- Proposed solution works with HSDF graphs.
- Transformation to HSDF can lead to an exponential increase in the size of the application graph that significantly increases the run-time of the analysis.

**What is it?**
The *Slack-based Merging* is a novel offline graph reduction technique that generates reduced-size HSDF graph, satisfying the throughput and latency constraints of the original HSDF graph.

It helps reducing the overall design time of the real-time system.
Slack-based Merging
How it works?

Throughput $\zeta = 1/3$,

End-to-end latency $D = 8$

Figure: HSDF graph
Slack-based Merging

How it works?

Throughput $\zeta = \frac{1}{3}$,

End-to-end latency $D = 8$

- Merges a pair of firings of the same SDF actor, whenever a slack is available.

Figure: HSDF graph
Slack-based Merging
How it works?

Throughput $\zeta = 1/3$, 
End-to-end latency $D = 8$

- Merges a pair of firings of the same SDF actor, whenever a slack is available.
- The slack $\sigma$ of a firing of an SDF actor is the difference between its latest finish time $\theta$ and its earliest start time $\vartheta$ minus its computation time.

Figure: HSDF graph
Slack-based Merging
How it works?

Throughput $\zeta = 1/3$,
End-to-end latency $D = 8$

- Merges a pair of firings of the same SDF actor, whenever a slack is available.
- The slack $\sigma$ of a firing of an SDF actor is the difference between its latest finish time $\theta$ and its earliest start time $\vartheta$ minus its computation time.
- The merge operation is considered valid iff the resulting graph:

\[ \begin{align*}
\text{(Diagram of a SDF graph with labeled nodes and edges)}
\end{align*} \]
Slack-based Merging
How it works?

Throughput $\zeta = 1/3$,

End-to-end latency $D = 8$

- Merges a pair of firings of the same SDF actor, whenever a slack is available.
- The slack $\sigma$ of a firing of an SDF actor is the difference between its latest finish time $\theta$ and its earliest start time $\vartheta$ minus its computation time.
- The merge operation is considered valid iff the resulting graph:
  - is deadlock free.
Slack-based Merging

How it works?

Throughput $\zeta = 1/3$,

End-to-end latency $D = 8$

- Merges a pair of firings of the same SDF actor, whenever a slack is available.
- The slack $\sigma$ of a firing of an SDF actor is the difference between its latest finish time $\theta$ and its earliest start time $\vartheta$ minus its computation time.
- The merge operation is considered valid iff the resulting graph:
  - is deadlock free.
  - satisfies the timing constraints (throughput and latency).
Slack-based Merging

How it works?

Throughput $\zeta = 1/3$,

End-to-end latency $D = 8$

- Merges a pair of firings of the same SDF actor, whenever a slack is available.
- The slack $\sigma$ of a firing of an SDF actor is the difference between its latest finish time $\theta$ and its earliest start time $\vartheta$ minus its computation time.
- The merge operation is considered valid iff the resulting graph:
  - is deadlock free.
  - satisfies the timing constraints (throughput and latency).
- The algorithm iterates until there is no possible merges.
Slack-based Merging

How it works?

Throughput $\zeta = \frac{1}{3}$,

End-to-end latency $D = 8$

- Merges a **pair of firings** of the same SDF actor, whenever a **slack** is available.
- The **slack** $\sigma$ of a firing of an SDF actor is the difference between its **latest finish time** $\theta$ and its **earliest start time** $\vartheta$ minus its **computation time**.
- The **merge** operation is considered **valid** iff the resulting graph:
  - is **deadlock free**.
  - satisfies the timing constraints (**throughput** and **latency**).
- The algorithm iterates until there is no possible merges.

Figure: Final merged
Slack-based Merging
Advantage and Disadvantage

- Generates a **reduced-size** HSDF graph that **speeds up** the overall design time.
Slack-based Merging

Advantage and Disadvantage

- Generates a **reduced-size** HSDF graph that **speeds up** the overall design time.
- The merging process **hides parallelism**.
Timing Parameter Extraction (TPE)
Timing Parameter Extraction (TPE)

Motivation

To create a *unified model* for all applications running on the multi-core platform.
Timing Parameter Extraction (TPE)

Motivation
To create a *unified model* for all applications running on the multi-core platform.

What is it?
The TPE transforms HSDF graphs into independent periodic real-time tasks by extracting its actors’ timing parameters $(a_i, C_i, T_i, D_i)$ at design time.
Motivation

To create a *unified model* for all applications running on the multi-core platform.

What is it?

The TPE transforms HSDF graphs into independent periodic real-time tasks by extracting its actors’ timing parameters \((a_i, C_i, T_i, D_i)\) at design time.

It consists of two main phases:

1. **Finding all the possible paths** in the applications graph.
Timing Parameter Extraction (TPE)

**Motivation**

To create a *unified model* for all applications running on the multi-core platform.

**What is it?**

The TPE transforms HSDF graphs into independent periodic real-time tasks by extracting its actors’ timing parameters \((a_i, C_i, T_i, D_i)\) at design time.

**It consists of two main phases:**

1. **Finding all the possible paths** in the applications graph.
2. **Extracting the timing parameters** of individual actors.
Timing Parameter Extraction (TPE)

First phase: Finding all possible paths
Timing Parameter Extraction (TPE)

First phase: Finding all possible paths

1) Creation of source and sink actors:

![Diagram of a graph with nodes labeled a1, a0, b2, b1, c0, d0, input, and output, with arrows indicating paths.]

**Figure**: Adding source s and sink t to HSDF.
Timing Parameter Extraction (TPE)

First phase: Finding all possible paths

1) Creation of source and sink actors:

- **Unifies all the paths** that traverse the graph from the input to the output.

Figure: Adding source $s$ and sink $t$ to HSDF.
1) Creation of source and sink actors:

- Unifies all the paths that traverse the graph from the input to the output.
- Allows to deal with multiple input/output graphs.

Figure: Adding source $s$ and sink $t$ to HSDF.
Timing Parameter Extraction (TPE)

First phase: Finding all possible paths
Timing Parameter Extraction (TPE)

First phase: Finding all possible paths

2) Path enumeration:

Partial Path: 
\[ P_l = \langle v_x, \ldots, v_j \rangle \]

Extend Partial Path using 
\[ \text{Succ}(v_j) = \langle v_{j_1}, v_{j_2}, v_{j_3}, \ldots, v_{j_l} \rangle \]

Resulting Paths: 
\[ P_{i_1} = \langle v_x, \ldots, v_j, v_{j_1} \rangle \]
\[ P_{i_2} = \langle v_x, \ldots, v_j, v_{j_2} \rangle \]
\[ \vdots \]
\[ P_{i_l} = \langle v_x, \ldots, v_j, v_{j_l} \rangle \]

- Finds all timed-constrained paths and orders them (descendingly) according to sensitivity \( \gamma \).
Timing Parameter Extraction (TPE)
First phase: Finding all possible paths

2) Path enumeration:

Partial Path:
\[ P_i = \langle v_x, \ldots, v_j \rangle \]

Extend Partial Path using
\[ \text{Succ}(v_j) = \langle v_{j_1}, v_{j_2}, v_{j_3}, \ldots, v_{j_l} \rangle \]

Resulting Paths:
\[ P_{i_1} = \langle v_x, \ldots, v_j, v_{j_1} \rangle \]
\[ P_{i_2} = \langle v_x, \ldots, v_j, v_{j_2} \rangle \]
\[ P_{i_l} = \langle v_x, \ldots, v_j, v_{j_l} \rangle \]

- Finds all timed-constrained paths and orders them (descendingly) according to sensitivity \( \gamma \).
- The sensitivity \( \gamma \) is a measure of the criticality of a time constrained path with respect to density.
Timing Parameter Extraction (TPE)
Second phase: Extracting timing parameters

The second phase repeats for each application. It does the following:

1. Picks a path $P_i$ in order of sensitivity.
Timing Parameter Extraction (TPE)
Second phase: Extracting timing parameters

The second phase repeats for each application. It does the following:

1. Picks a path $P_i$ in order of sensitivity.
2. Each actor in the selected path $P_i$ is assigned deadlines $D_j$ and offsets $S_j$. 
The second phase repeats for each application. It does the following:

1. Picks a path $P_i$ in order of sensitivity.
2. Each actor in the selected path $P_i$ is assigned deadlines $D_j$ and offsets $S_j$.
3. The method for assigning individual deadlines is based on the two deadline assignment techniques (NORM/PURE) that are widely used in the literature.
Timing Parameters Extraction (TPE)

- TPE transforms HSDF graphs (cyclic or acyclic) into arbitrary-deadline tasks.
Timing Parameters Extraction (TPE)

- TPE transforms HSDF graphs (cyclic or acyclic) into arbitrary-deadline tasks.
- Enables applying real-time analysis techniques on dataflow graphs follows from representing as tasks.
Communication-aware Mapping

Motivation

Need for an efficient mapping solution to improve utilization of the platform resources, taking into account:

- communication cost.
Communication-aware Mapping

Motivation

Need for an efficient mapping solution to improve utilization of the platform resources, taking into account:

- communication cost.
- satisfying timing constraints.
Communication-aware Mapping

Motivation

Need for an efficient mapping solution to improve utilization of the platform resources, taking into account:

- communication cost.
- satisfying timing constraints.
Communication-aware Mapping

Motivation

Need for an efficient mapping solution to improve utilization of the platform resources, taking into account:

- communication cost.
- satisfying timing constraints.

What is it?

- The Communication-aware Mapping is a heuristic for mapping mixed application models on multi-core platforms.
Communication-aware Mapping

Motivation

Need for an efficient mapping solution to improve utilization of the platform resources, taking into account:

- communication cost.
- satisfying timing constraints.

What is it?

- The Communication-aware Mapping is a heuristic for mapping mixed application models on multi-core platforms.
- Based on a mapping heuristic called Critical-Path-First (CPF).
Dataflow applications are **data-driven networks** of actors.
Dataflow applications are **data-driven networks** of actors.

This **communication** is significant.
Communication-aware Mapping
Communication Modelling

- Dataflow applications are **data-driven networks** of actors.
- This **communication** is **significant**.
- It **impacts** the **overall utilization** of the resources and the end-to-end response time.
Communication-aware Mapping
Communication Modelling

- Dataflow applications are \textit{data-driven networks} of actors.
- This \textit{communication} is \textit{significant}.
- It \textit{impacts} the \textit{overall utilization} of the resources and the end-to-end response time.

\textbf{Therefore}

Communication should be modelled in a way that ensures \textit{correct execution} of dataflow applications, \textit{satisfying their timing constraints}. 
Dataflow applications are data-driven networks of actors. This communication is significant. It impacts the overall utilization of the resources and the end-to-end response time.

Therefore

Communication should be modelled in a way that ensures correct execution of dataflow applications, satisfying their timing constraints.

The communication modelling is done in a two step process.
The first step is **initial modelling**, where we transform all the messages in the HSDF graph to actors. We refer to them as **message actors** ($m$).
The first step is **initial modelling**, where we transform all the messages in the HSDF graph to actors. We refer to them as **message actors** \( (m) \).

(a) Merged graph \( G_m \)
Communication-aware Mapping

Communication Modelling

- The first step is **initial modelling**, where we transform all the messages in the HSDF graph to actors. We refer to them as **message actors** ($m$).

(a) Merged graph $G_m$

(b) Merged graph with message actors $G_{com}$
Communication-aware Mapping
Communication Modelling

- WCET of message actors equal to the time required to traverse the IN of the platform from the source to destination.

How?
Communication-aware Mapping
Communication Modelling

- WCET of message actors equal to the time required to traverse the IN of the platform from the source to destination.
- Initially, we assume each message traverses the maximum number of hops on the platform.
Communication-aware Mapping
Communication Modelling

- WCET of message actors equal to the time required to traverse the IN of the platform from the source to destination.
- Initially, we assume each message traverses the maximum number of hops on the platform.
- Second step comes after mapping the application on the platform (will be discussed later).
Communication-aware Mapping

How it works?

- The *Communication-aware Mapping* uses the output of the TPE algorithm.
Communication-aware Mapping

How it works?

- The *Communication-aware Mapping* uses the output of the TPE algorithm.

- **Independent real-time tasks** considered as **single node graphs**.
Communication-aware Mapping

How it works?

- The *Communication-aware Mapping* uses the output of the TPE algorithm.
- **Independent real-time tasks** considered as **single node graphs**.
- The algorithm comprises **three stages**:
Communication-aware Mapping

How it works?

- The *Communication-aware Mapping* uses the output of the TPE algorithm.

- **Independent real-time tasks** considered as **single node graphs**.

- The algorithm comprises **three stages**:
  1. **Sensitive-Path-First (SPF)** heuristic, which is responsible for allocating the application actors (not the message actors), such that the system is schedulable.
Communication-aware Mapping

How it works?

- The *Communication-aware Mapping* uses the output of the TPE algorithm.
- **Independent real-time tasks** considered as **single node graphs**.
- The algorithm comprises **three stages**:
  1. **Sensitive-Path-First (SPF)** heuristic, which is responsible for allocating the application actors (not the message actors), such that the system is schedulable.
  2. **Eliminating message actors with zero computation**, whose source and destination actors have been mapped to the same core to eliminate them from the $G_{com}$ graph.
Communication-aware Mapping

How it works?

- **The Communication-aware Mapping** uses the output of the TPE algorithm.

- **Independent real-time tasks** considered as **single node graphs**.

- The algorithm comprises **three stages**:
  1. **Sensitive-Path-First (SPF)** heuristic, which is responsible for allocating the application actors (not the message actors), such that the system is schedulable.
  2. **Eliminating message actors with zero computation**, whose source and destination actors have been mapped to the same core to eliminate them from the $G_{com}$ graph.
  3. **TPE algorithm** to update the timing parameters of the actors and the message actors in the graph according to the current mapping platform.
Complete Approach

**System**

- **Applications**
  - **Dataflow** represented as SDF graphs
    - WCET, P/C rates, Throughput, Latency
  - **Traditional Real-time Tasks**
    - Arbitrary-deadline tasks

- **How to deal with large HSDF graphs?**
  - Slack-based Merging Heuristic

- **How to transform HSDF into real-time tasks?**
  - Timing Parameter Extraction Heuristic

- **Arbitrary-deadline tasks**

- **How to map efficiently on multi-core platform?**
  - Communication-aware mapping Heuristic
Complete Approach

System

Applications

- Dataflow
  represented as SDF graphs
  WCET, P/C rates, Throughput, Latency

- Traditional
  Real-time Tasks
  Arbitrary-deadline tasks

How to deal with large HSDF graphs?
Slack-based Merging Heuristic

Reduced-size graph

How to transform HSDF into real-time tasks?
Timing Parameter Extraction Heuristic

Arbitrary-deadline tasks

How to map efficiently on multi-core platform?
Communication-aware mapping Heuristic

System

Applications

- Dataflow
  represented as SDF graphs
  WCET, P/C rates, Throughput, Latency

- Traditional
  Real-time Tasks
  Arbitrary-deadline tasks

Unified modelling

Chapter 4:
Graph reduction technique

Chapter 5:
Timing Parameter Extraction

Chapter 6: Communication-aware Mapping
using traditional real-time schedulers

Multi-core Platform

Reduced-size HSDF
Communication Modelling
Reduce-size HSDF with communication
Overview

1. Introduction
2. Background
3. Problem
4. Proposed Solution
5. Contributions
6. Evaluation
7. Conclusion
8. Future Work
Evaluation

- We experimented the complete approach using SDF$^3$ benchmark applications through three experiments that evaluate the:
We experimented the complete approach using SDF$^3$ benchmark applications through three experiments that evaluate the:

1. **Communication modelling** methodology of the communication-aware mapping algorithm through the testing of the communication cost and its effect on the schedulability of the system.
We experimented the complete approach using SDF$^3$ benchmark applications through three experiments that evaluate the:

1. **Communication modelling** methodology of the communication-aware mapping algorithm through the testing of the communication cost and its effect on the schedulability of the system.

2. **SPF** mapping heuristic **against** the well-known FF bin-packing heuristic.
We experimented the complete approach using \textit{SDF}^3 \textbf{benchmark} applications through \textbf{three experiments} that evaluate the:

1. \textbf{Communication modelling} methodology of the communication-aware mapping algorithm through the testing of the communication cost and its effect on the schedulability of the system.
2. \textbf{SPF} mapping heuristic \textbf{against} the well-known \textbf{FF} bin-packing heuristic.
3. \textbf{Complete approach} to show the trade-off between using original and merged HSDF graphs in terms of number of allocated applications and the overall run-time of the complete approach.
Evaluation of the communication cost.

- direct relation between the number of allocated applications and the availability of communication resources.
Evaluation of the communication cost.

- **direct relation between the number of allocated applications and the availability of communication resources.**

- **Ignoring communication cost allows mapping up to 76% more applications (infinite case), which gives a wrong perception of the ability to map applications with timing constraints.**
SPF surpasses FF in terms of number of allocated applications and run-time that reaches up to a maximum of 28% and 22%, respectively.
Enhance the run-time achieving a reduction in the overall system design time that ranges from 82% to 90%.
Evaluation

Evaluation of complete approach.

- Enhance the run-time achieving a reduction in the overall system design time that ranges from 82% to 90%.
- Less 12% in number of allocated applications.
Conclusion

- The complete approach is implemented under the SDF$^3$ tool.
The complete approach is implemented under the $\text{SDF}^3$ tool. This work proposes a complete approach with a main goal to integrate mixed application models with timing requirements on the same multi-core platform.
The complete approach is implemented under the SDF$^3$ tool.

This work proposes a complete approach with a main goal to integrate mixed application models with timing requirements on the same multi-core platform.

Proposed three algorithms:
Conclusion

- The complete approach is implemented under the SDF$^3$ tool.
- This work proposes a **complete approach** with a main goal to integrate mixed application models with timing requirements on the same multi-core platform.
- Proposed three algorithms:
  - **Slack-Based Merging**, which addresses exponential explosion in HSDF graphs (ACM TODAES 2017).
Conclusion

- The complete approach is implemented under the SDF$^3$ tool.
- This work proposes a complete approach with a main goal to integrate mixed application models with timing requirements on the same multi-core platform.
- Proposed three algorithms:
  1. **Slack-Based Merging**, which addresses exponential explosion in HSDF graphs (ACM TODAES 2017).
  2. **Timing Parameter Extraction**, which transforms HSDF graphs into periodic arbitrary-deadline tasks, to enable applying real-time scheduling and analysis techniques. (PDP 2015)
Conclusion

• The complete approach is implemented under the SDF$^3$ tool.
• This work proposes a complete approach with a main goal to integrate mixed application models with timing requirements on the same multi-core platform.
• Proposed three algorithms:
  1. Slack-Based Merging, which addresses exponential explosion in HSDF graphs (ACM TODAES 2017).
  2. Timing Parameter Extraction, which transforms HSDF graphs into periodic arbitrary-deadline tasks, to enable applying real-time scheduling and analysis techniques. (PDP 2015)
  3. Communication-Aware Mapping, which maps mixed application models taking into account the communication cost. (RTCSA 2013)
Conclusion

- The complete approach is implemented under the SDF$^3$ tool.
- This work proposes a **complete approach** with a main goal to integrate mixed application models with timing requirements on the same multi-core platform.
- Proposed three algorithms:
  1. **Slack-Based Merging**, which addresses exponential explosion in HSDF graphs (ACM TODAES 2017).
  2. **Timing Parameter Extraction**, which transforms HSDF graphs into periodic arbitrary-deadline tasks, to enable applying real-time scheduling and analysis techniques. (PDP 2015)
  3. **Communication-Aware Mapping**, which maps mixed application models taking into account the communication cost. (RTCSA 2013)
- The **three algorithms** together **guarantee** that the system is schedulable and the timing constraints of the applications are satisfied.
Future Work

- find the necessary time-constrained paths in the graph that are critical for correct execution that satisfies timing constraints.
find the necessary time-constrained paths in the graph that are critical for correct execution that satisfies timing constraints.

improve the communication model to check the feasibility of the communication while mapping tasks on the platform.
Future Work

- find the necessary time-constrained paths in the graph that are critical for correct execution that satisfies timing constraints.
- improve the communication model to check the feasibility of the communication while mapping tasks on the platform.
- consider a real-time communication model that incorporate fixed-priority for scheduling messages on the IN.
Thanks for your attention.
Questions?