Implementation of Real-Time Data-Flow Synchronous Programs on MPPA

Matheus Schuh
PhD CIFRE Candidate
Started 21/02/2019

Academic Supervisors: Claire MAIZA
Pascal RAYMOND

Industrial Supervisor: Benoît Dupont de DINECHIN
Outline

1. Introduction
2. Code generation of Synchronous Programs
3. Many-Core Response Time Analysis
4. Real-Time Operating Environment and Communication
This talk

- Past work from Amaury Graillat\(^1\)
  - Parallel Code Generation of Synchronous Programs for a Many-core Architecture

- Past work from Hamza Rihani\(^1\)
  - Many-Core Timing Analysis of Real-Time Systems and its application to an industrial processor

- Overview of future work inside the scope of my thesis
  - Real-Time Operating Environments for Models of Computation Annotated with Logical Execution Time

\(^1\)CAPACITES Project
Basic concepts

Real-Time Systems

- A system that must provide valid outputs before a deadline
- Time-critical: timing constraints are part of the specification
- Soft/Hard Real-Time: according to criticality of application
Basic concepts

Synchronous Data-Flow languages

- Network of nodes
- Dependencies and thus order requirements
- Lustre (academic), SCADE (industrial)
Outline

1. Introduction

2. Code generation of Synchronous Programs

3. Many-Core Response Time Analysis

4. Real-Time Operating Environment and Communication
Lustre/SCADE ensures formal semantics and determinism
C generated code inherits these properties
Static schedule given by synchronous programs
WCET$^2$ analysis checks the schedulability
Sequential execution

$^2$Worst Case Execution Time
Lustre/SCADE ensures formal semantics and determinism
C generated code inherits these properties
Static schedule given by synchronous programs
WCET\(^2\) analysis checks the schedulability
Sequential execution

Parallel execution in many-core environments is the challenge

\(^2\)Worst Case Execution Time
Many-Core Code Generation

Extraction of parallelism

- Generation of sequential code for each node
- 1 node $\rightarrow$ 1 runnable

Interaction between nodes

- Instantaneous communication
  - Copy output to input
  - Notify communication channel
- Delayed communication (pre operator)
  - Use of a double buffer and scheduling constraints

Synchronization

- Dependencies are compiled into blocking waits

What about real-time guarantees with parallel execution?
# Many-Core Code Generation

## Extraction of parallelism
- Generation of sequential code for each node
- 1 node $\rightarrow$ 1 runnable

## Interaction between nodes
- Instantaneous communication
  - Copy output to input
  - Notify communication channel
- Delayed communication (\texttt{pre} operator)
  - Use of a double buffer and scheduling constraints
- Synchronization
  - Dependencies are compiled into blocking waits
Many-Core Code Generation

Extraction of parallelism
- Generation of sequential code for each node
- 1 node → 1 runnable

Interaction between nodes
- Instantaneous communication
  - Copy output to input
  - Notify communication channel
- Delayed communication (\texttt{pre} operator)
  - Use of a double buffer and scheduling constraints
- Synchronization
  - Dependencies are compiled into blocking waits

What about real-time guarantees with parallel execution?
Outline

1. Introduction
2. Code generation of Synchronous Programs
3. Many-Core Response Time Analysis
4. Real-Time Operating Environment and Communication
Interference and reaction time

- Single-Core
  - WCET is sufficient
- Many-Core
  - WCET + interference on shared resources = WCRT\(^3\)
- WCRT
  - Most precise approach is too complex
  - Naive approach is too pessimistic
- Time analysis is made based on
  - Knowledge of hardware (MPPA2 / MPPA3)
  - Knowledge of software (Synchronous data-flow)
  - Hypothesis of time-triggered execution

\(^3\)Worst Case Response Time
Input = Isolated WCET + WA + Initial scheduling/mapping

1 Estimate current interference
   1.1 For all tasks that interfere on the current scheduling
   1.2 Add this interference delay to the initial WCET

2 If phase 1 differs after previous step
   2.1 Reajust release dates and restart 1 with the new scheduling

3 Check final schedulability
   3.1 If the application is schedulable: the current schedule is guaranteed
   3.2 Otherwise, go back to the initial scheduling/mapping step
Multi-Core Interference Analysis (MIA) Tool

Method

Input = Isolated WCET + WA + Initial scheduling/mapping

1 Estimate current interference
   1.1 For all tasks that interfere on the current scheduling
   1.2 Add this interference delay to the initial WCET

2 If phase [1] differs after previous step
   2.1 Readjust release dates and restart [1] with the new scheduling

3 Check final schedulability
   3.1 If the application is schedulable: the current schedule is guaranteed
   3.2 Otherwise, go back to the initial scheduling/mapping step
Method

Input = Isolated WCET + WA + Initial scheduling/mapping

1 Estimate current interference
   1.1 For all tasks that interfere on the current scheduling
   1.2 Add this interference delay to the initial WCET

2 If phase [1] differs after previous step
   2.1 Readjust release dates and restart [1] with the new scheduling

3 Check final schedulability
   3.1 If the application is schedulable: the current schedule is guaranteed
   3.2 Otherwise, go back to the initial scheduling/mapping step
Multi-Core Interference Analysis (MIA) Tool

0. Input (Isolated WCET)

1. Estimate current interference

2. Reajust release dates

3. Check schedulability

Matheus Schuh
Model of Computation

Banked Memory
- One bank for each core containing code, input buffer and local variables
- Execute in a local bank, write to a remote bank
- Interference on communication only

Task activation
- Time-triggered execution
- MIA tool calculates release dates respecting data dependencies and timing constraints

Platform
- Bare metal
- Non-preemptive static schedule
- Mapping between runnables and cores done by external tool
Framework Overview

Contribution

External tool

Parallelism Extraction

functional code

N1.c N2.c N3.c
N4.c N5.c N6.c

Mapping +
Non-preemptive
scheduling

Code Generation
System + Communication

Executable for
Kalray

WCET Analysis

Network Calculus
(WCTT)

Release dates

MIA

Communication and Dependency graph

NoC Routing + Rate Attribution
Outline

1. Introduction
2. Code generation of Synchronous Programs
3. Many-Core Response Time Analysis
4. Real-Time Operating Environment and Communication
Motivation

- Find the right abstraction level for efficient implementation of Real-Time applications
  - RTOS\textsuperscript{4}
  - High-level communication layer, such as DDS\textsuperscript{5}
  - Be more generic than bare metal
- Versatile model of computation
  - Lustre/SCADE
  - Simulink
  - LET, such as Giotto
  - Mixed criticality

\textsuperscript{4}Real-Time Operating System
\textsuperscript{5}Data Distribution Service
From MPPA2 to MPPA3

- Larger local memory
  - Allow RTOS implementation and replication
- Different memory bus arbiter
  - New modelisation of access timing
- Simpler NoC
  - More predictable inter-cluster communication
- Future work
  1. Port of code generation framework
  2. Evolution of response time analysis software
  3. RTOS and communication impact

\(^6\)Network-On-Chip
Thanks for your attention!

Questions?

You can find me at
matheus.schuh@univ-grenoble-alpes.fr
http://www-verimag.imag.fr/~schuhm/

Most of the images were extracted from thesis manuscripts and slides of Amaury and Hamza