MPPA and its use on Real-Time Systems

Matheus Schuh PhD CIFRE Candidate
1^{st} year

Academic Supervisors: Claire MAIZA
Pascal RAYMOND

Industrial Supervisor: Benoît Dupont de DINECHIN

Outline

1 Introduction

2 Framework for Code Generation of Synchronous Programs

3 Related Work

4 Evolution of MIA tool

5 MPPA3 modeling

6 Conclusion

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 ongoing work of my thesis
  - Real-Time Operating Environments for Models of Computation Annotated with Logical Execution Time
  - Related work
  - MIA evolution
  - MPPA3 modeling

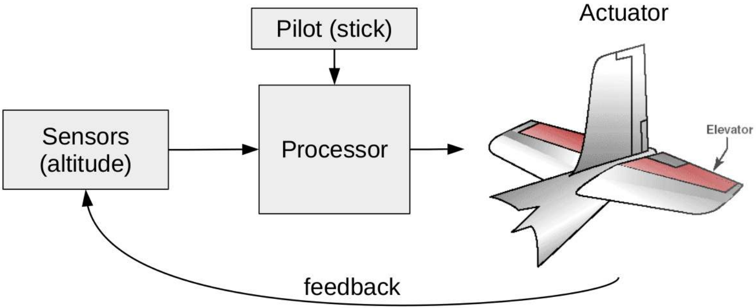
---

[1] CAPACITES Project

Basic concepts

Real-Time Systems

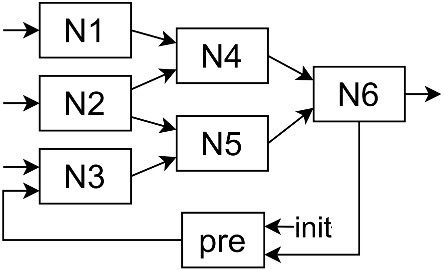
- A system that must provide valid outputs before a deadline
- Time-critical: timing constraints are part of the specification
- Soft/Hard Real-Time: according to criticality of application



Basic concepts

Synchronous Data-Flow languages

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



Outline

1 Introduction

2 Framework for Code Generation of Synchronous Programs

3 Related Work

4 Evolution of MIA tool

5 MPPA3 modeling

6 Conclusion

Single-Core Code Generation

- Lustre/SCADE ensures formal semantics and determinism
- C generated code inherits these properties
- Static schedule given by data-flow programs
- WCET[2] analysis checks the schedulability
- Sequential execution

---

[2] Worst Case Execution Time

Single-Core Code Generation

- Lustre/SCADE ensures formal semantics and determinism
- C generated code inherits these properties
- Static schedule given by data-flow 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 → 1 runnable

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 (`pre`/`fby` operator)
  - 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 (`pre`/`fby` operator)
  - Double buffer and scheduling constraints
- Synchronization
  - Dependencies are compiled into blocking waits

What about real-time guarantees with parallel execution?

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
- Timing analysis is made based on
  - Knowledge of hardware: MPPA
  - Knowledge of software: Synchronous Data-Flow
  - Hypothesis of time-triggered execution
- Multi-Core Interference Analysis (MIA) tool

---

[3] Worst Case Response Time

Framework Execution Model

Platform

- Bare metal
- Mono-rate non-preemptive static schedule
- Mapping between runnables and cores done by external tool

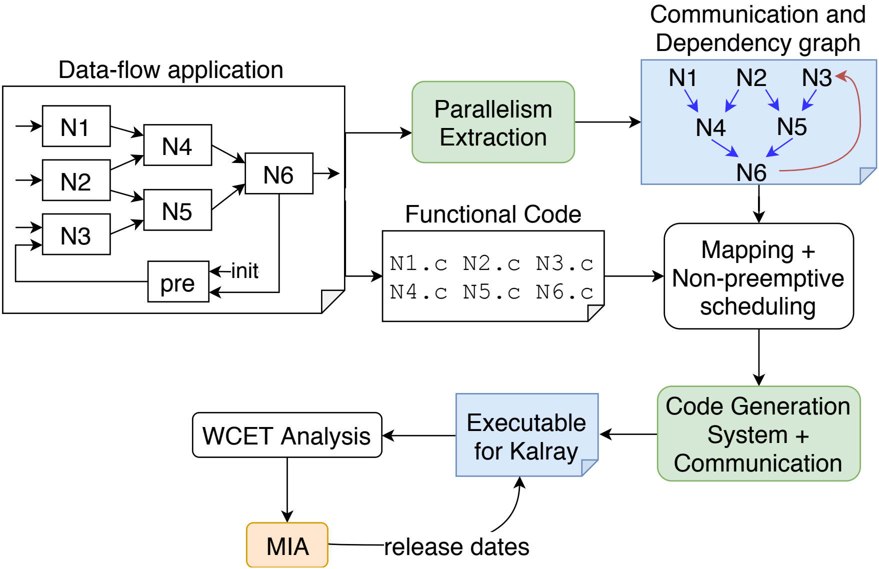
Task activation

- Time-triggered execution
- MIA: release dates respecting data dependencies and timing

Banked Memory

- One bank for each core: code, input buffers and local variables
- Execute in a local bank, write to a remote bank
- Interference on communication only

Framework Overview



Outline

1 Introduction

2 Framework for Code Generation of Synchronous Programs

3 Related Work

4 Evolution of MIA tool

5 MPPA3 modeling

6 Conclusion

Event-triggered vs Time-Triggered

Event-Triggered

- Tasks start as soon as their dependencies are satisfied
- Good for high performance
- May introduces temporal indeterminism

Time-Triggered

- Total control of when tasks start
- Mainly done statically

Different approaches

- Temporal Isolation: *Quentin Perret*
  - Application domain: avionic
  - Phased execution that forces isolation
- Run-time adaptation: *Stefanos Skalistis*
  - Parallel interference-sensitive run-time adaptation mechanism
  - Based on the actual execution time of tasks
- Interference Delay into schedulability analysis: *Benjamin Rouxel*
  - Contention-aware scheduling strategies
  - Minimize the pessimism of the global response time
- Compiler-level Integration: *Dumitru Potop-Butucaru*
  - Real-time systems compilation
  - Allows interferences for better efficiency

Outline

1 Introduction

2 Framework for Code Generation of Synchronous Programs

3 Related Work

4 **Evolution of MIA tool**

5 MPPA3 modeling

6 Conclusion

Multi-Core Interference Analysis

Inputs

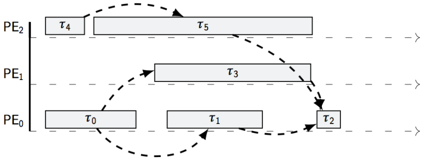
- Set of release date of all tasks
- Dependent tasks
- WCET in isolation + WC number of accesses

Main idea

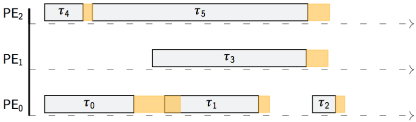
- Bounded interference
- Time-triggered execution

Original algorithm example

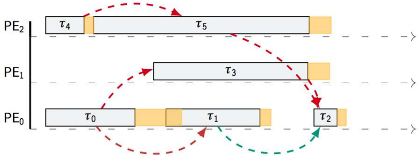
0. Input (Isolated WCET)



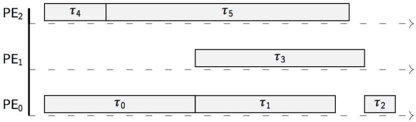
1. Estimate current interference



2. Adjust release dates



3. Check schedulability



Original algorithm in detail

Method

1. Start with initial release dates
2. Compute response times (1st fixed point) + interferences
3. Update the release dates
4. Repeat until no release date changes (2nd fixed point)

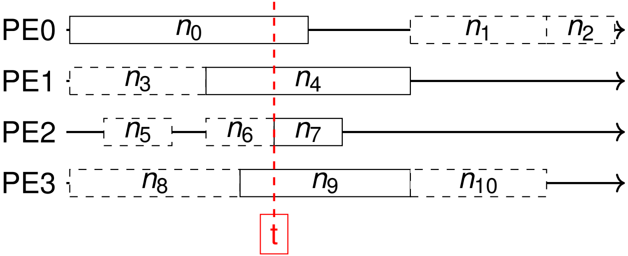
Original MIA

- Developed during Hamza thesis with this iterative algorithm
- Complexity of O(n^4)
  - Where n is the number of tasks
- Stopped converging for hundred of tasks
  - Scalability issues
- Written in C++

New interference calculation algorithm

- Accepted paper @ DATE 2020
- Complexity of O(n^2)
  - No nested loops within all tasks
  - No fixed-point iteration
- Scales to thousands of tasks
- Written in Python
- Collaboration with LIP
  - Matthieu Moy
  - Maximilien Dinechin

New algorithm example



Closed: n_6	t is after their finish date
Alive: n_0, n_4, n_9	t is between release date and finish date
Opening: n_7	t is at their release date
Future: n_1, n_2, n_{10}	t is before their finish date

New algorithm in detail

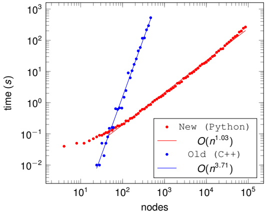
Method

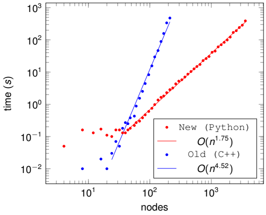
1. Start $t = 0$ and at each iteration jumps to the smaller value of:
    - The nearest end of alive tasks
    - The minimal release date of future tasks
2. Tasks with their dependencies satisfied are scheduled and the interference with alive tasks is calculated
    - They cannot interfere with dead tasks
    - Their interference with future tasks is yet to be computed
3. When a task is scheduled
    - Its release date is definitely set
    - Will not move with future tasks

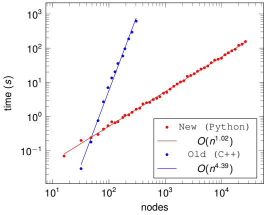
Complexity reduction

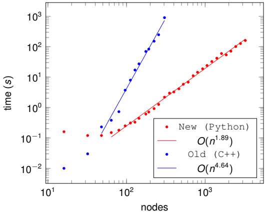
- Only tasks in the alive group need to be considered for interference calculation

Experimental Results


$LS = 4$

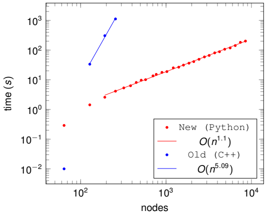

$NL = 4$


$LS = 16$

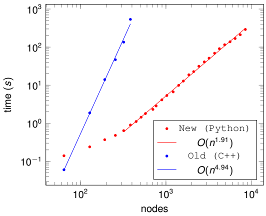

$NL = 16$

Experimental Results

$LS = 64$



$NL = 64$



Key numbers

- LS64 with 256 tasks
  - `C++`: 1121.79s × `Python`: 4.13s
  - **270** times faster
- NL64 with 384 tasks
  - `C++`: 535.24s × `Python`: 0.9s
  - **593** times faster

Outline

1 Introduction

2 Framework for Code Generation of Synchronous Programs

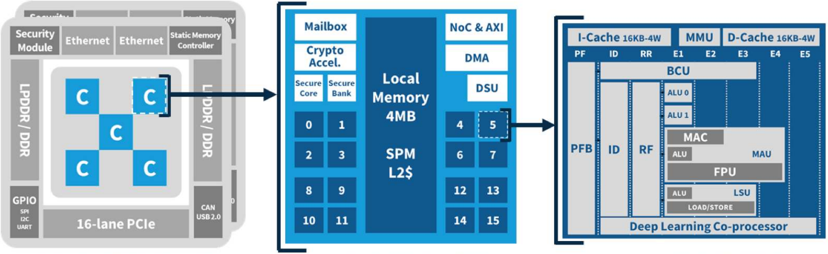
3 Related Work

4 Evolution of MIA tool

5 **MPPA3 modeling**

6 Conclusion

Coolidge overview



COOLIDGE PROCESSOR

5 compute clusters at 1200 MHz
2x 100Gbps Ethernet, 16x PCIe Gen4

COMPUTE CLUSTER

16+1 cores, 4 MB local memory
NoC and AXI global interconnects

6-ISSUE VLIW CORE

64x 64-bit register file
128MAC/c tensor coprocessor

Key modeling points

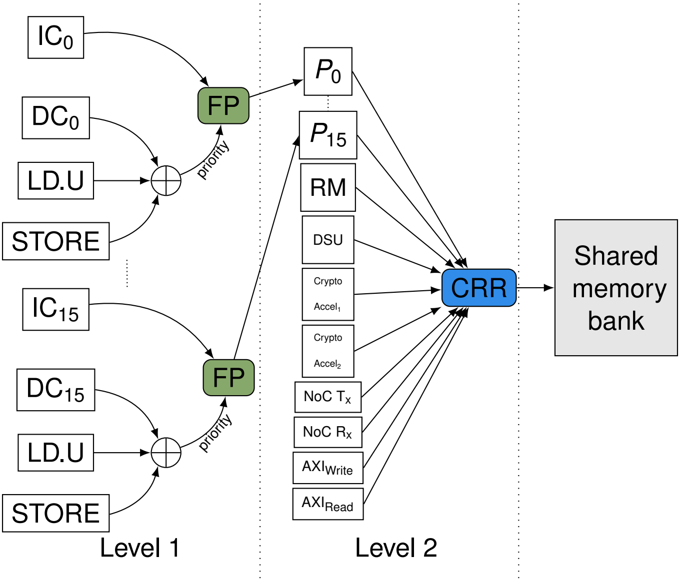
Intra-Cluster arbitration

- Cache L1 arbiter: **Fixed-Priority** for DC, LD.U and STORE
  - Code static analysis to determine longest DC interactions
- Shared Memory arbiter: **Configurable Round-Robin**
  - Per cluster configuration
  - Determines how many requests each initiator can issue at a time

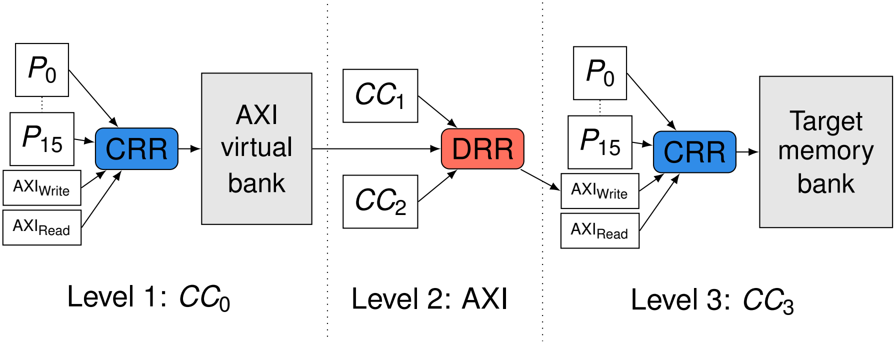
Inter-Cluster arbitration

- Interaction with DMA NoC on MPPA3 is different
- New Crossbar (AXI)
  - Point to point connection between clusters
  - **Deficit Round-Robin** arbitration at cluster arrival point

Intra-Cluster arbitration



Inter-Cluster arbitration



Difficulties

- New arbitration policies
  1. FP: Cache L1
  2. CRR: SMEM
  3. DRR: Crossbar
  - Timing analysis is harder
  - More caveats than a RR or TDMA
- Hardware was not ready yet (now it is!)
  - Simulator does not model these details
  - No way to verify the accuracy of our model

Outline

1. Introduction
2. Framework for Code Generation of Synchronous Programs
3. Related Work
4. Evolution of MIA tool
5. MPPA3 modeling
6. **Conclusion**

Thesis objectives

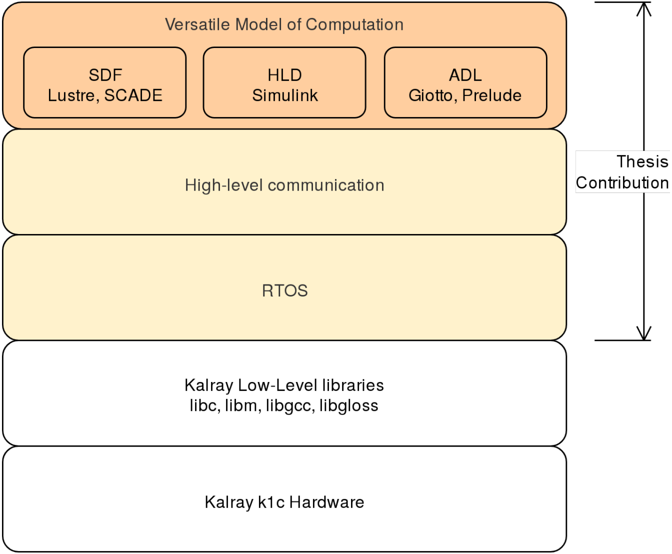
- Right abstraction level for efficient implementation of Real-Time applications
  - RTOS[4]
  - High-level communication layer, such as DDS[5]
  - More generic than bare metal w/o losing flexibility
- Versatile model of computation
  - Lustre/SCADE
  - Simulink
  - LET, such as Giotto
  - PREM (Predictable Execution Model)
  - Mixed criticality

---

[4] Real-Time Operating System

[5] Data Distribution Service

Revisited Framework Overview



Versatile Model of Computation

SDF
Lustre, SCADE

HLD
Simulink

ADL
Giotto, Prelude

High-level communication

RTOS

Kalray Low-Level libraries
libc, libm, libgcc, libgloss

Kalray k1c Hardware

Thesis Contribution

Ongoing/Future work

Ongoing

- PREM on MPPA2
- SCADE MPPA3 Integration

Future

- Experiments with RTOS tasks generation
- Possibly LET

Thanks for your attention!
Questions?

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