2022 IEEE Real-Time Systems Symposium (RTSS)

Edited by Benny Akesson

Organization

RTSS@Work chair

- Benny Akesson, ESI (TNO) and University of Amsterdam

Program Committee

- Matthias Becker, KTH Royal Institute of Technology, Sweden
- Federico Terraneo, Politecnico di Milano, Italy
- Zheng Dong, Wayne State University, USA
- Kilho Lee, Soongsil University, South Korea
- Behnaz Pourmohseni, Bosch, Germany
- Patricia Derler, Palo Alto Research Center (PARC), USA
- Anaïs Finzi, TTTech Computertechnik, Austria

Message from the RTSS@Work Chair

Welcome to RTSS@Work 2022, the open demo session organized as a part of the 43rd IEEE Real-Time Systems Symposium, held in Houston, Texas on December 6, 2022.

The goal of RTSS@Work is to provide a platform for researchers to present prototypes, tools, simulators, and systems, which extend the state-of-the-art in Real-Time Technologies and Techniques. It augments the traditional forum by enabling presenters to demonstrate working systems, thereby allowing them to directly engage with the audience, generate interest in new research topics, and encourage wider adoption of common frameworks.

This year's RTSS@Work is very special, as it is the first physical instance after years of virtual events due to the COVID pandemic. It is nice to meet each other again and to physically demonstrate and discuss the work we have done. While the COVID pandemic reduced the number of submissions in previous years, I am happy to announce that we had nine demo submissions this year, on par with the pre-pandemic era. The program committee, comprising seven researchers, selected eight demos to appear in the session.

I would like to conclude by thanking the program committee for accepting my invitation and for spending their time reviewing and selecting the demo abstracts. I would also like to thank the authors for submitting to RTSS@Work, for delivering the camera-ready abstracts on time, and for demonstrating their work at the conference.

Benny Akesson, ESI (TNO) and University of Amsterdam, the Netherlands
RTSS@Work 2022 Chair

Contents

.

Demonstrating The MAAT REQ Tool: Using Algebraic Process Models To Support Time-Sensitive Requirements Design

Boutheina Bannour*, Arnault Lapitre*, Patrick Tessier*, Guillaume Giraud†
* Université Paris-Saclay, CEA, List, Palaiseau, France
† PES R&D Department, RTE

***Abstract*— In this work, we implement a process algebra-based framework for real-time requirements analysis into the MAAT REQ tool. MAAT REQ provides an automatic transformation of structured textual requirements into process algebra and exploits it in the exploration of their intended real-time behaviors and concurrency. The tool's capabilities of requirement assessment are demonstrated on a practical example of an alarm system.**

I. Motivation and contribution

Formulating requirements that accurately describe real-time behaviors, and eliminating misunderstandings, is a crucial but difficult task for designers. The complexity of real-time requirements has made the use of formal methods widespread to check their desired properties and clarify their formulation. In a previous work [1], we proposed a framework based on requirements structured following EARS [6] templates, following recommendations from the International Council on Systems Engineering (INCOSE). Real-time details are introduced to refine event-driven, state-driven system behaviors. We utilize such information to aggregate (functional) requirements' behaviors into Process Algebra (PA) models and thus analyze them by simulation or formal validation. Compared to related works on formalizing requirements using PA [1], we go a step further by considering real-time aspects. The central component of [1] is an implementation of a PA using SAT solving based on our know-how of the symbolic execution tool DIVERSITY [5], relevant in the presence of constraints mixing time and data. We have not investigated zone-based techniques usually undertaken with Timed Automata (TA) for their run-time efficiency, which motivates the following contributions.

- Implementation a Clocked PA based on work [2] which states links with TA and zone-based handling of time and clocks,
- Enhanced EARS syntax with time details that directly map onto the Clocked PA,
- Redesign of the prototype of [1] into the MAAT REQ tool, which provides UML modeling of requirements, PA, and connections to SAT solving together with zone-based techniques.

We give in next section some technical insights on our contributions, yet many details are left to the demonstration. The overall architecture of the tool MAAT REQ is shown in Fig.1. The tool provides an advanced textual editing for structured requirements based on LSP [7]. We have created a requirement UML metamodel and a process UML metamodel with modeler Papyrus [3]. First, we parse the textual requirements to obtain the requirements model. Then, we perform a model-to-model transformation to get the process model. After that, analyzes are performed on this process model. A feedback is produced on the consistency of the entry requirements. Thus, the user can revise them and repeat the analyzes. The tool developments are service-oriented, and we use OSGi [4] to orchestrate the services.

II. Technical insights

Clocked Processes [2] are defined by the following syntax:

$$P ::= \{x := 0\}_{x \in R}\ \psi \triangleright +_{i \in I}(\phi_i, a_i).P_i \mid P_1 \ [\![A]\!]\ P_2 \mid K$$

The prefix process $(\phi_i, a_i).P_i$ performs action a_i guarded by clock constraint ϕ_i and then behaves like process P_i. The operator $+_{i \in I}$_ denotes a choice among processes. The operator _$[\![A]\!]$_ denotes the parallel composition of processes that synchronize on actions from A. The construct $\{x := 0\}_{x \in R}\ \psi \triangleright +_{i \in I}(\phi_i, a_i).P_i$ defines: a set of clocks from R to be reset, and a clock invariant ψ that has to be satisfied on time passing for the execution of every action a_i in the sum. This notion comes from TA and requires static handling to evaluate the process. K permits to call a process definition of the form $K = P$, where K is a unique process identifier. A specification consists of a main process and a collection of process definitions.

Listing 1: Processes of the Alarm System

```
1  AlarmSystem = //timed with clock x
2   {} ⊤ ▷ (⊤, set_button_is_pressed).
3   {x := 0} x ≤ 60 ▷ (x = 60, activate_alarm).
4   {} ⊤ ▷ (⊤, motion_detected).
5   {x := 0} x ≤ 0 ▷ (⊤, emit_tone).(
6    {x := 0} x ≤ 300 ▷ (
7     (⊤, alarm_is_disarmed).
8      {x := 0} x ≤ 0 ▷ (⊤, tone_off).AlarmSystem
9   + //choice op.
10    (x = 300, tone_off).
11     {} ⊤ ▷ (⊤, alarm_is_disarmed).AlarmSystem )
12  [[{alarm_is_desarmed}]] //parallel op.
13   HandleSecurity )
14
15 HandleSecurity = //timed with clock y
16  {y := 0} y ≤ 60 ▷ (
17   (⊤, alarm_is_disarmed).0 // 0 as inactive proc.
18  + //choice op.
19   (y = 60, alert_emergency_center).HandleSecurity )
```

List.1 shows the **demonstration example** of the Alarm System (AS). AS is started by pressing the set button (line 2). The alarm is activated after 60s to allow for some time to leave the area (line 3). When motion is detected (line 4), the alarm sounds (line 5) until AS is disarmed or reaches the end of the alarm duration of 300s (lines 6-11). In addition,

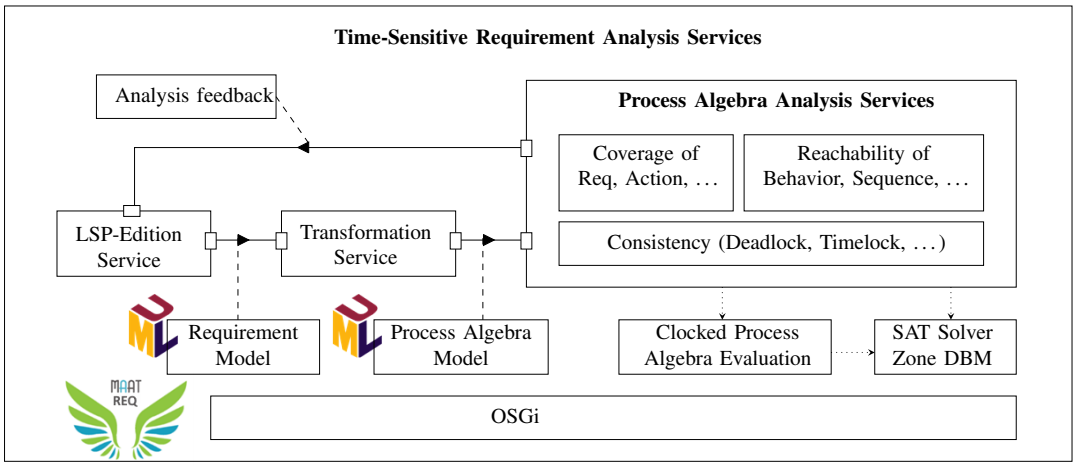


Fig. 1: MAAT REQ

if AS is not disarmed, then the emergency center is alerted every 60s until AS is disarmed (sub-process, lines 15-19).

Process evaluation. Clocked Processes have been given operational semantics using theoretical regions automata [2], which can be implemented efficiently using zones. Zones are sets of clock constraints and thus represent symbolically all valuations that satisfy these constraints without enumeration. We integrate the implementation of zones by the DBM (Difference Bound Matrix) data structure provided by the TA model-checker UPPAAL [8].

Our implementation generates an execution tree depicted in Fig.3. The tree construction starts with an initial node EC_0 (EC<id:0,h:0,step:1>, EC for Execution Context), which contains the main process to execute and an initial zone Z_0. Next, Z_0 is substituted by $reset(Z_0, R)$. Roughly speaking, clocks from R to reset are computed by static analysis s.t. all parallel processes synchronize on clock resets: e.g., $R = \{x, y\}$ when execution reaches the parallel composition (line 12) s.t. left (resp. right) process resets clock x (resp. y). Then, $up(Z)$ represents all states reachable from a zone Z by time elapsing that is compatible with process invariant set INV, $up(Z) \wedge_{\psi \in INV} \psi$ (time elapsing cannot invalidate either invariant of parallel processes). For previous case, $INV = \{x \leq 300, y \leq 60\}$ is issued from left (resp. right) process invariant $x \leq 300$ (resp. $y \leq 60$). This step is associated an EC in the tree: EC_2 obtained from context EC_0, EC_{15} obtained from context EC_{13}, etc. Next, an EC is computed for each follow-up discrete action a under satisfiable guard ϕ, i.e., $Z \wedge \phi$ is a zone with a non-empty set of valuations. The previous procedure is repeated on every EC of the tree. As executions can be infinite (due to process calls), stopping criteria on tree size can be set. Redundant behavior can be detected as well. Deciding whether or not an execution is redundant is done by testing zone inclusion and simplifying inactive processes.

Structured requirement. Requirement statements are provided by a grammar based on EARS [6]. Fig.2 shows the structured requirements of the Alarm System.

R1 : **when** set button is pressed, **the** system **shall** activate the alarm **immediately after** 60s

R2 : **after** the alarm activation, **when** motion is detected, **the** system **shall** emit a tone **immediately**

R3 : **after** tone emission, **inside time period** 300s, **when** the alarm is disarmed, **the** system **shall** turn off the tone **immediately** [**goto**] (R1)

R4 : **at end time period** 300s [**scope**] (R3), **the** system **shall** turn off the tone **immediately**

R5 : **after** tone off [**ref**] (R4), **the** system **shall wait for** the alarm to disarm [**goto**] (R1)

R6 : **after** tone emission, **inside time period** 60s, **the** system **shall wait for** the alarm to disarm

R7 : **at end time period** 60s [**scope**] (R6), **the** system **shall** alert the emergency center [**goto**] (R6)

Fig. 2: Structured requirements

A user-defined glossary, tailored to the needs of the requirement engineer, defines systems, triggers, and also equivalence for ease of use (e.g., "emit a tone"/"tone emission"). Each requirement statement is expressed by a -possibly complex- precondition, followed by a realization, specifying the system's action. Requirement behaviors are initiated when a triggering event occurs, signified with keyword **when**⟨ trigger ⟩ (e.g., R1, R2). *State-driven* requirements are active while the system is in a defined state. As the concern is timing, they are built with the keyword **inside time period** ⟨ period ⟩ (e.g., R3). We introduce details to enhance the sequencing of system behaviors: they can be triggered periodically through pattern **every** ⟨ period ⟩, subsequently to other behaviors through **after** ⟨ system response ⟩ / **at end time period** ⟨ period ⟩, or within some time slot (e.g., **within**⟨ time interval ⟩). R3 and R4 show that requirements can be complex and use several of these constructs at the same time,

and possibly with cross-requirement references. In [1], we outline the main transformation patterns into process algebra by taking advantage of the requirement structure. System responses sharing the same trigger are composed in parallel, and triggers can be non-deterministically produced upon a system response such that each (sub-)system is assumed to have an implicit initialization upon which its behavior occurs. State-driven requirements and systems responses with time details are transformed by various patterns involving clock reset, invariant, or guard constructs of the process algebra.

III. Conclusion

The objective of the demo is to present the capabilities of the MAAT REQ tool: requirement design, transformation into Process Algebra (PA) and analysis, applied on the Alarm System. We can test whether certain requirements can be covered at some point in the PA execution with traceability feedback. Moreover, we can detect various requirement inconsistencies that the tool can highlight as synchronization lacks, deadlocks, or timelocks in the PA execution. Finally so as to consider more requirements and vary their structure, we will consider other illustrative examples[1]. We are making our demo materials available online for the conference audience.

References

[1] M. Arnaud, B. Bannour, G. Giraud, and A. Lapitre. "Investigating Process Algebra Models to Represent Structured Requirements for Time-sensitive CPS". In: *33nd Int. Conf. on Software Engineering and Knowledge Engineering, SEKE*. KSI Research Inc., 2021.

[2] S. Cattani and M. Kwiatkowska. "A Refinement-based Process Algebra for Timed Automata". In: Formal Aspects Comput., 2005.

[3] Eclipse. *Papyrus Modeling Tool*. Accessed October 13, 2022. URL: `https://www.eclipse.org/papyrus/`.

[4] OSGi Working Group. *Open Services Gateway initiative (OSGi)*. Accessed October 13, 2022. URL: `https://www.osgi.org/`.

[5] CEA List Institute. *The DIVERSITY Tool*. Accessed October 13, 2022. URL: `https://projects.eclipse.org/proposals/eclipse-formal-modeling-project/`.

[6] A. Mavin, P. Wilkinson, A. Harwood, and M. Novak. "Easy Approach to Requirements Syntax (EARS)". In: *17th Int. Requirements Engineering Conf. RE*. IEEE, 2009.

[7] Microsoft. *Language Server Protocol (LSP)*. Accessed October 13, 2022. URL: `https://microsoft.github.io/language-server-protocol/`.

[1] A part of this work was supported by the PRISSMA project, co-financed by the French Grand Defi on Trustworthy AI for Industry. A part of this work was supported by European commission through CPS4EU project that has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 826276. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Hungary, Italy, Germany.

[8] Uppsala University and Aalborg University. *The UPPAAL Tool*. Accessed October 13, 2022. URL: `https://uppaal.org/`.

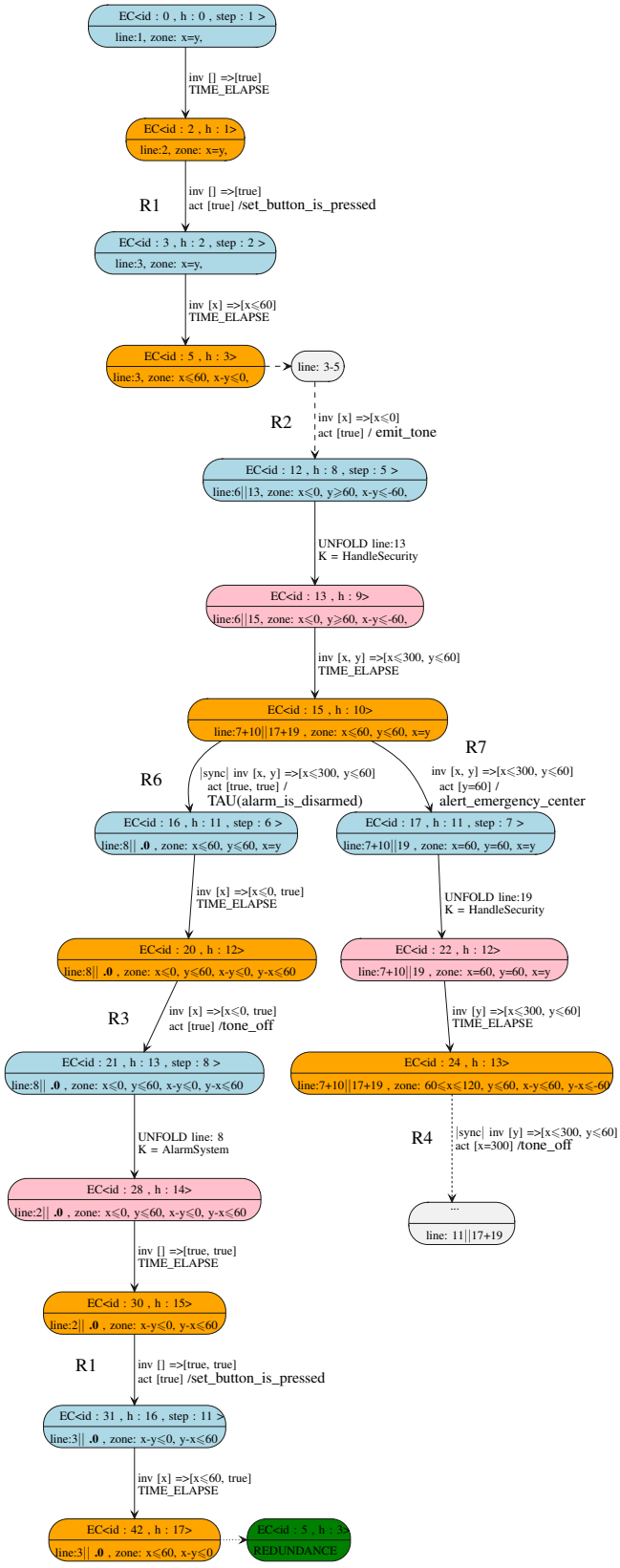


Fig. 3: Process evaluation for requirement coverage

An Open-Source Power Monitoring Framework for Real-Time Energy-Aware GPU Scheduling Research

Mohsen Karimi, Yidi Wang, and Hyoseung Kim
University of California, Riverside
mkari007@ucr.edu, ywang665@ucr.edu, hyoseung@ucr.edu

***Abstract*—General-purpose graphics processing units (GPUs) have received substantial interest from the real-time systems community as they can be much more powerful than CPUs on massively-parallelizable, data-intensive workloads. However, GPUs operations are usually power-intensive, and when used in systems with stringent power constraints such as automobiles and battery-powered devices, the use of GPUs makes power analysis difficult and can lead to power/energy budget violations. Although many of today's GPUs have onboard sensors to report their power consumption, they are usually slow and imprecise to be used for real-time GPU research, especially when observing the power consumption behavior of GPUs and developing scheduling techniques in dynamic workload scenarios. In this work, we present an open-source GPU power monitoring framework that measures the actual power consumption of GPUs at runtime with fast and precise responses. The framework can measure the voltage and current of multiple GPUs using an array of INA260 sensors and detect current and voltage changes as little as 1.5 mA and 1.5 mV, respectively, with sampling rates of up to 5 KHz per GPU. We believe this framework will help researchers no longer rely on inaccurate readings from onboard sensors to conduct real-time energy-aware GPU scheduling research.**

I. Introduction

GPUs, known for their parallel processing capabilities, are widely used in many applications ranging from image processing and machine learning tasks to complex simulations in computational physics. The high performance of GPUs introduces several additional considerations in system design, one of which is their high power consumption. However, it is challenging to analyze the exact power consumption of GPU-using tasks since the detailed architecture of commercial off-the-shelf (COTS) GPUs is not publicly open. While some manufacturers provide tools for power monitoring, such as onboard sensors and APIs for NVIDIA GPUs, they are not suitable for use under real-time scheduling scenarios with multiple GPU-using tasks, due to their low sampling rate and poor accuracy In this work, we design and implement an open-source GPU monitoring framework that is capable of measuring the precise power consumption of COTS GPUs. Our framework is non-intrusive and transparent to GPU workloads as it does not affect the performance of GPU tasks during power monitoring and does not require any modification to the application code.

II. Demo Description

High-end GPUs usually receive power from PCI Express (PCI-E) as well as directly from the power supply unit (PSU). To measure the entire power consumption of each GPU, both of these powers should be measured when the GPU is operating. To measure the current consumption from each power line, a sensor should be installed in series with the power line, which makes the implementation somewhat complicated. Specifically, for PCI-E power, we use a PCI-E extension cable [2] to access the power lines, and then cut the 12V line from the PCI-E extension cable and installed a sensor in series with the line. We chose INA260 [3] as our sensor to measure voltage and current as it provides a sufficient level of sampling rate and accuracy. A similar approach is used for the separate power coming from the PSU. Consequently, we can measure the current and voltage of each GPU using two sensors, the one for energy provided by PCI-E and the other for energy provided by the PSU. Raspberry Pi 3 (RPI3) is used to collect all the data from sensors via I2C and store them in the memory SD card as csv files. The data is then transferred to the PC for further processing. The connections for the sensors are shown in Fig. 1.

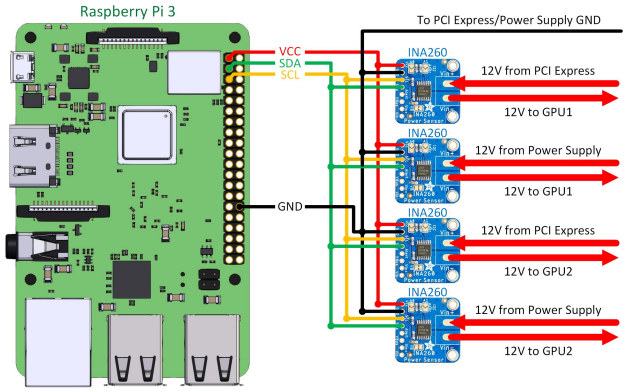


Figure 1: The connections of the sensors

We provide an open-source software library that is publicly available at [1]. We also provide an experimental setup to test the monitoring framework, which is also used in [4] to measure the power consumption of two heterogeneous GPUs, NVIDIA RTX 3070 and T400, under various real-time task execution scenarios. The setup for the experiment is shown if Fig. 2. In this setup, we are able to monitor the power consumption of the two GPUs at the same time by using four INA260 sensors. The RPI3 and PC are connected over WiFi and the Network Time Protocol (NTP) is used to synchronize time between RPI3 and PC. The data is stored in RPI3 as csv files containing the voltage and current of the sensors as well as the

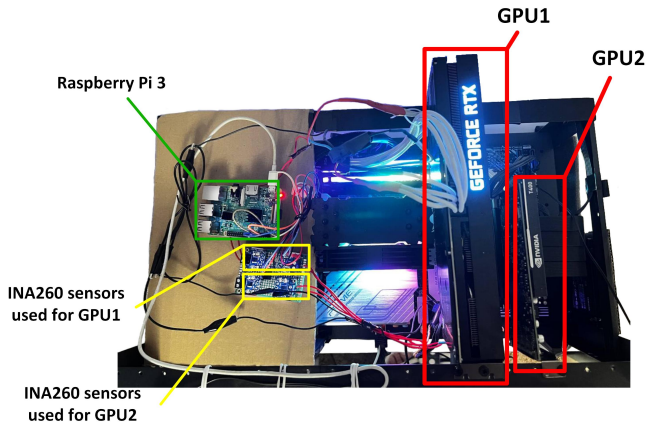


Figure 2: The hardware setup for experiment

timestamp of each sample in microseconds. The program is written in C language and tested on the Linux Kernel 5.15.56-v7+ running on RPI3. Each of the INA260 sensors is capable of sampling current and voltage at the maximum speed of 140 μs per sample. However, the bandwidth limitation of the I2C bus would make it challenging to receive data from multiple sensors at that speed. The operating system scheduler can also cause delays due to other tasks running on the system. Therefore, we set the priority of the task to *real-time* in Linux operating system and disabled unnecessary services such as GUI to reduce the delay caused by other tasks. We could achieve a sampling rate of more than 5K samples/s when recording voltage and current of two GPUs simultaneously, which corresponds to one sample every 200 μs. The block diagram of the connections for the setup with two GPUs is shown in Fig. 3.

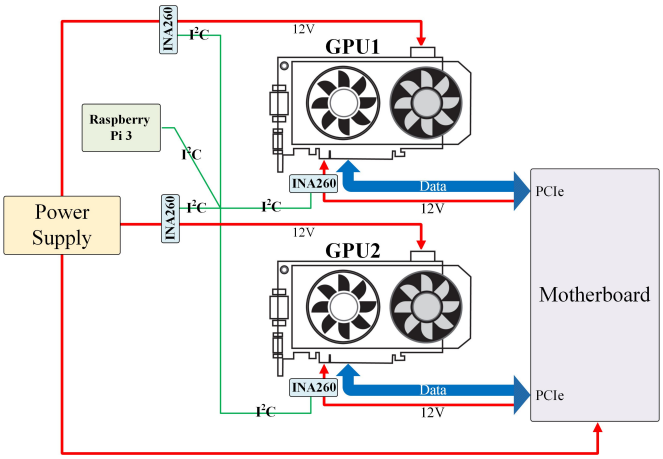


Figure 3: Block diagram of the connections for two GPU setup

To test the performance of the system, we sampled 1 million current and voltage samples and recorded the measurement time for each of the samples. Fig. 4 shows the histogram of measurement times for one million samples. The minimum, maximum, average, and 99th percentile of the measurement times are recorded as 168 μs, 224 μs, 177 μs, and 181 μs, respectively.

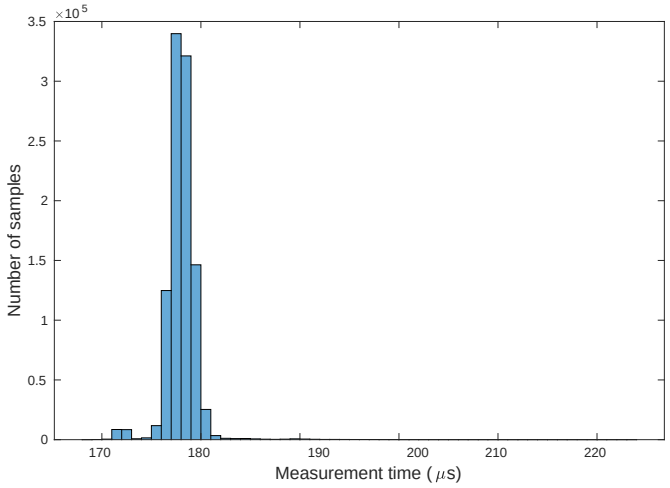


Figure 4: Histogram of measurement times

Although the current setup uses 4 sensors, our framework itself can be extended to up to 16 sensors (8 GPUs) for a single i2c bus; this limitation is imposed by the fact that the INA260 sensor can be configured to have up to 16 different I2C addresses by modifying the A0 and A1 connections of the sensor [3]. However, this limitation can be easily overcome by using multiple I2C buses and we expect our framework can be used for monitoring eight or more GPUs.

References

[1] GPU Power Monitoring System. https://github.com/rtenlab/gpu_power_monitoring.

[2] Corsair. Pcie 3.0 x16 extension cable. https://www.corsair.com/us/en/Products/p/CC-8900419.

[3] Texas Instrument. Ina260 36v, 16-bit, precision i2c output current/voltage/power monitor. https://www.ti.com/product/INA260.

[4] Y. Wang, M. Karimi, and H. Kim. Towards energy-efficient real-time scheduling of heterogeneous multi-gpu systems. In *2022 IEEE Real-Time Systems Symposium (RTSS)*, 2022.

A Scheduling and Analysis Tool for Parallel Real-Time Applications on Multicore Platforms

Morteza Mohaqeqi, Gaoyang Dai, Behnam Khodabandeloo, Petros Voudouris and Wang Yi
Uppsala University, Sweden
{morteza.mohaqeqi, gaoyang.dai, behnam.khodabandeloo, petros.voudouris, yi}@it.uu.se

***Abstract*—This paper presents a tool for the design and analysis of real-time systems specified as a set of reoccurring DAG tasks deployed on a heterogeneous multiprocessor platform. First, it allows designers to evaluate – by simulation and analytical methods – different scheduling policies using user-specified system configuration parameters such as the number of processor cores, memory access latency, and task parameters etc., and randomly generated task sets. Second for a system design consisting of a fixed hardware configuration, a task set and a scheduling policy, the tool will generate a run-time schedule providing performance real-time guarantees.**

***Index Terms*—timing analysis, scheduling simulator, parallel real-time tasks**

I. Introduction

Today, parallel architectures such as multi- and many-core processors have become ubiquitous in computing. In embedded domains, there is an increasing trend towards the usage of heterogeneous and parallel architectures for performance-demanding and real-time applications. Paradoxically, the introduction of performance enhancing architectural solutions, such as heterogeneous processor cores and multi-threading introduces a large degree of complexity and makes traditional single-core timing analysis approaches unsuitable for new architectures. This brought a significant challenge (and also a great opportunity) for embedded systems designers to explore the hardware parallelism. Our approach to facing this challenge is to provide a tool to implement and evaluate customized schedulers, and faithfully realize analytically-approved high-level task models.

Applications Areas. This tool is designed (and under development) for intended applications in safety-critical domains where high-performance and real-time requirements must be ensured. Typical application areas include automotive systems involving self-driving and 5G/6G networks, that are computationally demanding real-time systems deployed on heterogeneous multi-core and many-core platforms.

Main Features. The tool offers two features. First, it allows the designers to analyse the timing behavior and performance of a system (using either simulation and/or analytic methods) with (a large number of randomly generated) possible configurations of hardware and software components, and scheduling policies. The goal is to select and validate the potentially best system configuration and run-time scheduling policy. Second, for a given system configuration (a design), it shall provide a run-time schedule, with real-time and performance guarantees such as worst-case response times (WCRTs) and throughput.

The tool uses directed acyclic graphs (DAGs) to describe software components. A DAG [15] is a directed graph where nodes represent the sequentially executed sub-components of a software and edges represent precedence constraints and (or) input-and-output data relations between the nodes. In summary, the tool provides the following basic utilities for timing analysis of DAG tasks running on multicore systems.

- Simulating the execution of a set of periodic real-time DAGs on a specified hardware platform; based on that, timing and performance measures may be computed, and
- Analytical computation of a safe upper-bound on the WCRT of each task.

A user can specify the target hardware platforms with configuration parameters such as number and type of processor cores, as well as memory overheads. In addition, the mapping between software components onto hardware resources as well as the intended run-time scheduling policy among a set of pre-specified policies can be specified.

II. An Overview of the Tool

Figure 1 shows the main window of the tool, which contains

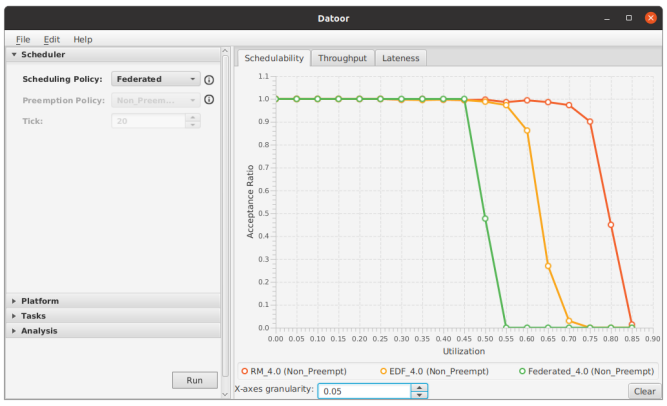


Fig. 1. The main window of the tool; comparing different schedulers.

two parts: configuration (the left-hand side), and results (the right-hand side). In the configuration part, the user can specify software and hardware configuration for which the timing

This work is supported in part by the European Research Council (ERC) and Knut and Alice Wallenberg Foundation (KAW) through projects ERC CUSTOMER (Grant 834166) and KAW UPDATE (Grant 20190134). We thank Duc Anh Nguyen for his efforts in the implementation.

analysis (simulation or analytical) is to be done. More details of this part is presented in subsequent sections. The result of the analysis is obtained in terms of performance and time-related measures (e.g., acceptance ratio) that are provided to the user through a number of charts. In the case of simulation-based analysis, if a system is determined unschedulable, the tool will output a deadline miss scenario.

The tool is implemented in Java and can be smoothly run in different environments.

III. Application Modelling

To model parallel real-time applications, the tool adopts the reoccurring task model where each task is represented by a period and a DAG of nodes and directed edges. The shared resource access of the tasks is modeled by a semaphore-like mechanism, where a job is supposed to be suspended once the required resource is not available. A simple priority-based protocol is considered, for which both simulation and analytical computation of WCRT have been developed.

To assess a timing/schedulability analysis approach, a set of (DAG) task sets are randomly generated according to the following *graph generation* and *period assignment* methods.

Random Graph Generation: Graph generation methods specify a method to create graph structures. In our tool, we have included three methods: the *layer-by-layer* approach [4], *series-parallel* graphs [13], and the *STR2RTS* benchmark [14].

Period Assignment: Period values can be generated using the *UUnifast* algorithm [1]. In addition, they can be chosen from a pre-defined set of values based on to two application areas: 5G networks [11], which contains the values of $\{0.125, 0.25, 0.5, 1\}$ ms, and automotive systems designed according to the AUTOSAR reference model [9]. An extended set of the automotive periods, specified in [12], is also added.

In addition to randomly generated task sets, the tool can analyze user-specified task sets. In this case, the structure and timing parameters of the DAGs, including periods and WCETs, are determined by the user.

IV. Supported Scheduling Algorithms

A set of well-known schedulers have been considered, including RM, EDF, and Federated [10], as well as Dynamic Federated [5] and Virtually-Federated [8], scheduling. For some scheduling policies, both analytical method and simulation can be done, while for others only one approach is available.

The scheduler can be chosen to be fully preemptive, preemptive at certain points (ticks), or non-preemptive. Further, it can be *global* or *partitioned*. Also, a partially partitioned scheduling is added, where a subset of the jobs are statically assigned to the cores, and the others are globally scheduled.

Figure 2 shows a sample schedule generated by the tool. The left-side panel shows the tasks and the obtained WCRTs.

V. Platform Description

Currently a platform description consists of mainly the number of processor cores and the memory access time, which

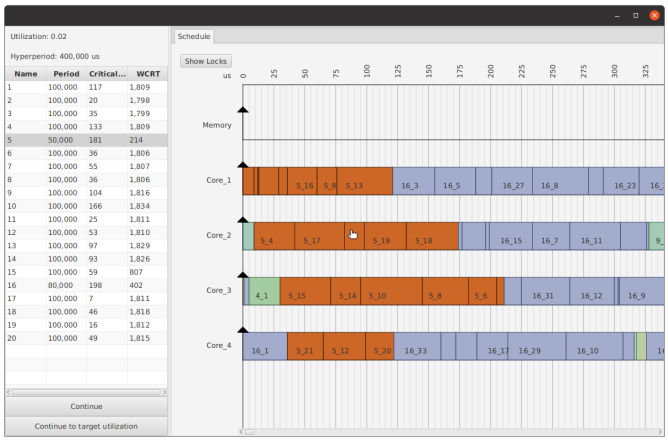


Fig. 2. The schedule obtained from simulation.

is specified in terms of three user-specified parameters: (1) the time for loading initial data from memory, (2) context-switch overhead incurred by any preemption, and (3) communication overhead, incurred by data transfer between any two consecutive nodes of a DAG locating on different cores. In addition, there may be a set of other resources shared by the tasks.

VI. Related Work

An open-source implementation of a number of DAG timing analysis methods has been provided in [16][1]. This is a relatively general framework, which considers a common implementation of DAG. This basic implementation is then used for implementing the corresponding schedulability tests. The tool comes with no graphical interface.

YARTISS [2] and SimSo [3] are two timing evaluation tools for multicore real-time systems. The focus of both is on timing analysis based on a simulation-based approach.

A modeling and development tool for multicore embedded systems has been proposed by the Eclipse APP4MC project [6]. Specifically, a simulation environment for analysis of such models has been developed [7]. While the project targets a general way for detailed description of multi-/many-core embedded systems, our tool gives a more focused and specialized facility for timing analysis of multicore real-time systems.

VII. On-going and Future Work

A compiler is currently under development, which, based on the inserted codes (currently Java and C) of DAG nodes and a system configuration, will generate a parallel threading software ensuring the validated timing and performance properties.

As future work, we are extending the tool to deal with heterogeneous platforms, as well as multiple languages to program the DAG tasks, and user-specified scheduling algorithms. Such algorithms may be added in one of the following two

[1]Available on GitHub: https://github.com/mive93/DAG-scheduling/blob/master/src/DAGTask/DAGTask.cpp

ways: (1) implementing the algorithm and integrating it with the tool source code, (2) describing new algorithms using a high-level language. We plan to open-source the tool, which facilitates the former.

REFERENCES

[1] Enrico Bini and Giorgio C Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, 2005.

[2] Younès Chandarli, Frédéric Fauberteau, Damien Masson, Serge Midonnet, and Manar Qamhieh. *Yartiss: A tool to visualize, test, compare and evaluate real-time scheduling algorithms*. PhD thesis, 2012.

[3] Maxime Chéramy, Pierre-Emmanuel Hladik, and Anne-Marie Déplanche. Simso: A simulation tool to evaluate real-time multiprocessor scheduling algorithms. In *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, 2014.

[4] Daniel Cordeiro, Grégory Mounié, Swann Perarnau, Denis Trystram, Jean-Marc Vincent, and Frédéric Wagner. Random graph generation for scheduling simulations. In *International ICST Conference on Simulation Tools and Techniques (SIMUTools)*, 2010.

[5] Gaoyang Dai, Morteza Mohaqeqi, and Wang Yi. Timing-anomaly free dynamic scheduling of periodic DAG tasks with non-preemptive nodes. In *RTCSA*, pages 119–128, 2021.

[6] Eclipse. APP4MC. https://projects.eclipse.org/projects/automotive.app4mc. Accessed: 2022-10-14.

[7] Eclipse. APP4MCsim. https://gitlab.eclipse.org/eclipse/app4mc/org.eclipse.app4mc.tools.simulation. Accessed: 2022-10-14.

[8] Xu Jiang, Nan Guan, Haochun Liang, Yue Tang, Lei Qiao, and Wang Yi. Virtually-federated scheduling of parallel real-time tasks. In *RTSS*, pages 482–494, 2021.

[9] Simon Kramer, Dirk Ziegenbein, and Arne Hamann. Real world automotive benchmarks for free. In *Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, 2015.

[10] Jing Li, Jian Jia Chen, Kunal Agrawal, Chenyang Lu, Chris Gill, and Abusayeed Saifullah. Analysis of federated and global scheduling for parallel real-time tasks. In *ECRTS*, pages 85–96, 2014.

[11] Shao-Yu Lien, Shin-Lin Shieh, Yenming Huang, Borching Su, Yung-Lin Hsu, and Hung-Yu Wei. 5G new radio: Waveform, frame structure, multiple access, and initial access. *IEEE communications magazine*, 55(6):64–71, 2017.

[12] Mitra Nasri, Geoffrey Nelissen, and Björn B Brandenburg. Response-time analysis of limited-preemptive parallel DAG tasks under global scheduling. In *ECRTS*, pages 21–1, 2019.

[13] Bo Peng, Nathan Fisher, and Marko Bertogna. Explicit preemption placement for real-time conditional code. In *Euromicro Conference on Real-Time Systems*, pages 177–188, 2014.

[14] Benjamin Rouxel and Isabelle Puaut. STR2RTS: Refactored streamIT benchmarks into statically analyzable parallel benchmarks for WCET estimation & real-time scheduling. In *International Workshop on Worst-Case Execution Time Analysis (WCET)*, 2017.

[15] Abusayeed Saifullah, David Ferry, Jing Li, Kunal Agrawal, Chenyang Lu, and Christopher D Gill. Parallel real-time scheduling of DAGs. *IEEE Trans. Parallel Distrib. Syst.*, 25(12):3242–3252, 2014.

[16] Micaela Verucchi and Marko Bertogna. A comprehensive analysis of dag tasks: solutions for modern real-time embedded systems. 2020.

Know your Enemy: Benchmarking and Experimenting with Insight as a Goal

Mattia Nicolella*, Denis Hoornaert†, Shahin Roozkhosh*, Andrea Bastoni†, and Renato Mancuso*
*Boston University, †Technische Universität München
*{mnico, shahin, rmancuso}@bu.edu, †{denis.hoornaert, andrea.bastoni}@tum.de

***Abstract*—Available benchmark suites are used to provide realistic workloads and to understand their run-time characteristics. However, they do not necessarily target the same platforms and often offer a diverse set of metrics, leading to the lack of a knowledge base that could be used for both systems and theoretical research. RT-Bench, a new benchmark framework environment, tries to address these issues by providing a uniform interface and metrics while maintaining portability. This demo illustrates how to leverage this framework and its recently-added features to improve the understanding of the benchmarks' interaction with its system.**

***Index Terms*—Benchmark, Profiling, Classification**

I. Introduction

Benchmarking plays an indispensable role in the real-time community to evaluate, consolidate, and validate novel research. Using benchmarks to evaluate the performance of production systems is also of great value as simulators only partially depict real platforms' behavior. Benchmarking is thus beneficial for many aspects of the real-time research community: from system research to theoretical system modeling.

For system research, understanding the run-time behavior of realistic benchmarks is crucial to assess performance gains and showcase novel system designs. In such cases, *local knowledge* such as the maximal memory activity during the application run-time is valuable. On the other hand, theoretical research can benefit from using workload characteristics obtained empirically to test scheduling and regulation mechanisms against realistic loads. Informed improvements in the quality of regulation mechanisms require access to a coherent database of measurements providing a *global knowledge* via the aggregation of several performance metrics.

Despite its importance, to date, no existing reference knowledge base provides such *local* and *global* knowledge. Unfortunately, this leaves the community to rely on personal knowledge or experience. The problem is (at least partially) imputable to the high fragmentation of benchmark suites. In fact, the most commonly used benchmark suites in the community differ in several aspects, including (1) system compatibility, (2) requirements, (3) measured metrics, and (4) reporting formats.

To address these issues, we have created a new benchmark framework called RT-Bench[1] that was initially presented in [1] and that we are continuously improving and expanding. As part of the continued effort in the project, we hereby present the first milestone towards creating and establishing a public knowledge base of benchmark performance and profile characteristics. In the context of this demo, we will illustrate how the expanding knowledge base can be exploited to gain *wider* and *deeper* understanding of the interaction between realistic benchmarks and underlying hardware. To this end, we will demonstrate how to use RT-Bench, and showcase its recently-added features[2] to profile, extract, and report benchmarks' characteristics. Specifically, in our demo, we will:

- Present the latest advancements in the capabilities of RT-Bench originally presented in [1] that now includes 67 benchmarks.
- Illustrate how to interpret the profiles of complex benchmarks using key performance metrics. As an example, the profile of a benchmark issued from the `image-filters`[3] suite is provided.
- Provide a first comprehensive overview and classification of benchmarks issued from several suites.

II. Focus of the Demo

This section presents and discusses the results of two classes of experiments to gain *global* and *local* knowledge about the benchmarks at hand. These experiments showcase the capabilities of RT-Bench to collect and export measurement data. In Section II-A, the *local* knowledge experiment leverages the capability of RT-Bench to simultaneously monitor several performance counters during the execution of a benchmark. On the other hand, the *"global knowledge"* experiments presented in Section II-B showcase the ability to directly contrast and classify benchmarks issued from different suites.

While RT-Bench is also compatible with `x86_64` Intel CPUs, our evaluation is based on ARM embedded platforms. In particular, we use a Raspberry Pi 3B+ to carry out the experiments. The system features (1) a four-CPUs (Cortex-A53) cluster operating at 1.5GHz[4], (2) per-core 32KB+32KB instruction and data caches, and (3) a 1MB shared last-level cache (LLC). We use a Linux kernel version 5.15.61, and RT-Bench applications have been compiled with GCC 10.2.1. The RT-bench benchmarks we used include the adapted version of the `image-filters` suite, the San-Diego Vision Benchmark Suite [2] (SD-VBS), and the TACLeBench suite [3].

[1]https://gitlab.com/rt-bench/rt-bench

[2]Comprehensive documentation on RT-Bench features can be found in the documentation: https://rt-bench.gitlab.io/rt-bench/

[3]https://gitlab.com/rt-bench/image-filters

[4]The frequency scaling governor is explicitly set to `performance`.

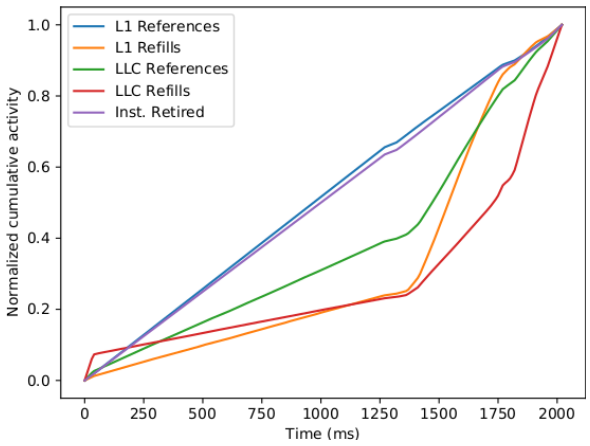


Fig. 1. Profiling of the `canny` image filter (see `image-filters` benchmark suite) via performances counters.

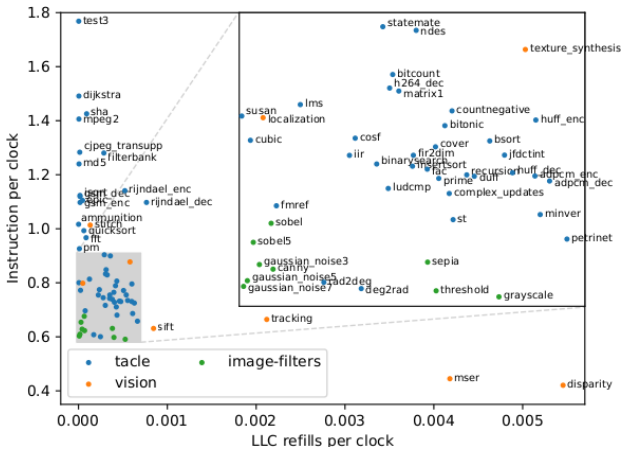


Fig. 2. Clustering of benchmarks and their suite based on instruction per clock (IPC) and last-level cache refills per clock (LLC-RPC).

A. Local Knowledge - Profiling

For this experiment, we selected the `canny` benchmark from the `image-filters` suite to illustrate the profiling and analysis capabilities provided by RT-bench. In fact, due to its six-pass filtering, the benchmark implementing the Canny algorithm [4] is an ideal candidate. The benchmark is run using the `vga` input size (i.e., 640×480 pixels) on a single core, and all the supported performance counters have been monitored. Fig. 1 displays the normalized cumulative activity for each performance counter.

Fig. 1 clearly illustrates the fluctuations in resource demand taking place during a benchmark's execution. While the evolution of the instructions retired and the L1-D cache references remain stable throughout, the trend of other performance counters increases at various rates, hinting at the existence of several execution phases with distinct characteristics. In this case, such profiling is particularly interesting for budget-based memory regulation mechanisms.

B. Global Knowledge - Benchmark Clustering

Using the aggregated performance counter activity and the timing measurements, it is possible to extract, compare, and classify benchmarks that belong to the various suites. Thanks to the wide choice of metrics reported, many classification criteria are available. Our demo will cover a handful of them. One such classification is reported in Fig. 2. Here, we showcase the relative positioning of the supported benchmarks w.r.t. their Instruction per Clock-cycles (IPC) and their Last-level Cache Refills per Clock-cycles (LLC-RPC). The figure is a visual representation of how CPU-bound vs. memory-bound the considered applications are.

Three observations can be made from Fig. 2. Firstly, only four benchmarks (all from SD-VBS) stand out w.r.t. how frequently they cause LLC refills. Secondly, all other benchmarks create less frequent LLC refills and are grouped together in a large cluster where only very few display an IPC higher than 1. Finally, most benchmarks lay in a dense low-IPC and low-refill frequency cluster. This cluster is mainly composed of TACleBench and image-filters benchmarks. The presence of the former can be explained by the small input data size used, whereas the latter can be explained by its reliance on costly floating point operations.

III. Conclusion

In this article, we demonstrate how the recent advancements in RT-bench help get further insights into benchmarks.

The authors of this article and members of the RT-Bench project are committed to maintaining and expanding the tools, the supported benchmarks, and the measurement database. Future extensions include the analysis of Artificial Neural Networks models powered by widely used libraries (e.g., Tensorflow) and the capacity to define task sets to release simultaneously.

Acknowledgment

The material presented in this paper is based upon work supported by the National Science Foundation (NSF) under grant number CCF-2008799. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the NSF. Andrea Bastoni and Denis Hoornaert were supported by the Chair for Cyber-Physical Systems in Production Engineering at TUM and the Alexander von Humboldt Foundation. In addition, Andrea Bastoni has been supported by EIT Urban Mobility, an initiative of the European Institute of Innovation and Technology (EIT), a body of the European Union.

References

[1] M. Nicolella, S. Roozkhosh, D. Hoornaert, A. Bastoni, and R. Mancuso, "Rt-bench: An extensible benchmark framework for the analysis and management of real-time applications," in *Proceedings of the 30th International Conference on Real-Time Networks and Systems*, 2022.

[2] S. K. Venkata, I. Ahn, D. Jeon, A. Gupta, C. Louie, S. Garcia, S. Belongie, and M. B. Taylor, "Sd-vbs: The san diego vision benchmark suite," in *2009 IEEE International Symposium on Workload Characterization*.

[3] H. Falk, S. Altmeyer, P. Hellinckx, B. Lisper, W. Puffitsch, C. Rochange, M. Schoeberl, R. B. Sørensen, P. Wägemann, and S. Wegener, "TACLeBench: A benchmark collection to support worst-case execution time research," in *WCET 2016*, M. Schoeberl, Ed., vol. 55. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2016, pp. 2:1–2:10.

[4] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, 1986.

Hardware Data Re-organization Engine for Real-Time Systems

Shahin Roozkhosh*, Denis Hoornaert†, Renato Mancuso* and, Manos Athanassoulis*
*Boston University †Technische Universität München
*{shahin, rmancuso, mathan}@bu.edu, †denis.hoornaert@tum.de

***Abstract*—Access patterns and cache utilization play a key role in the analyzability of data-intensive applications. In this demo, we re-examine our previous research on software-hardware co-design to push data transformation closer to memory from a real-time perspective. Deployed in modern CPU+FPGA systems, our design enables efficient and cache-friendly access to large data by only moving relevant bytes from the target memory. This (1) compresses the cache footprint and (2) reorganizes complex memory access patterns into sequential and predictable patterns.**

***Index Terms*—Memory Semantic, Data re-organization**

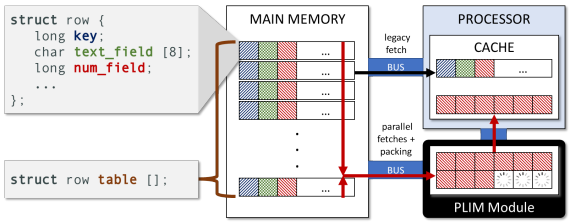


Fig. 1: On-the-fly data transformation enhancing data locality.

I. Introduction

One of the key bottlenecks in modern computing is moving data through the memory hierarchy to processing elements. The corresponding predictability issues are particularly problematic in real-time systems, especially when large-footprint applications exhibiting complex memory access patterns are considered. Multi-level caches have been introduced to hide the latency of memory fetches. They are effective in optimizing performance when data accesses are characterized by *spatial* and *temporal* locality.

Unfortunately, achieving spatiotemporal locality in large-footprint applications is challenging. Motivated by this, our recently published work [1] investigates the role of hardware-aided on-the-fly data reshaping for a specific class of large-footprint applications, i.e., database systems.

Relational databases typically store in-memory relations (tables) employing a row-oriented layout—offering good locality for transactional processing—or a column-oriented layout, with good locality for analytical processing. New applications, however, blend analytical and transactional processing. Therefore, no single optimal layout exists. At the same time, switching between them introduces costly bookkeeping and data duplication overheads [2].

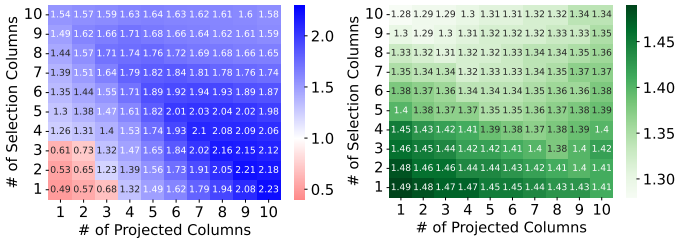
The same challenge also appears in real-time workloads such as image processing and neural-network-based applications, where accessing tensor data often results in complex strides that break locality. Moreover, the mismatch between the size of cache lines and data items (e.g., integers, double) results in unwanted data being transferred from main memory. As the size of the accessed data grows, moving data through the memory hierarchy becomes a fundamental bottleneck. The higher the pressure exerted on the bottleneck, the more unpredictability worsens.

While the main focus of our previous work is to optimize the average-case performance of relational databases, our intuition suggests that on-the-fly data reorganization can also bring significant benefits in terms of predictability for two main reasons: (1) reduced inter-process cache line eviction—thanks to cache footprint compression and (2) conversion of complex access patterns into sequential accesses—from the cache and prefetcher perspective. Overall, enforcing access patterns with high locality is increasingly more challenging in data-intensive applications in both real-time and relational systems. In all these applications, the processing is performed by streaming over a set of data items that are (1) orders of magnitude larger than the typical size of CPU caches; (2) often sparsely stored in memory; and (3) accessed with hard-to-predict, input-dependent patterns that are not optimized for the linear organization of data in DRAM. In addition, often, the computation performed on each data item is minimal. Thus, hiding the cost of data movement via deep pipelines and instruction-level reordering becomes ineffective.

In our demo, we will review the implications of on-the-fly data reorganization in CPU+FPGA systems. Next, we will provide a walk-through of our Relational Memory Engine (RME), capabilities, and deployment procedure on a real hardware platform. Finally, we will showcase the live acquisition of measurements that highlight the benefits of data reorganization from the standpoint of performance predictability.

II. Data Re-organization Engine

The RME is a hardware module located between the last-level cache (LLC) and memory. Cache refills on a (configurable set of) variables go through the FPGA where RME resides to capture CPU-memory accesses on-the-fly. Upon the first capture, it initiates a set of transfers from main memory to carve out only the desired bytes and into an internal buffer where data locality is maximized (Figure 1). Once ready, a cache line of packed useful data is available to the CPU as if it existed in the main memory.



(a) Speedup - RME vs Columnar (b) Speedup - RME vs Row

Fig. 2: Heat map of the RME speed-up against columnar 2a and row 2b store for varying projected and selected columns.

We implemented and deployed RME on commercially available Systems-on-Chips (SoCs) integrating an on-chip FPGA and a traditional multi-core processor (e.g., Intel HARPv2, Xilinx UltraScale+). By employing commercially available CPU+FPGA SoCs, we create an immediately-usable complete prototype capable of running realistic applications. Our design is based on the Programmable Logic In the Middle (PLIM) [3] approach and can be employed to achieve greater control over memory traffic by instantiating custom logic as an intermediary between processors and main memory.

III. Data-reshape for real-times systems

RME creates a re-organized alias of the target memory based on a software-provided configuration. RME achieves the timeliness requirements of real-time systems by accessing only the desired subset of data items in main memory on behalf of the processing units before sending fully compressed cache lines to the LLC. This mechanism effectively filters out all undesired elements that would otherwise pollute the cache, enabling high data locality in upstream caching layers.

Motivated by real-time applicability, first, we experimentally demonstrate that RME offers efficient native accesses to any matrix column or column group, outperforming direct row-wise and direct columnar accesses. To perform a fair comparison, we implement RME, the row-store (ROW), and the column-store (COL) approach in the same memory. The default size of each row is 64 bytes, and the column width is 4 bytes. Each experiment was repeated 30 times, and we reported averages and standard deviations. We run two sets of experiments for RME: hot (when the targeted data is ready in the internal) and cold (otherwise).

We design a synthetic benchmark (Listing 1) to test the behavior of our engine under representative memory access patterns. Consider the following operation: Given a matrix M, it reads over the columns subset based on a different selection predicate. Here, $COL_{p_1}, ..., COL_{p_i}$ are projection columns and $COL_{s_1}, ..., COL_{s_j}$ are selection columns.

Listing 1: Synthetic Matrix Operation

```
READ COL_{p_1}, ..., COL_{p_i} FROM M WHERE COL_{s_1}, ..., COL_{s_j} > k ;
```

A. Latency Showcase

Figures 2a and 2b show the speedup of RME compared to the in-memory row-store and column-store. In the x- and y-axis we vary the number of projection (i) and selection (j) columns. Figure 2a shows that when the number of involved columns is small ($\leq$ 4), column-store dominates over RME (colored red). However, as the number of columns increases due to the tuple materialization cost, the diminished prefetching columnar access performance falls behind. In fact, RME can be up to $2.23\times$ faster than columnar access (bottom rightmost cell). Figure 2b further highlights that RME **always** outperforms in-memory row access by being $1.3-1.5\times$ faster.

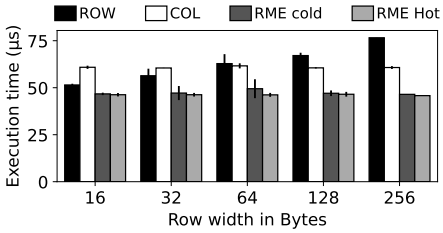


Fig. 3: RME enables deterministic accesses latency.

B. Predictability Showcase

We continue our experimentation with the benchmark above where $i = 1, j = 1, COL_i \neq COL_j$, focusing on the comparison between RME, direct row-wise (ROW), and direct columnar access (COL). We access 4 byte-wide columns while varying the row size. Figure 3 shows the absolute latency.

We note from this figure that even without having the projected column in the Reshape Buffer in FPGA (RME *cold*), *RME has faster execution* than both ROW and COL in all experiments. The reason is that (1) RME better exploits the internal memory bandwidth to fetch only the desired data items at bus-width granularity, and (2) the CPU caches are not polluted with unwanted fields.

RME's latency remains virtually the same as it accesses only the relevant data. However, answering the query via direct access of the row-oriented data leads to poor cache utilization as larger rows lead to higher cache pollution. Conversely, RME exhibits *stable and predictable performance regardless of the row size*. Thus, RME allows predicting and exploiting data reuse across processing phases.

C. Real-Time Evaluation

RME outperforms the row-store layout because, by definition, it accesses fewer data. On the other hand, queries that access fewer columns can be more efficiently evaluated from a columnar layout. However, when the number of projected columns is high enough (more than four in our setup), RME outperforms the columnar layout. Further, the RME implementation used in this setup runs at only 1/3 of the maximum FPGA frequency. Operating at a higher frequency may reduce memory access time and increase the benefits of RME.

IV. Conclusion

We **depart from the traditional view of memory as a flat array of bytes**. We reshape the data via near-memory computation before moving it to the CPU, resulting in improvement of both performance and determinism of memory accesses.

ACKNOWLEDGMENT

The material presented in this paper is based upon work supported by the National Science Foundation (NSF) under grant number CCF-2008799. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the NSF. Denis Hoornaert was supported by the Chair for Cyber-Physical Systems in Production Engineering at TUM and the Alexander von Humboldt Foundation.

REFERENCES

[1] S. Roozkhosh, D. Hoornaert, J. H. Mun, T. I. Papon, A. Sanaullah, U. Drepper, R. Mancuso, and M. Athanassoulis, "Relational memory: Native in-memory accesses on rows and columns," in *Proceedings 26th International Conference on Extending Database Technology, EDBT 2023, Ioannina, Greece, March 28-31, 2023*. [Online]. Available: https://doi.org/10.48786/edbt.2023.06

[2] R. Appuswamy, M. Karpathiotakis, D. Porobic, and A. Ailamaki, "The Case For Heterogeneous HTAP," in *Proceedings of the Biennial Conference on Innovative Data Systems Research (CIDR)*, 2017. [Online]. Available: http://cidrdb.org/cidr2017/papers/p21-appuswamy-cidr17.pdf

[3] S. Roozkhosh and R. Mancuso, "The potential of programmable logic in the middle: cache bleaching," in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2020.

Demonstrating R-TOD: Real-Time Object Detector with Minimized End-to-End Delay

Seungha Kim[1], Ho Kang[1], Sol Ahn[1], Kyungtae Kang[2], Nikil Dutt[3], and Jong-Chan Kim[1,4]
[1]*Graduate School of Automotive Engineering, Kookmin University, Korea*
[2]*Department of Computer Science and Engineering, Hanyang University, Korea*
[3]*Department of Computer Science, University of California, Irvine, USA*
[4]*Department of Automobile and IT Convergence, Kookmin University, Korea*
{*0206dbsh, rkdghrk12, nur636*}*@kookmin.ac.kr, ktkang@hanyang.ac.kr, dutt@uci.edu, jongchank@kookmin.ac.kr*

***Abstract*—The end-to-end delay of object detection systems should be thoroughly analyzed and minimized. In this regard, our previous work [1] presented a real-time object detector with minimized delays (a.k.a. R-TOD). This work, in turn, demonstrates R-TOD by comparing its delays with our baseline object detector (i.e., renowned Darknet YOLO). For that, we use two identical computing platforms with cameras and monitors so that the audience can see the different delays between the physical appearance of objects and their detection on monitors.**

I. Introduction to R-TOD

R-TOD is a real-time object detector presented in our previous work [1], which aims at minimizing the end-to-end delay from the physical appearance of an object to its detection. The work was motivated by observing significant time lags of many state-of-the-art object detectors, for example, the Darknet YOLO object detectors [2], [3], despite their decent frame rates.

To solve this peculiar timing issue, we thoroughly analyzed the internal architecture of object detection systems, finding three significant defects. Then an optimal architecture was proposed, which shows 76% average and 67% 99th percentile delay reductions for YOLOv3, without any loss in object detection accuracy. R-TOD is currently implemented in Nvidia integrated GPU (iGPU) platforms (e.g., Nvidia Jetson AGX Xavier and Nvidia Jetson AGX Orin).

A. End-to-End Delay Analysis

We analyzed the internal architecture of the Darknet YOLO object detector, where we found the following issues with significant impacts on end-to-end delays:

- **Queue delay.** There is a queue between the camera subsystem (i.e., producer) and the detector subsystem (i.e., consumer), causing a significant queue delay due to the faster consumer compared with the slower producer.
- **Imbalanced pipeline stages.** Three pipeline stages (i.e., fetch, inference, and display) comprising the detector subsystem have different lengths, causing significant idle times in the pipeline schedule.
- **CPU-GPU memory contention.** In iGPU systems, concurrently running GPU kernels and CPU threads incurs significant shared memory contention, causing significantly increased execution times.

B. End-to-End Delay Optimization

Based on the analysis, we developed three optimization techniques as follows:

- **On-demand capture.** By eliminating the queue between the camera and detector subsystems, we can remove the queue delay. Then images are captured on demand by a blocking call. Thus, a new blocking delay factor appears while waiting for the arrival of a captured image. However, it is far smaller than the queue delay.
- **Zero-slack pipeline.** In usual embedded systems, the inference stage has the dominant pipeline cycle length. To eliminate the idle time between a fetch stage and its inference stage in the next cycle caused by the imbalanced pipeline stages, we release the fetch stage such that its completion time becomes close to the completion of the inference stage in the same pipeline cycle.
- **Contention-free pipeline.** To alleviate the significant memory bandwidth contention in our iGPU-based target system, we modify the pipeline architecture such that the inference stage no longer concurrently executes with other CPU threads. As a result, this new architecture significantly reduces the delay at the cost of a slightly increased object detection cycle time.

II. Demonstration

For the demonstration, we will install two object detection systems with cameras, computing platforms, and monitors. The end-to-end delays can be tangibly recognized by seeing when the physical movement in the real world is detected by the object detector in the monitor. In such ways, the baseline object detector and our R-TOD will be compared in real time so the audience can actually see the difference.

Acknowledgment

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIT) (2022R1A2C1013197). The corresponding author is J.-C. Kim.

REFERENCES

[1] W. Jang, H. Jeong, K. Kang, N. Dutt, and J.-C. Kim, "R-tod: Real-time object detector with minimized end-to-end delay for autonomous driving," in *2020 IEEE Real-Time Systems Symposium (RTSS)*, Dec 2020, pp. 191–204.

[2] J. Redmon, "Darknet: Open source neural networks in C," http://pjreddie.com/darknet/, 2013–2016.

[3] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

Demo Paper: Real-Time Monitoring of Heart Rate Variability with PPG

Jingye Xu, Yuntong Zhang, Mimi Xie, Wei Wang and Dakai Zhu
The University of Texas at San Antonio

***Abstract*—Heart rate variability (HRV) is a critical vital sign that can predict a number of different diseases such as heart attack, arrhythmia, and stress. Traditionally, hospitals use electrocardiogram (ECG) devices to record the heart's bioelectrical signals which are converted to HRV values. Despite the high accuracy, this method is expensive and inconvenient. Recently, using photoplethysmography (PPG) sensors that collect reflective light signals has been adopted as a cost-effective alternative for measuring heart health. However, due to the sensitivity of PPG sensors, HRV estimation with PPG signals remains a challenging problem. To this end, this paper demonstrates a real-time monitoring system of HRV with PPG sensors. The real-time HRV monitoring system, which benefits from machine learning, is developed with a resource-limited ultra-low-power microcontroller unit (MCU). In addition, it empowers the system with adaptive reconfiguration capability at run-time to improve energy efficiency and adapt to different demands. Moreover, the demo shows the HR and HRV in a real-time manner with a display.**

I. Introduction

Heart rate variability (HRV) which measures the difference in time between successive heart beats is widely considered as one of the most important vital signs of body health [1]. HRV analysis has become an increasingly important diagnostic tool in cardiology as it shows relations to heart rate turbulence, maximal oxygen uptake, inflammatory response, and exercise capacity [2], [3]. As a result, HRV monitoring system is indispensable for people who need real-time monitoring of heart activities.

Traditionally, hospitals use electrocardiogram (ECG) devices consisting of electrodes mounted on the human body to record the heart's electrical signals [4]. ECG devices can provide accurate and real-time HRV monitoring, which is the best choice for patients who need intensive care in hospitals. However, those ECG devices are heavy and not portable as they require cable connections. Recently, ECG technology has been integrated into Apple Watch [5], [6] for everyday HRV monitoring. Despite the capability of starting HRV monitoring anytime and anywhere, it still has several limitations due to the special operation features of ECG. To start HRV monitoring with the ECG module, Apple Watch users are required to rest their arms on a table or on their lap and keep their fingers touching the Digital Crown (the button on Apple Watch), which is inconvenient and impossible for continuous long-time monitoring [7].

This project was funded (in-part) by The University of Texas at San Antonio, Office of the Vice President for Research, Economic Development & Knowledge Enterprise.

Alternatively, using photoplethysmography (PPG) sensors that collect reflective light signals appears to be a promising approach to measuring heart health as it is low-cost and more convenient than ECG devices [8]. Although PPG sensors can not provide the R-R interval values which are the essential information for calculating HRV, they can extract the "peak"-"peak" interval values that can be interpreted as the cardiac R-R interval [9]. The location of a peak represents the time instant at which a heartbeat occurs. HRV computation requires accurate identification of the location of peaks in the PPG signal. However, due to the sensitivity of PPG sensors, HRV computation with PPG signals remains a challenging problem.

To address the problem, we design a real-time monitoring system of HRV with PPG sensors. This paper demonstrates our prototype deployed on an ultra-low-power microcontroller (MCU) and how it works by adaptively switching in different modes for different demands.

II. Framework for efficient PPG-based HRV estimation

A. Real-time On-device HRV Monitoring

Since the PPG sensor needs to be integrated into a low-power wearable device with limited computing resources (low CPU frequency and small memory size), the implementation of real-time HRV monitoring needs to take the specifications of the ultra-low-power MCU into account. To fulfill the requirements of real-time, we propose the data pipeline as shown in Fig. 1.

Initially, the light signals of the PPG sensor will be stored in a buffer on MCU as the signals data array will be used to estimate peaks by a signal processing function. The size of the buffer is determined by the sampling rate times four seconds. For example, when the sampling rate of the sensor is set to 25Hz, the buffer size is 100 (25×4) with each value being a four-byte floating number. The reason we choose floating numbers to store the raw is that the raw data usually has a value up to 14,000 and the peak estimation algorithm requires floating number operations. Once we get the estimated peaks from signal processing, we get an approximately calculated HR. Then the calculated HR will be sent to the HR model as the input to predict HR. In our experiment, there is one HR being predicted and stored in the non-volatile memory every second. The predicted HR will be sent to a UART client to provide real-time monitoring. After every specific time interval (like the 30s, 60s, etc.), we will retrieve all historical HR to estimate an HRV during this period. The HR

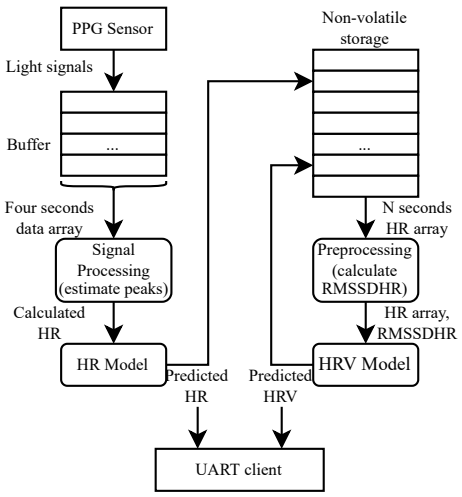


Fig. 1. Data pipeline of real-time on-device HRV monitoring system.

array will be sent to a preprocessing function to calculate one feature named RMSSDHR. The calculation for RMSSDHR is shown in Equation (1), where HR_i denotes the i^{th} HR, and N denotes the total number of HR in a given period (the period can be 30-300 seconds in our experiment). After that, we use the calculated RMSSDHR and the HR array as the input features to predict the HRV. The predicted HRV will be stored in the non-volatile memory and sent to the UART client for monitoring as well.

$$\text{RMSSDHR} = \sqrt{\frac{\sum_{i=1}^{N-1}(60,000/HR_{i+1} - 60,000/HR_i)^2}{N-1}} \tag{1}$$

The data pipeline proposed above fully considers the memory and computing limitations of an ultra-low-power MCU. Non-volatile storage is used to avoid data loss when power is off. We only store light signals in the RAM to ensure enough memory space for HR signal processing, model inference, and HRV preprocessing.

B. Adaptive Run-time Reconfiguration

To implement run-time reconfiguration, we deploy three system configuration modes, which are designed for various demands, on the board as shown in Table I. In these three modes, the framework will work under different deep learning models, sampling rate (SR), and MCU Digitally Controlled Oscillator (DCO) frequency. Thus as a result, with the pre-configurations, we can easily switch between different modes to meet different needs.

TABLE I
VARIOUS SYSTEM MODES.

Mode	Model	DCO/MHz	SR/Hz
Fast mode	121-10-10-1	8	25
Energy saving mode	121-10-10-1	1	12.5
High accurate mode	301-12-12-1	8	100

III. Demonstration Setup

A. Hardware Setup

Fig. 2 depicts the setup of the demonstration. The PPG sensor is MAXREFDES117 that provides reliable light signals and has noise filter units. The MSP430FR5994 development kit is one of the MSP430 families produced by Texas Instruments. The development kit contains everything needed to start developing on the ultra-low-power microcontroller platform, including on-device debug probe for programming, debugging, and energy measurements. The PPG sensor communicates with the MSP430FR5994 through the I2C bus. The MSP430FR5994 board connects to a Raspberry Pi through a micro-USB to get power and to a convenient UART communication. The Pi will decode the UART messages and display the monitoring results on a TFT LCD display with a resolution of 320×240.

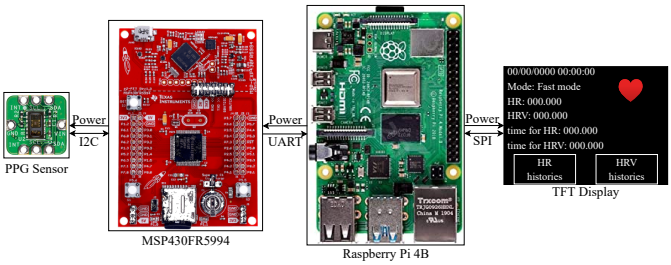


Fig. 2. Setup.

References

[1] J Camm. Task force of the european society of cardiology and the north american society of pacing and electrophysiology. heart rate variability: Standarts of measurement, physiological interpretation and clinical use. *Circulation*, 93:1043–1065, 1996.

[2] George E Billman, Heikki V Huikuri, Jerzy Sacha, and Karin Trimmel. An introduction to heart rate variability: methodological considerations and clinical applications. *Frontiers in physiology*, 6:55, 2015.

[3] Juul Achten and Asker E Jeukendrup. Heart rate monitoring. *Sports medicine*, 33(7):517–538, 2003.

[4] David B Geselowitz. On the theory of the electrocardiogram. *Proceedings of the IEEE*, 77(6):857–876, 1989.

[5] Anand N Ganesan, Derek P Chew, Trent Hartshorne, Joseph B Selvanayagam, Philip E Aylward, Prashanthan Sanders, and Andrew D McGavigan. The impact of atrial fibrillation type on the risk of thromboembolism, mortality, and bleeding: a systematic review and meta-analysis. *European heart journal*, 37(20):1591–1602, 2016.

[6] Nino Isakadze and Seth S Martin. How useful is the smartwatch ecg? *Trends in cardiovascular medicine*, 30(7):442–448, 2020.

[7] Apple Inc. Take an ecg with the ecg app on apple watch, May 2022.

[8] John Allen. Photoplethysmography and its application in clinical physiological measurement. *Physiological measurement*, 28(3):R1, 2007.

[9] Christopher G Scully, Jinseok Lee, Joseph Meyer, Alexander M Gorbach, Domhnull Granquist-Fraser, Yitzhak Mendelson, and Ki H Chon. Physiological parameter monitoring from optical recordings with a mobile phone. *IEEE Transactions on Biomedical Engineering*, 59(2):303–306, 2011.

Demo Abstract: Real-Time Out-of-Distribution Detection on a Mobile Robot

Michael Yuhas[1,2], Arvind Easwaran[1]
[1]*School of Computer Science and Engineering*
[2]*Energy Research Institute @ NTU, Interdisciplinary Graduate Program*
Nanyang Technological University, Singapore
michaelj004@e.ntu.edu.sg, arvinde@ntu.edu.sg

***Abstract*—In a cyber-physical system such as an autonomous vehicle (AV), machine learning (ML) models can be used to navigate and identify objects that may interfere with the vehicle's operation. However, ML models are unlikely to make accurate decisions when presented with data outside their training distribution. Out-of-distribution (OOD) detection can act as a safety monitor for ML models by identifying such samples at run time. However, in safety critical systems like AVs, OOD detection needs to satisfy real-time constraints in addition to functional requirements. In this demonstration, we use a mobile robot as a surrogate for an AV and use an OOD detector to identify potentially hazardous samples. The robot navigates a miniature town using image data and a YOLO object detection network. We show that our OOD detector is capable of identifying OOD images in real-time on an embedded platform concurrently performing object detection and lane following. We also show that it can be used to successfully stop the vehicle in the presence of unknown, novel samples.**

I. Introduction

Machine learning (ML) models are not likely to perform well when they receive samples outside of their training data distributions. This poses a major risk to safety critical cyber-physical systems such as autonomous vehicles (AVs). Consider an ML-based object detector deployed to an AV: during training, the object detector may be exposed to images with little or no snow, and during operation, a heavy snowstorm could lead to poor results. Fig. 1-a shows an in-distribution (ID) image for such a system: the YOLOv7 model [1] successfully detects the ducks closest to the vehicle when no snow is present. However, when excessive snowfall occurs (simulated by falling confetti) the scene becomes out-of-distribution (OOD) (Fig. 1-b), and the model can no longer detect the ducks. It is imperative that such OOD samples are identified to prevent the system from taking dangerous control actions. An OOD detector can be used as a run-time safety monitor to achieve this goal, however, the detector must also meet hard real-time deadlines to ensure that detection occurs with sufficient time to avoid a collision [2]. Furthermore, OOD detection on image data relies on deep neural networks, which must share computational resources with other safety critical tasks.

This research was funded in part by MoE, Singapore, Tier-2 grant number MOE2019-T2-2-040. This research is part of the programme DesCartes and is supported by the National Research Foundation, Prime Minister's Office, Singapore under its Campus for Research Excellence and Technological Enterprise (CREATE) programme.

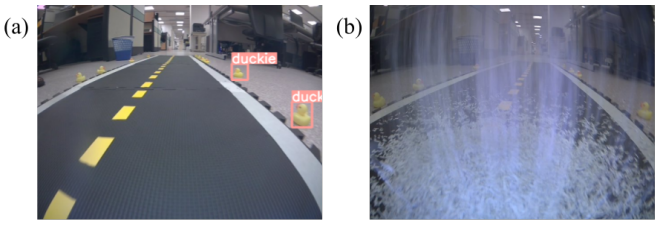


Fig. 1. (a) The YOLO (you only look once) model detects the two nearest ducks when an image is ID; (b) the YOLO model is unable to identify any objects in the OOD image with simulated snowfall.

Prior works have deployed OOD detectors to mobile robots and demonstrated their ability to meet deadlines and detect harmful samples at run time. In [3], the reconstruction loss of a variational autoencoder (VAE) was used to perform OOD detection on image data in a simulated environment. Both detection accuracy and execution time were considered, however, the experiments were not performed on a physical system. In [2], the latent space of a VAE was used for OOD detection on a mobile robot to trigger an emergency stop before a collision occurred. While this work was demonstrated on a physical system, the OOD detector failed to meet real-time constraints and was not always able to stop the robot in time. In [4], a deep radial basis function (RBF) network was trained to steer a mobile robot around a racetrack and simultaneously perform OOD detection. This integrated OOD detection and steering control network requires regression tasks to be reformulated as classification tasks, which is not always desirable. So far, no work has deployed a deep ML model and an independent OOD detector to a CPS and observed the effect on the response times of both tasks.

We will perform a live demonstration of the feasibility of an OOD detector as a real-time safety monitor for a mobile robot performing lane navigation and object detection tasks.

1) Our OOD detector meets real-time deadlines *and* does not interfere with the lane follower's operation.
2) Our OOD detector successfully triggers emergency braking in OOD conditions without excessive false-positives.
3) Our test bed allows us to easily create unique test scenarios; while we consider the case of snowfall as OOD, the audience can attempt to foil the OOD detector with other conditions.

II. System Design

We use the Duckietown platform [5] to simulate an AV. Duckietown consists of small, holonomic robots (Duckiebots) that navigate a miniature world (Duckietown) with lane markings that mimic actual roads. The inhabitant of Duckietown (rubber ducks) serve as obstacles that the Duckiebot must not hit. This platform allows us to deploy multiple ROS (Robot Operating System) packages to a Duckiebot, each running within its own Docker container. Fig. 2 shows the high-level software block diagram of our system. A lane follower based on traditional computer vision (CV) techniques identifies road markings, calculates the desired pose of the robot, and sends this information to a kinematics node that calculates the wheel rotations required to achieve that pose. In parallel, an object detection node identifies potential obstacles and sends bounding box coordinates to a graphical display for visualization. An OOD detector acts as a safety monitor for the entire system and sends an emergency stop message if an incoming image is OOD. The system has one sensor (camera node) and two actuators (wheels driver node). The source code for our software implementation is available on GitHub. [1]

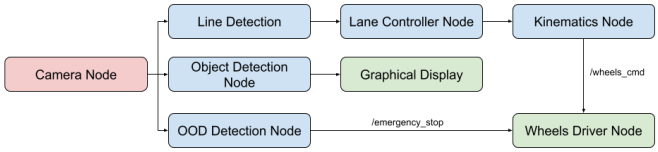


Fig. 2. Software system block diagram of our Duckiebot. Red indicates a sensing node, blue corresponds to a signal processing node, and green represents an output node.

A. Lane Following

Duckietown's native lane following package is used to steer the robot [5]. It uses traditional CV algorithms and consists of the following steps: image equalization, road marking detection, projection from image space to the world frame, a Bayesian filter for lane localization, and finally the lane controller which generates the wheel commands. In isolation we measured an average-case execution time (ACET) of 40.1 ms and a worst-case execution time of 134.3 ms. By using traditional CV line detection, we are able to show that the OOD detector is also useful at protecting non-ML components from anomalous images. For example, under conditions such as heavy snow, lane markings may not be visible and the lane follower will mistake spurious lines in the image as road markings leading to inappropriate control actions.

B. Object Detection

To accomplish object detection, we trained a YOLOv7 tiny object detection network [1] on the Duckietown object detection dataset that contains three classes: rubber ducks, traffic cones, and Duckiebots. YOLOv7 tiny is a lightweight variant of the YOLO family of object detection networks that identifies bounding box coordinates and object class information simultaneously [6]. Although not as accurate as two-stage methods like Fast R-CNN, YOLO networks are capable of faster inference times, which is desirable in our application. YOLOv7 tiny can be trained for different input image sizes and Fig. 3 shows the resulting trade-off between ACET and the model's ability to reliably identify objects. All ACETs are measured with the model quantized to 8-bits via dynamic quantization [7] and the QNNPACK backend for inference on the AArch64 target platform [8]. The YOLOv7 tiny model trained on 64x64 images (Fig. 3-a) is unable to make any correct detections, but has the fastest ACET. We select the model trained on 160x160 images (Fig. 3-d) for use in our demo: it is better at identifying the rubber ducks, but has an ACET of 163.4 ms.

Fig. 3. Confusion matrices and corresponding ACETs for a YOLOv7 tiny network trained on the Duckietown dataset with input sizes (a) 64x64, (b) 96x96, (c) 128x128, and (d) 160x160.

C. Out-of-Distribution Detection

In order to construct an accurate, yet fast OOD detector, we used the OOD design methodology proposed in [9]. This methodology takes an existing OOD detector and searches preprocessing parameter combinations and quantization schemes to find several candidate solutions. We plan to focus on OOD caused by falling snow (Fig. 1-b), so we started with the optical flow OOD detector proposed in [10] that uses a variational autoencoder to identify environmental motion not present in the training set. Our ID dataset consisted of 2048 images gathered by the Duckiebot navigating an empty track autonomously. We converted sequential images into optical flow matrices using the Farnebäck optical flow algorithm and ran a genetic algorithm (GA) to select the input image size, input depth (sequential flows), and interpolation method that

[1] https://github.com/CPS-research-group/CPS-NTU-Public

maximize the F1 score of the OOD detector. To calculate F1 score, we used a test set containing eight videos with frames labeled ID or OOD based on the presence of simulated snowfall. Table I shows the candidate solutions identified by the GA and their respective functional and non-functional performance. All the execution times (ETs) are calculated with 8-bit dynamic quantization [7] and tested with the QNNPACK backend [8] on the target platform. We selected the best OOD detector identified by the GA (60x80 input size, 10 flows, bilinear interpolation) for our demo.

TABLE I
CANDIDATE SOLUTIONS FOR THE OOD DETECTOR WITH THEIR FUNCTIONAL PERFORMANCE (F1 SCORE) AND ET MEAN AND VARIANCE.

Size	Flows	Interp.	F1 Score	ET (mean)	ET (var.)
30x40	4	Bilinear	0.43	44.1 ms	1098 ms^2
60x80	5	Bilinear	0.97	53.3 ms	1485 ms^2
90x120	2	Bilinear	0.54	58.1 ms	900 ms^2
120x160	12	Bilinear	0.90	70.0 ms	1096 ms^2

III. DEMONSTRATION

All experiments were performed on a DB21M Duckiebot equipped with a Jetson Nano 2GB running L4T 32.1 with the PREEMPT_RT kernel patch installed. Fig. 4 shows the main components of our demonstration. A Duckiebot drives forward along a road while the object detection network identifies duckies in the environment. To simulate snowfall, confetti is dumped in front of the robot. During the demonstration:

1) The OOD detector successfully triggers an emergency stop when snowfall is present in the environment.
2) In the absence of OOD samples, the vehicle can reach the end of the track without a false positive detection.
3) The OOD detector maintains a response time of less than 800 ms and the lane follower still meets its deadlines.

Fig. 5 shows the response times for the three tasks (OOD detection, object detection, and lane following) when run together on the Duckiebot. We observe that the OOD detector always has a response less than 800 ms and is able to stop the vehicle before a collision or leaving the road. Furthermore, the OOD detector does not interfere with the lane follower's response time and the Duckiebot is still able to navigate. However, the response time of the object detector increases drastically in comparison to its ACET when it shares the CPU with other tasks. We believe this is due to the system utilization approaching 100 percent and that when all three tasks are run simultaneously, all 2GB RAM have been utilized and memory intensive tasks (like YOLO inference) must continually swap pages in and out of memory as they are scheduled.

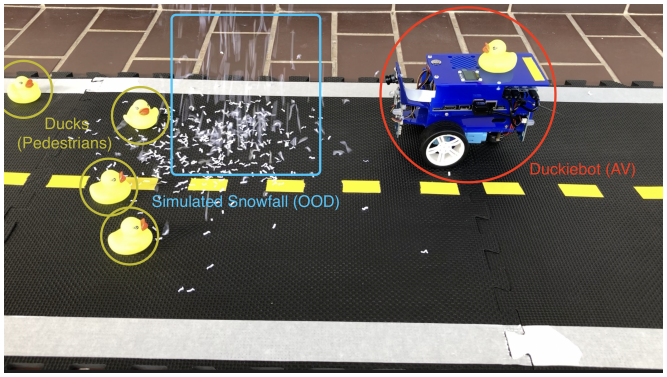


Fig. 4. The main components of the experimental setup: the Duckiebot (red) is a surrogate for an AV, the ducks (yellow) simulate pedestrians, and the confetti (blue) simulates snowfall.

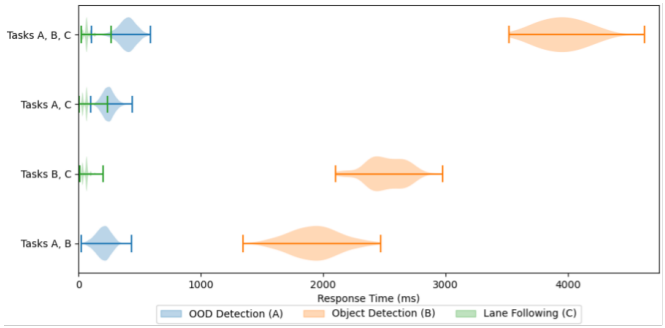


Fig. 5. Response time distributions when the OOD detection task (A), object detection task (B), and lane following task (C) run simultaneously on the Duckiebot. The response times of all three tasks increase in comparison to their ETs, but object detection suffers the most degradation.

REFERENCES

[1] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," Jul. 2022, doi: 10.48550/ARXIV.2207.02696.

[2] M. Yuhas, Y. Feng, D. J. X. Ng, Z. Rahiminasab, and A. Easwaran, "Embedded Out-of-Distribution Detection on an Autonomous Robot Platform," in *Proceedings of the Workshop on Design Automation for CPS and IoT*, May 2021, pp. 13–18, doi: 10.1145/3445034.3460509.

[3] F. Cai and X. Koutsoukos, "Real-time Out-of-distribution Detection in Learning-Enabled Cyber-Physical Systems," in *2020 ACM/IEEE 11th International Conference on Cyber-Physical Systems (ICCPS)*, Apr. 2020, pp. 174–183, doi: 10.1109/ICCPS48487.2020.00024

[4] M. Burruss, S. Ramakrishna and A. Dubey, "Deep-RBF Networks for Anomaly Detection in Automotive Cyber-Physical Systems," in *2021 IEEE International Conference on Smart Computing (SMARTCOMP)*, Aug. 2021, pp. 55–60, doi: 10.1109/SMARTCOMP52413.2021.00028.

[5] L. Paull *et al.*, "Duckietown: An open, inexpensive and flexible platform for autonomy education and research," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 1497–1504, doi: 10.1109/ICRA.2017.7989179.

[6] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016, pp. 779–788, doi: 10.1109/CVPR.2016.91.

[7] R. Krishnamoorthi, J. Reed, M. Ni, C. Gottbrath, and S. Weidman. "Introduction to Quantization on PyTorch." PyTorch.org. https://pytorch.org/blog/introduction-to-quantization-on-pytorch/#dynamic-quantization (accessed Oct. 15, 2022).

[8] M. Dukhan, Y. Wu, H. Lu, and B. Maher. *QNNPACK*. (2019). Accessed: Oct. 15, 2022 [Online]. Available: https://github.com/pytorch/QNNPACK/README.md.

[9] M. Yuhas, D. J. X. Ng, and A. Easwaran, "Design Methodology for Deep Out-of-Distribution Detectors in Real-Time Cyber-Physical Systems," in *2022 IEEE 28th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, Aug. 2022, pp. 180–185, doi: 10.1109/RTCSA55878.2022.00025.

[10] Y. Feng, D. J. X. Ng, and A. Easwaran, "Improving Variational Autoencoder based Out-of-Distribution Detection for Embedded Real-time Applications," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 20, no. 5s, pp. 1–26, Oct. 2021, doi: 10.1145/3477026.