

Interdependent multi-version scheduling in heterogeneous energy-aware embedded systems

Julius Roeder, Benjamin Rouxel, Sebastian Altmeyer, Clemens Grelck

Angaben zur Veröffentlichung / Publication details:

Roeder, Julius, Benjamin Rouxel, Sebastian Altmeyer, and Clemens Grelck. 2019. "Interdependent multi-version scheduling in heterogeneous energy-aware embedded systems." In Proceedings of the 13th Junior Researcher Workshop on Real-Time Computing, JRWRTC 2019, Toulouse, France, November 6-8, 2019, edited by Benjamin Rouxel and Antonio Paolillo, 45-48. Toulouse: IRIT - Institut de Recherche en Informatique de Toulouse. https://www.irit.fr/rtns2019/wp-content/uploads/2019/11/RTNS_Junior_Workshop.pdf.

Nutzungsbedingungen / Terms of use:

licgercopyright

Dieses Dokument wird unter folgenden Bedingungen zur Verfügung gestellt: / This document is made available under the following conditions:

Deutsches Urheberrecht

Weitere Informationen finden Sie unter: / For more information see:

<https://www.uni-augsburg.de/de/organisation/bibliothek/publizieren-zitieren-archivieren/publizieren>



Interdependent Multi-version Scheduling in Heterogeneous Energy-aware Embedded Systems

Julius Roeder
University of Amsterdam
Amsterdam, Netherlands
Email: j.roeder@uva.nl

Benjamin Rouxel
University of Amsterdam
Amsterdam, Netherlands
Email: benjamin.rouxel@uva.nl

Sebastian Altmeyer
University of Augsburg
Augsburg, Germany
Email: sebastian.altmeyer@
informatik.uni-augsburg.de

Clemens Grelck
University of Amsterdam
Amsterdam, Netherlands
Email: c.grelck@uva.nl

Abstract—High-performance heterogeneous multi-core embedded systems are increasingly popular in various fields. Embedded systems engineers need to reason about more than just functional correctness of applications; they also need to reason about energy, time and security (ETS). In this paper, we sketch our coordination language and scheduling approach to enable ETS-aware applications. We present an Integer Linear Programming (ILP) based scheduler on a real life drone application, that minimizes energy consumption, guarantees timing and offers security.

Index Terms—energy-aware scheduling, heterogeneous multi-core, real-time scheduling, coordination.

I. INTRODUCTION

Increasing demand for processor performance, low power consumption and reducing heat dissipation, has lead to a surge in demand for high-performance multi-core embedded systems [1]–[4]. Powerful embedded systems, such as the Odroid-XU4 [5] and Nvidia Jetson [6], are multi-/many-core heterogeneous systems; however, with great performance comes great energy consumption (at least relative to traditional embedded systems). Furthermore, not only performance and energy consumption are important, but also security as decisions made by autonomous systems can save or cost lives. Thus, an engineer needs to take all three, time, energy and security (ETS), into account.

To tackle these challenges and enable end users to reason about ETS-characteristics of applications, we borrow from and combine two so far disjoint established research fields, namely *coordination* and *real-time scheduling* [7], they have so far addressed our targeted problem from very different angles and with very different motivations and intentions.

An application organized according to the coordination paradigm consists of a collection of interacting, independent, identifiable black-box components, i.e. the coordination language is used to describe the task graph of the given application. Components, also known as actors or tasks¹, i.e. represent application features, sequential building blocks of application, implemented in a general purpose programming language. Each component has defined functional properties, communication and interaction with other parts of the same application. Hence, coordination describes the flow of data through the different parts of an application [8]. Coordination is a well established computing paradigm with a plethora of languages, abstractions and approaches; for a survey see [9].

¹Components and tasks will be used hereafter interchangeably.

An application described in a coordination language can then be mapped and scheduled using techniques from the real-time scheduling domain. The literature on real-time scheduling algorithms for multi-core architectures is vast, with many properties (e.g. type of scheduling algorithm, task model) and are classified in three main categories: partitioned, global and hybrid [1]. Similarly to [10], in this paper we propose an Integer-Linear-Programming (ILP) based approach to produce schedules. According to the taxonomy proposed by Davis and Burns [1] our approach can be classified as static, partitioned, time-triggered and non-preemptive.

Different approaches to minimizing energy consumption of an application on heterogeneous multi-core hard-real time systems are surveyed in [11]. Our ILP based scheduler addresses energy and security as equally important as time. The ILP scheduler can be used to generate schedules for multi-version concurrent applications, executing on heterogeneous platforms (e.g. Odroid-XU4).

In Section II, we briefly describe the architecture model, task model and coordination approach. In Section II-B, we illustrate the coordination approach with a use-case from the area of unmanned aerial vehicles. In Section III, we describe our current ILP formulation and our approach towards deriving static schedules. In Section III-B, we show the viability of our ILP scheduler on the use-case, while targeting the Odroid-XU4.

II. ARCHITECTURE MODEL, TASK MODEL AND COORDINATION APPROACH

We consider a heterogeneous multi-core architecture (Odroid-XU4 [5]) based on the ARM big-LITTLE architecture [12]. This architecture consists of a heterogeneous CPU, with four energy efficient cores (i.e. LITTLE cores) and four high performance, deep pipeline cores (i.e. big cores). Additionally, the architecture contains a Mali GPU [5]. In such a heterogeneous platform, multi-version components can be scheduled on different computational units, e.g. a component implemented with OpenCL [13] can be scheduled on the GPU or the CPU.

In this work, we consider Directed Acyclic task Graphs (DAG). In a graph $G = (V, E)$, the set of nodes/vertices V represents the components and the set of edges E represents data dependencies between components. An edge between two components is present when they depend on each other, i.e. the source component needs to be completed before the sink

can be executed. The task graph is expressed with the help of a coordination language.

A. Coordination

A coordination language is independent from the actual code, but it guides the scheduler on how this code should be executed. An example is the coordination language S-Net [8]. However, like other coordination approaches S-Net merely addresses functional aspects of coordination programming and does not include non-functional requirements, i.e. time and security. Our coordination language [14], implemented as a Domain Specific Language (DSL), allows to incorporate the ETS-characteristics of the application.

Given the focus of the safety-critical embedded systems domain, we exclusively work with the system-level programming language *C*. Hence, a component is technically a callable *C* function with certain restrictions on its functional behaviour, together with a set of non-functional properties, i.e. timing, energy and security. Therefore, each component has specific ETS-characteristics that are crucial to determine the best mapping and schedule. Additionally, each component can have multiple implementations with equivalent functional requirements, but different ETS-characteristics. Thus, a component can have the previously mentioned multiple versions.

Our DSL coordination language allows users to specify component-specific and system-wide ETS-constraints, expressed as: 1) deadlines on the response-time of a component, 2) energy budgets of the entire system and 3) minimum security levels of each component.

B. Coordination illustrated by drone use case

Figure 1 shows the graph representation of an application that will be executed on a drone. Due to space limitation, we will not present this use-case with our coordination language. The application is currently under development at the University of Southern Denmark (SDU) and Sky-Watch, an industrial partner [15]. The application records and analyses a video on the fly and is executed on the aforementioned Odroid-XU4.

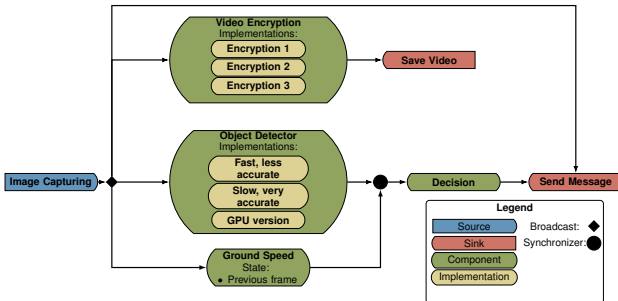


Fig. 1. Drone application coordination model

The application is organized as seen in Figure 1:

- **Image Capturing:** A frame is streamed to the computer.
- **Broadcast:** A frame is broadcasted to three components.
- **Video Encryption:** The scheduler selects the encryption level based on the ETS-characteristics of each encryption level.

- **Video Encryption & Save Video:** The frame is encrypted and saved to disk.
- **Object Detection:** The frame is analysed by the Darknet Neural Network, Tiny Darknet Neural Network [16] or OpenCV [17], depending on the ETS-characteristics of each.
- **Ground Speed:** Computes the ground speed of the drone.
- **Synchronizer:** The results of the previous components (Object Detection & Group Speed) are synchronized and sent to the final Decision component.
- **Decision & Send Message:** A decision is made and sent to the ground station.

The two neural networks (Darknet and Tiny Darknet [16]) differ in their complexity. The Tiny Darknet neural network is much smaller and therefore can run inference approximately 70 times faster on the Odroid-XU4, at the price of reduced accuracy. Thus, scheduling one or the other version of the *Object Detection* component can have a large impact on the run time.

We aim at building a complete toolchain and workflow to compile a coordinated application to a final executable as presented by Figure 2.

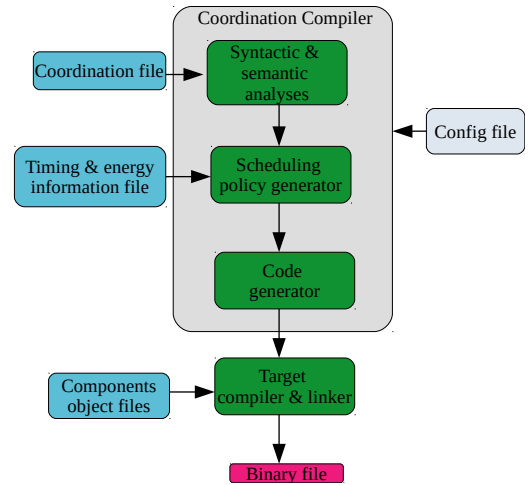


Fig. 2. Coordination workflow

A major part of our approach is to improve scheduling policies, represented by the *Scheduling policy generator* in Figure 2. Hence, in the following section we introduce our first version of a multi-version ILP based scheduler. A more thorough description of the rest of the toolchain and coordination language can be found in [14].

III. SCHEDULING

Once the different components are identified and defined via the coordination technique, we need to schedule the different components, i.e. assign components to different cores or co-processors (spatial mapping) and in a given order (temporal mapping). On top of different versions of the same components, the ETS-characteristics also differ depending on the exact execution unit (big core vs. LITTLE core vs. GPU) of a heterogeneous system. This increases the state space of the scheduling as we need to schedule the different components

TABLE I
ILP VARIABLES AND CONSTANTS

Function	predecessors(p)	retrieves all predecessors of p , where $p \in \tau$
Sets	B	Set of cores
	τ	Set of all components
	v_p	All versions of component p , where $p \in \tau$
	O	Set of sinks
Constants	D^C	Deadline of component set
	D^S	Minimum security level
	$C_{p,m}$	Run time of component p on core m , where $p \in \tau, m \in B$
	$E_{p,m}$	Energy consumption of component p on core m , where $p \in \tau, m \in B$
	$S_{p,m}$	Security level of component p on core m , where $p \in \tau, m \in B$
	$F_{p,i}$	Run time of version i of component p , where $i \in v_p, p \in \tau$
	$G_{p,i}$	Energy consumption of version i of component p , where $i \in v_p, p \in \tau$
	$H_{p,i}$	Security of version i of component p , where $i \in v_p, p \in \tau$
	M	Sum of run time of all components on all cores
	Integer Variables	D^E
ρ_p		Start time of component p , where $p \in \tau$
Binary Variables	$w_{p,m}$	component p mapped to core m , where $p \in \tau, m \in B$
	$a_{p,i}$	version i of components p is selected, where $i \in v_p, p \in \tau$
	$same_{p,q}$	components p and q are scheduled on the same core, where $p, q \in \tau$
	$order_{p,q}$	Same core variable order p to q , where $p, q \in \tau$

depending on the overall application ETS-constraints and the ETS-characteristics of each implementation of each component.

As a starting point for the space-time scheduling we used Integer Linear Programming (ILP) to generate static schedules for the aforementioned drone use-case. ILP refers to a class of constrained optimization problems and is used to tackle problems in scheduling. All variables in an ILP problem are integers, constrained by linear inequalities. The objective function is a linear function of the variables, that needs to be either minimized or maximized.

A. ILP Formulation

The set of integer variables needed for the scheduler can be found in Table I.

Objective function The goal is to minimize the energy consumption D^E over all components of the application eq. (1):

$$\text{minimize } D^E = \sum_{p \in \tau} \sum_{m \in B} (E_{p,m} \times w_{p,m}) \quad (1)$$

where, $E_{p,m}$ is the energy consumption of component p on core m and $w_{p,m}$ is a binary variable indicating if component p is executed on core m .

Time and Security constraints Besides energy consumption, we also need to constrain the time and security aspects of the application. Thus, eq. (2) guarantees that the sinks of the application are finalized before the deadline D^C . The **finalization time** of every sink $o \in O$ is the sum of the start time ρ_o and the worst-case execution time $C_{o,m}$ of sink o executed on core m . Equation (3) ensures that every executed

component p has a security level $S_{p,m}$ equal or above the minimum security level D^S .

$$\sum_{m \in B} (\rho_o + C_{o,m} \times w_{o,m}) \leq D^C, \forall o \in O \quad (2)$$

$$\sum_{m \in B} S_{p,m} \times w_{p,m} \geq D^S, \forall p \in \tau \quad (3)$$

Mapping one component to one core Equation (4) ensures that component p is mapped on one and only one processor.

$$\sum_{m \in B} w_{p,m} = 1, \forall p \in \tau \quad (4)$$

Single version constraint Equation (5) enforces that exactly one and only one version of a component is selected, represented by $a_{p,i} = 1$.

$$\sum_{i \in v_p} a_{p,i} = 1, \forall p \in \tau \quad (5)$$

Determining ETS-characteristics Equations (6) to (8) impose that the energy, time and security of a component ($E_{p,m}, C_{p,m}, S_{p,m}$) are equal to the respective ETS-characteristics ($G_{p,i}, F_{p,i}, H_{p,i}$) of the version that is selected when $a_{p,i} = 1$.

$$E_{p,m} = \sum_{i \in v_p} (a_{p,i} \times G_{p,i}), \forall p \in \tau, \forall m \in B \quad (6)$$

$$C_{p,m} = \sum_{i \in v_p} (a_{p,i} \times F_{p,i}), \forall p \in \tau, \forall m \in B \quad (7)$$

$$S_{p,m} = \sum_{i \in v_p} (a_{p,i} \times H_{p,i}), \forall p \in \tau, \forall m \in B \quad (8)$$

Prevent overlapping on same core To prevent the overlapping of components on the same core we introduce three additional constraints. Equation (9) calculates if two components are assigned to the same core ($same_{p,q}$). Instead of a *logical or*, a summation over B is sufficient, due to the uniqueness of $w_{p,m}$. The *logical and* (\wedge) can be linearized, see [18] for details. Equation (10) determines the order of tasks p, q , which can be p then q ($order_{p,q}$) or q then p ($order_{q,p}$). Equation (11) prevents two tasks running on the same core to execute at the same time and enforces the correct task order. The start time of p (ρ_p) has to be larger than the end time of q ($\rho_q + C_{q,m}$).

$$same_{p,q} = \sum_{m \in B} w_{p,m} \wedge w_{q,m}, \forall (p, q) \in (\tau \times \tau), p \neq q \quad (9)$$

$$same_{p,q} = order_{p,q} + order_{q,p}, \forall (p, q) \in (\tau \times \tau), p \neq q \quad (10)$$

$$\rho_p + (1 - order_{q,p}) \times M \geq \rho_q + (C_{q,m} \times w_{q,m}), \quad (11)$$

The constraint eq. (11) must only be activated when two tasks are mapped on the same core ($order_{q,p} = 1$). Hence, a nullification with a big-M notation is applied [19]. M , defined in eq. (12), is the sum of the maximum run time of all components p . Thus, due to the large value of M the left hand side of eq. (11) is always larger than the right hand side, if $order_{q,p} = 0$.

$$M = \sum_{p \in \tau} \max(C_{p,m}) \quad (12)$$

Data dependencies in task graph In order to comply with the data dependencies in the task graph we introduce the

following constraint eq. (13). It ensures, that if one component p depends on the data of another component q , the start time of p (ρ_p) is larger than the end time of q ($\rho_q + C_{q,m}$).

$$\rho_p \geq \rho_q + \sum_{m \in B} (C_{q,m} \times w_{q,m}), \forall p \in \tau, \forall q \in \text{predecessors}(p) \quad (13)$$

B. Example Use Case

We demonstrate the generation of a static schedule of the drone use case using the ILP formulation presented in Section III-A. Not all component implementations of the algorithm are fully functional at the current stage of the project. We rely on values measured at development time, to demonstrate the scheduling and coordination techniques. Once full implementations and corresponding ETS properties will be known, we will replace these hypothetical values with actual ones. The described ETS-characteristics of a component differ per computational unit, thus the heterogeneity of the Odroid-XU4 can be taken into account.

Depending on parameters (i.e. energy budget, deadline, minimum security level), the ILP identifies different schedules. One schedule resulting from the ILP formulation is shown in Figure 3. Components are executed in the right order, according to dependencies, on different computational units without overlap.

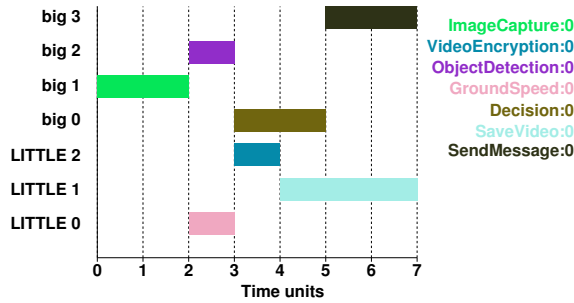


Fig. 3. Possible schedule for the drone use case showing which version of a component is scheduled and mapped on which computational unit.

As opposed to Figure 3, we increased the general security level to generate the second schedule presented in Figure 4. A higher security level lead the scheduler to pick different component versions, e.g. video encryption, and increased the energy consumption by 2 units. Once again the schedule respects component dependencies, while picking different versions and computational units due to requirement changes.

IV. CONCLUSION

In this paper, we briefly introduced the coordination paradigm to describe an application. We also presented an ILP formulation that allows us to generate different static schedules depending on ETS-characteristics and constraints. We further demonstrate its application on a realistic use case.

Furthermore, to enable the use of our coordination and scheduling layer for larger problems we will explore scheduling heuristics. Additionally, when all component implementations will be functional we will test the impact of our scheduling policy on real hardware and measure the impact on run time and energy consumption.

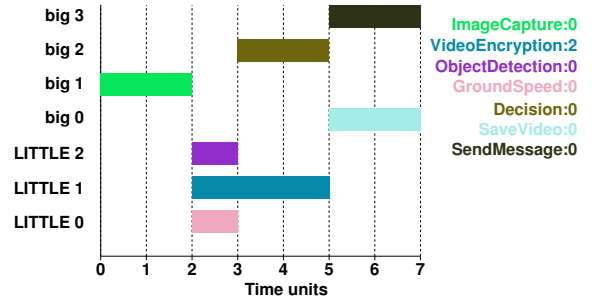


Fig. 4. Alternative schedule for the drone use case with higher security requirements.

ACKNOWLEDGMENT

The project has received funding from the European Unions Horizon2020 research and innovation programme under grant agreement No 779882.

REFERENCES

- [1] R. Davis and A. Burns, "A survey of hard real-time scheduling algorithms for multiprocessor systems," in *ACM Computing Surveys*, 2011.
- [2] M. Becker, D. Dasari, B. Nolic, B. Akesson, V. Nélias, and T. Nolte, "Contention-free execution of automotive applications on a clustered many-core platform," in *Real-Time Systems (ECRTS), 2016 28th Euromicro Conference on*, pp. 14–24, IEEE, 2016.
- [3] H. Rihani, M. Moy, C. Maiza, R. I. Davis, and S. Altmeyer, "Response time analysis of synchronous data flow programs on a many-core processor," in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, pp. 67–76, ACM, 2016.
- [4] B. Rouxel, S. Skalistis, S. Derrien, and I. Puaud, "Hiding communication delays in contention-free execution for spm-based multi-core architectures," in *31th Euromicro Conference on Real-Time Systems (ECRTS19)*, 2019.
- [5] "Odroid-xu4." <https://wiki.odroid.com/odroid-xu4/odroid-xu4>. Accessed: 2019-09-06.
- [6] "Nvidia jetson." <https://www.nvidia.com/en-us/autonomous-machines/embedded-systems-dev-kits-modules/>. Accessed: 2019-09-06.
- [7] "Teamplay public deliverable 7.5: Achievements in the technical work packages m9." https://gitlab.inria.fr/TeamPlay_Public/TeamPlay_Public_Deliverables/blob/master/D7.5.pdf, 2018. Accessed: 2019-09-05.
- [8] C. Grelck, S.-B. Scholz, and A. Shafarenko, "Asynchronous stream processing with s-net," *International Journal of Parallel Programming*, vol. 38, no. 1, pp. 38–67, 2010.
- [9] G. Ciatto, S. Mariani, M. Louvel, A. Omicini, and F. Zambonelli, "Twenty years of coordination technologies: State-of-the-art and perspectives," in *International Conference on Coordination Languages and Models*, pp. 51–80, Springer, 2018.
- [10] B. Rouxel, S. Derrien, and I. Puaud, "Tightening Contention Delays While Scheduling Parallel Applications on Multi-core Architectures," *ACM Trans. Embed. Comput. Syst.*, vol. 16, no. 20, pp. 1–20, 2017.
- [11] S. Z. Sheikh and M. A. Pasha, "Energy-efficient multicore scheduling for hard real-time systems: A survey," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 17, no. 6, p. 94, 2018.
- [12] ARM Ltd., "White Paper: big. LITTLE Technology : The Future of Mobile," p. 12, 2013.
- [13] "Opencl." <https://www.khronos.org/opencl/>. Accessed: 2019-09-09.
- [14] J. Roeder, B. Rouxel, and C. Grelck, "Towards time-, energy- and security-aware functional coordination," in *IFL 2019 - 31st Symposium on Implementation and Application of Functional Languages*, 2019.
- [15] EU H2020, "TeamPlay Project," 2018. <https://teamplay-h2020.eu/>.
- [16] J. Redmon, "Darknet: Open source neural networks in c." <http://pjreddie.com/darknet/>, 2013–2016.
- [17] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.
- [18] G. G. Brown and R. F. Dell, "Formulating integer linear programs: A rogues' gallery," *INFORMS Transactions on Education*, vol. 7, no. 2, pp. 153–159, 2007.
- [19] I. Griva, S. G. Nash, and A. Sofer, *Linear and nonlinear optimization*, vol. 108. Siam, 2009.