PathTracing: Raising the Level of Understanding of Processing Latency in Heterogeneous MPSoCs

Claudion Rubattu, Francesca Palumbo, Shuvra S Bhattacharyya, Maxime Pelcat

PathTracing: Raising the Level of Understanding of Processing Latency in Heterogeneous MPSoCs

Claudio Rubattu
University of Sassari
Sassari, Italy
crubattu@uniss.it
INSA Rennes, IETR UMR CNRS 6164
Rennes, France
claudio.rubattu@insa-rennes.fr

Francesca Palumbo
University of Sassari
Sassari, Italy
fpalumbo@uniss.it

Shuvra S. Bhattacharyya
University of Maryland
College Park, USA
ssb@umd.edu

Maxime Pelcat
INSA Rennes, IETR UMR CNRS 6164
Rennes, France
maxime.pelcat@insa-rennes.fr

ABSTRACT

Understanding and predicting response time is a major concern in most systems. However, the complexity of heterogeneous Multi-processor Systems-on-Chips (MPSoCs) makes it difficult to provide early evaluation of system execution latency when executing parallel applications. In particular, knowledge about the factors that determine latency is a must in order to effectively drive system-level scheduling and applicative design decisions.

In this paper, we aim at demonstrating that a novel knowledge level is required for analyzing the key factors that influence system execution latency. For that purpose, we propose the concept of Jaccard Gantt similarity score and demonstrate that the straightforward method consisting in scheduling a Directed Acyclic Graph (DAG) of tasks, each with a Deterministic Actor Execution Time (DAET) set from individual task characterization, leads to low Jaccard Gantt similarity scores. We thus propose a new level of system analysis, called PathTracing, that relies on an evaluation of the application critical path and on an analysis of the interferences caused both by scheduling and by architectural costs.

KEYWORDS

model-based design, early performance analysis, MPSoC, design space exploration, processing latency, learning-based model.

ACM Reference Format:
Claudio Rubattu, Francesca Palumbo, Shuvra S. Bhattacharyya, and Maxime Pelcat. 2020. PathTracing: Raising the Level of Understanding of Processing Latency in Heterogeneous MPSoCs. In *RAPIDO '21: 13th Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools, Jan 20, 2021, Budapest, Hungary*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

RAPIDO '21, Jan 20, 2021, Budapest, Hungary

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

High performance embedded systems and Cyber-Physical Systems (CPS) now process, close to sensors, complex workloads that need to be spread over heterogeneous and specialized Processing Elements (PEs) to comply with systems timing constraints. In this context, performance measurement is crucial in achieving efficient solutions with respect to the relevant Key Performance Indicators (KPIs), such as throughput and latency. Latency, also called *system execution latency* or *response time*, has varied definitions in various communities. Indeed, its definition requires the notion of a unit of execution, potentially indefinitely repeated on input data streams, whose lifetime defines latency. Regardless of the type of applicative workload (signal processing, stream processing, batch processing, etc.), response time is determined by a succession of causal, time consuming mechanisms that are usually gathered into a Directed Acyclic Graph (DAG) of data-dependent tasks, representing one execution iteration. The application DAG then serves as the entry point for execution time studies [4]. On the hardware level, heterogeneous MPSoCs are efficient solutions when executing multi-functional applications with workloads that can vary depending on timing requirements. However, when an application is parallelized and scheduled on an MPSoC, the performance of the system in terms of response time is difficult to predict and understand from application and architecture models. Indeed, latency is a highly non-linear property, affected by many software, hardware, and scheduling phenomena.

This paper demonstrates that predicting an execution latency by scheduling an application Directed Acyclic Graph (DAG) and setting Deterministic Actor Execution Time (DAET) timings (i.e. average timings statistically representative of the unitary processing time of a task) leads to a very poor modeling, even for static stream processing applications. Indeed, the structure of the parallel execution, as represented by a Gantt chart, is lost, and the execution variabilities due to cache misses and inter-process synchronizations make such predictions unrealistic.

As a consequence, the paper calls for, and present preliminary results on, a new level of system execution knowledge. This knowledge level will be called PathTracing throughout the document. It

consists of extracting from an application model a Longest-Latency Path (LLP), i.e. a list of causal elements that embed both application-related latency causes (the critical path) but also hardware related interferences increasing the real latency.

Generalizing this concept, PathTracing is a new form of *activity*, as it represents the emphpressure that an application workload puts on a hardware subtrate, resulting in a particular KPI [15]. In the current context, the LLP is the application activity for system execution latency. Having such information on the application workload opens up a large set of studies, ranging from abstract Model of Architecture (MoA) design to scheduling optimizations and design space exploration.

To the extent of our knowledge, this paper constitutes the first effort to decompose from application behaviors the determining factors of system response latency in an MPSoC. The contributions of the paper are:

- a method to compare Gantt charts,
- a demonstration that classically simmulated Gantt charts do not match real executions,
- a proposition of a new knowledge level for analyzing system latency.

The paper is organized as follows: Section 2 proposes an analysis of the different levels of architecture knowledge a designer can incorporate in a latency evaluation method, introducing the new PathTracing level. Section 3 proposes a new metric to compare two Gantt charts and Section 4 uses this metric on use cases to show that a latency evaluation exploiting exact scheduling information as well as DAET extracted from tasks profiling fails modeling the real Gantt chart. Finally, Section 5 shows on the same examples what information can be displayed by using PathTracing, before Section 6 concludes the study.

2 UNDERSTANDING THE CAUSES OF PROCESSING LATENCY IN A REAL-LIFE MPSOC

This section discusses the methods that can be employed to simulate the execution on an MPSoC of an application represented by a DAG of data-triggered tasks, where vertices are tasks and directed edges are messages sent from one task to another. We specifically target a single-rate DAG (srDAG) representation, a subset of the Synchronous Dataflow (SDF) [9] Model of Computation (MoC) where all data produced by a task on a given edge are consumed by the reveiving task when it executes. Such an srDAG can be generated either from a static tagged model such as SDF to represent a period in a periodic time behavior, or can be generated from a parameterized or dynamic application MoC to represent applicative behavior for a limited portion of time. The srDAG has the advantage of simplicity at the cost of a lack of scalability for heavily multirate applications (e.g., see [6]). This srDAG entry point (depicted in Figure 1a) is chosen because it is common to many studies in the literature (e.g., [2, 4, 10–12]) and can be generated from many applicative representations, ranging from advanced Dataflow (DF) algorithms (such as SDF, Cyclo-Static Dataflow (CSDF) [1] or Parameterized and Interfaced Synchronous Dataflow (PiSDF) in our case [3]) to real-time application task sets.

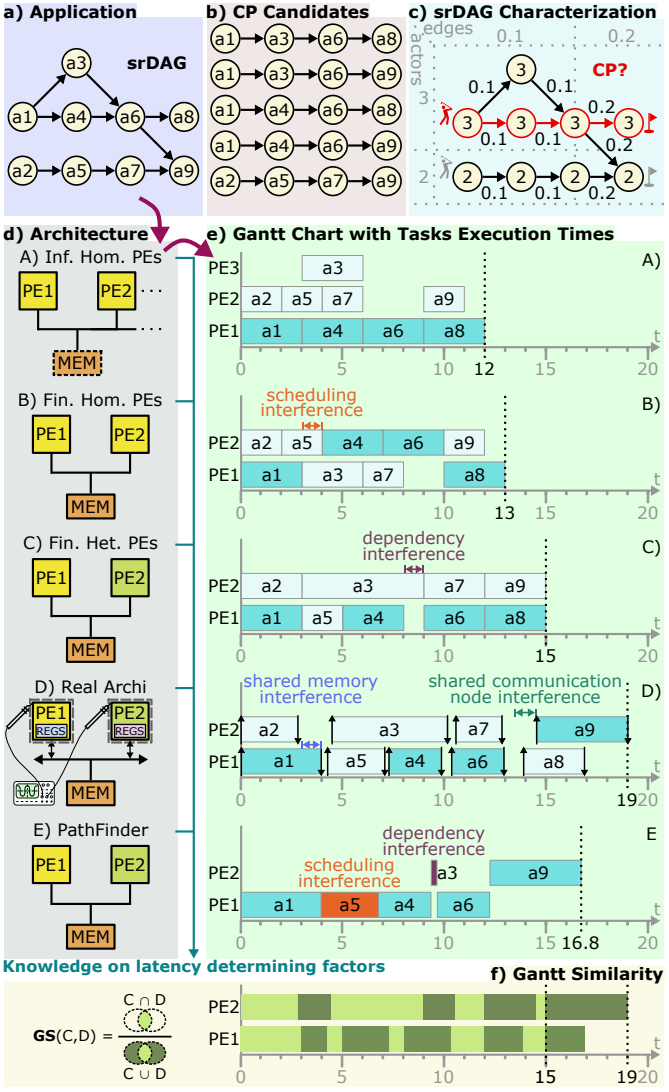

Figure 1: Latency evaluation strategies depending on the level of architecture knowledge.

2.1 Discussion on the Levels of Exploitable Architectural Information

The next sections list incremental levels of architectural knowledge that can be used when modeling response time of a srDAG on an MPSoC. The discussion starts low architectural knowledge (A) up to observation of the running target with functional code (D) and the new proposed PathTracing level (E).

*2.1.1 Level A – **DAG with Sigle DAET Tags**.* Several works have analysed the theoretical latency of the DAG depending on its Critical Path (CP) [7, 13, 17]. As shown in Figure 1b, the DAG is divided into task chains starting from each source node (with no predecessor) and ending to every sink node (with no successor), that we will call *paths*. The latency evaluation from CP analysis is usually

performed by using heuristic approaches based on path exploration and task characterizations. For each CP candidate, DAG nodes are tagged with timing weights. Thus, the CP is characterized by the following assumptions: i) it consists of a task chain that maximizes the sum of contributions, and ii) the CP determines the latency of the whole application (as shown in Figure 1c. In this context, since no information about the architecture is used, CP is considered to freely run on a system with unbounded (or large enough) memory and infinite (or as many as necessary) number of homogeneous PEs, as depicted in Figure 1d-A).

2.1.2 Level B – ***DAG with Sigle DAET Tags and Architecture with a Specified Number of PEs****.* With a known number of homogeneous PEs and a bounded memory (see Figure 1d-B), a heuristic-based scheduling/mapping strategy can be applied in order to achieve an optimized solution in terms of timing performance [16]. This procedure leads to a more realistic model of execution in which paths do not need to be analyzed in the latency representation, usually depicted with a Gantt chart. As tasks share PEs and messages share memory, contentions appear on architectural resources among tasks or communication operations. These contentions lead to interferences in DAG execution, which in turn increase latency and, if not predicted, reduce latency predictability.

2.1.3 Level C – ***Architecture with a Specified Number of PEs of Different Types, and DAG tagged with one DAET per PEs Type****.* A more accurate solution in terms of the DAG execution time can be achieved by distinguishing the types of the PEs in the target system (see Figure 1d-C). Nevertheless, with respect to the homogeneous case, this implies to provide latency prediction with the characterization of: i) the tasks for each type of PE, and ii) the elements of the hardware infrastructure. To do so, a Design Space Exploration (DSE) consisting of a statistical evaluation of the target-dependent costs associated to computation and communication may be required.

2.1.4 Level D (Trace) – ***DAG with Start and End Time Tags and Real Architecture****.* Figure 1d-D) highlights the differences between i) the model of the system interconnects and the bus system linking multiple PEs, and ii) the abstract memory and its actual hierarchical structure. This case corresponds to an execution trace, extracting from code executions the start and end times of all tasks. It can be used as the reference Gantt chart to be looked for by models. At this post-execution stage, system latency is perfectly known. However, such a trace does not provide a full knowledge on the execution, as only start and end times of tasks are known but the causal chain of latency-causing phenomena is still unknown.

2.1.5 Proposed Level E: PathTracing – ***DAG with Start and End Time Tags, Full Scheduling Information and Architecture with a Specified Number of PEs****.* We propose PathTracing as a new level of latency explaining system model (Figure 1d-E). This level extracts the LLP as the subset of tasks causing latency. At a PathTracing level, the objective of the analysis is to be able to tag the Gantt chart, knowing whether a particular task execution or message effectively belongs to the Longest-Latency Path (LLP), and to remove all tasks that do not participate to latency. Pathtracing can be evaluated by a heuristic, evaluating start and end times, and tracing the most probable causal chain. Apart from critical path tasks, the LLP may contain different types of interferences. *Scheduling interferences* are tasks which execution delay the execution of a critical path task. *Dependency interferences* are tasks, or parts of tasks representing waiting time by a critical path task for data produced by non-CP tasks. Other types of interferences can be defined and traced, to refine the level of understanding on the parallel execution.

The next sections will demonstrate the need for the PathTracing level by showing the large differences between a predicted Gantt chart of levels A) to C) and the measured Gantt chart of level D. For this purpose, the next section introduces a novel Gantt chart comparison method based on Jaccard distance.

3 ON USING JACCARD SCORE TO EVALUATE A GANTT CHART PREDICTION

For each step in Figure 1d, example Gantt charts are shown in Figure 1e (where the CP actors are depicted in strong blue). Although level C requires a rather high knowledge on tasks execution times and tasks causality, we aim at demonstrating on real use cases that it provides low accuracy estimations and improper explainability of the real execution. For evaluating this mismatch, we translate the concept of *Jaccard score* for comparing Gantt charts. We name this metric Jaccard Gantt Similarity (JGS), and compute it by applying the method of Intersection over Union (IoU) to the context of parallel processing. This score between 0 and 1 indicates the degree of similarity between two Gantt charts X and Y:

$$GS(X,Y) = \sum_{i=0}^{T} [IoU(X_i, Y_i)] \cdot w_i = \\ = \sum_{i=0}^{T} \left[\frac{area(X_i \cap Y_i)}{area(X_i \cup Y_i)} \right] \cdot \frac{area(Y_i)}{area(Y)} \tag{1}$$

where T is the number of tasks present in the srDAG, and X_i and Y_i represent the boxes associated to the execution times of the i-th task in X and Y respectively. X_i and Y_i are represented as rectangles with arbitrary constant height. Since tasks affect execution with different weights, w_i balances the contributions of the different tasks.

With respect to the example shown in Figure 1, a comparison between the Gantt charts of the level C and D leads to a GS = 0.43 (see Figure 1f). In reality in an MPSoC, the mismatch can be much higher, as shown in the next section, due to variations in tasks processing times and communications.

4 GANTT SIMILARITY ANALYSIS APPLIED TO USE CASES

The chosen applications for testing Gantt similarities have been designed in the PREESM dataflow compiler [14], which uses a non-preemptive ASAP multi-core list scheduling [8] to generate code for a variety of MPSoCs. PREESM manipulates parameterized applications so a variety of data-, task-, and pipeline-parallel application versions can be manipulated. Three applications have been evaluated for this study, each with 6 parameter configurations, and run onto an 8-core heterogeneous Exynos processor with 4 ARM Cortex-A7 cores and 4 ARM Cortex-A15 cores [5].

4.1 Use-case Applications

In the context of image/video processing, the selected applications show distinct features in terms of tasks, firing, and parallelism. These have been evaluated in 6 different scenarios by tuning their parameters in order to change the dataflow graph structure (see Table 1)

Work-dominated — Stabilization: The video stabilization application reduces the effects of undesired camera movements during video recording. Post-processing techniques make it possible to compute image motion, leading to the generation of a new video in which movements are compensated. For each frame, the proposed solution[1] consists of a task graph with only one very costly data parallel actor. In this application running on 8 cores, the latency is due more to the total amount of processing (work) than to the application critical path because the critical path is very short.

Balanced Work and Span — Stereo Matching: From the comparison of two different regularized images of the same scene, the stereo matching application computes the depth information per pixel. Indeed, its values corresponds to the distance between the locations of the same pixel in the two views. The considered implementation[2] running on 8 cores presents a latency due to both the total amount of processing (work) and to the application critical path.

Span-dominated — SIFT: In the context of computer vision, Scale Invariant Feature Transform (SIFT) computation is an algorithm used to detect representative features of an image. Keypoints are extracted by the comparison between corresponding points of the input image and of the same image evaluated in its blurred versions and at different resolutions. In particular, the evaluated implementation[3] shows a limited parallelism so the latency is caused by the application critical path.

Table 1: Instances of the edges and actors in the 6 scenarios of the use-case applications.

S	STABILIZATION		STEREO		SIFT	
	actors	edges	actors	edges	actors	edges
1	9	41	24	80	101	550
2	13	57	28	94	118	645
3	15	65	32	102	135	740
4	22	93	38	136	152	835
5	73	297	74	331	172	951
6	127	513	218	1099	222	1229

4.2 Jaccard Gantt Similarity (JGS) Evaluation Results

The level C) of Figure 1 uses advanced knowledge on the target rchitecture. Thus, one would expect that the real Gantt chart of level D) is close to the predicted Gantt chart of level C). However, Figure 2 shows that on all considered applications, the Jaccard Gantt similarity score is low, even carefully selecting DAET timings for task executions. Figure 2 demonstrates that lavel C) is not sufficient to model the latency in a complex MPSoC, and that an analysis of the latency-causing features in the application is needed. In details, for the Stereo application, the accuracy of the Gantt estimation decreases along the scenarios, as the number of actors raises. In the other two applications dominated by work or span, the values of JGS does not present high variations. However, the similarity between the predicted and the measured Gantt charts remains low for all the applications, showing the limits of the latency estimates when.

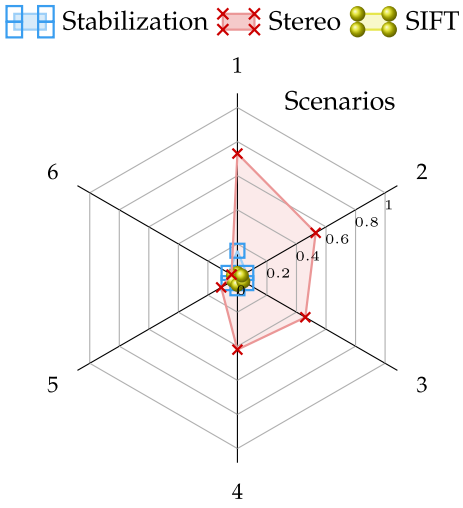

Figure 2: Gantt similarity evaluation through Jaccard score between Levels C (scheduling-based simulation with actors DAET) and D (board measurements). 1 refers to a perfect match. We see that in most case, the match is under 40%, i.e. the Gantt intersection is less than 40% of the Gantt union.

5 PATHTRACING RESULTS ON THE CONSIDERED APPLICATIONS

This section illustrates results that can be obtained by applying PathTracing analysis on the three applications. PathTracing is performed through a heuristic which detailed description is not covered in this publication. After a critical path evaluation through inter-task slack analysis, the heuristic traces successively scheduling and dependency interferences. Figures 3 to 5 show PathTracing results on the three applications in a fixed Scenario.

These results have in fact been used to determine that, on the considered MPSoC, the stabilization application is work-dominated, the stereo matching application is balanced between its work and span contribution, and the Scale-Invariant Feature Transform (SIFT) computation application is span-dominated. Indeed, PathTracing allows a designer to precisely observe if the critical path dominates latency (span domination), if application concurrency "fills" the architecture with processing while making critical path negligible in its effect on latency (work domination), or if critical path and work both have a non-negligible effect on the measure latency. The ratio between CP tasks and interferences shows how much the scheduler has added new tasks in the LLP, increasing latency due to the

[1] github.com/preesm/preesm-apps/.../org.ietr.preesm.stabilization
[2] github.com/preesm/preesm-apps/.../org.ietr.preesm.stereo
[3] github.com/preesm/preesm-apps/.../SIFT

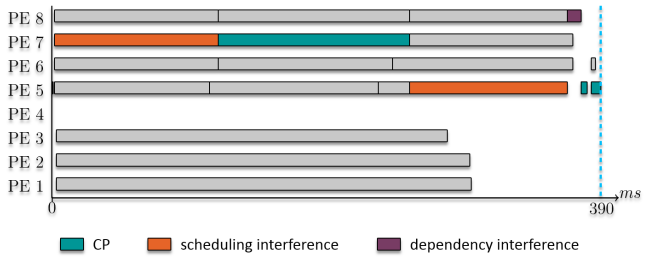

Figure 3: PathTracing of the Stabilization, work-dominated application.

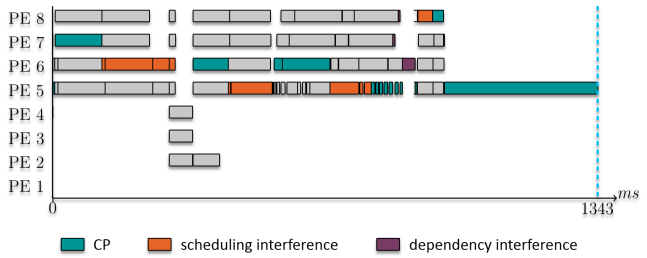

Figure 4: PathTracing of the Stereo Matching, balanced work and span application.

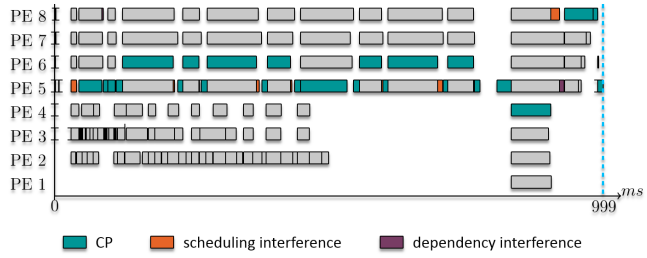

Figure 5: PathTracing of the SIFT point computation, span-dominated application.

amount of work. Additionally, depending on the sophistication of the heuristics used for computing PathTracing, and on the number of defined interference categories, more insights can be given on the causes of an underperforming parrallel execution.

6 CONCLUSION

In this work, a new level of multicore execution analysis has been proposed, and called PathTracing. PathTracing aims at explaining the factors determining system execution latency in an MPSoC. A new metric called Gantt Jaccard Similarity has been introduced in order to evaluate Gantt Chart similarity between a simulation and the real execution of an application. Tested on three use cases running on an 8-core heterogeneous MPSoC, this Gantt assessment shows the strong difficulties related to predicting and understanding response time based on DAET. Moreover, even knowing exact start and end dates of actors execution (level D), the cause of potential execution underperformances remain unknown. In this context, PathTracing aims at providing latency explainability for applications described by a single-rate (sr)DAG MoC tagged with monitored execution times by highlighting in the Gantt chart the tasks that do affect latency, and by classifying them into either critical path or a form of interference.

The considered applications for the demonstration are image/video stream processing applications and cover span-dominated, work dominated, as well as balanced work/span application cases. Task processing time cost is hypothesized to dominate inter-task communication costs, and the list of tasks effectively causing latency is referred to as Longest-Latency Path (LLP).

Apart from demonstrating the need for such a new level of execution understanding, this paper shows preliminary results on PathTracing-level latency analysis. Future work will aim at consolidating these results, generating explainable Gantt charts that will offer a new tool to system designers for evaluating their MPSoC-based designs.

ACKNOWLEDGMENTS

This research received funding from the European Union's under grant agreements No 783162 (FitOptiVis ECSEL project) and No 732105 (CERBERO H2020 project). Prof. F. Palumbo is grateful to the University of Sassari supporting her research activity through the "fondo di Ateneo per la ricerca 2019".

REFERENCES

[1] G. Bilsen et al. 1996. Cycle-static dataflow. *IEEE Transactions on Signal Processing* 44, 2 (Feb 1996), 397–408.
[2] Vincenzo Bonifaci et al. 2019. A Generalized Parallel Task Model for Recurrent Real-Time Processes. *ACM Transactions on Parallel Computing* 6, 1 (June 2019).
[3] Karol Desnos et al. 2013. Pimm: Parameterized and interfaced dataflow meta-model for mpsocs runtime reconfiguration. In *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*.
[4] Adi Fuchs et al. 2019. The accelerator wall: Limits of chip specialization. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*.
[5] Hardkernel co. Ltd. [n.d.]. *ODROID-XU3*. https://www.hardkernel.com/shop/odroid-xu3/
[6] C. Hsu et al. 2007. Efficient Simulation of Critical Synchronous Dataflow Graphs. *ACM Transactions on Design Automation of Electronic Systems* 12, 3 (2007).
[7] Torsten Kempf, Gerd Ascheid, and Rainer Leupers. 2011. Multiprocessor Systems on Chip: Design Space Exploration.
[8] Yu-Kwong Kwok et al. 1999. Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors. *Comput. Surveys* 31, 4 (1999), 406–471.
[9] E. A. Lee et al. 1987. Synchronous data flow. *Proc. IEEE* 75, 9 (Sep. 1987), 1235–1245.
[10] Y. Li et al. 2017. Energy Optimization With Dynamic Task Scheduling Mobile Cloud Computing. *IEEE Systems Journal* 11, 1 (2017), 96–105. https://doi.org/10.1109/JSYST.2015.2442994
[11] A. Melani et al. 2015. Response-Time Analysis of Conditional DAG Tasks in Multiprocessor Systems. In *2015 27th Euromicro Conference on Real-Time Systems*. https://doi.org/10.1109/ECRTS.2015.26
[12] Tony Nowatzki et al. 2013. A General Constraint-Centric Scheduling Framework for Spatial Architectures. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*.
[13] M. Pelcat et al. 2012. *Physical Layer Multi-Core Prototyping: A Dataflow-Based Approach for LTE eNodeB*. Springer-Verlag London.
[14] M. Pelcat et al. 2014. Preesm: A dataflow-based rapid prototyping framework for simplifying multicore DSP programming. In *2014 6th European Embedded Design in Education and Research Conference (EDERC)*.
[15] M. Pelcat et al. 2018. Reproducible Evaluation of System Efficiency With a Model of Architecture: From Theory to Practice. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37, 10 (2018), 2050–2063.
[16] A. K. Singh et al. 2013. Mapping on multi/many-core systems: Survey of current and emerging trends. In *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*.
[17] Oliver Sinnen. 2007. Task Scheduling for Parallel Systems. In *Wiley series on parallel and distributed computing*.