Just-In-Time Scheduling Techniques for Multicore Signal Processing Systems

Julien Heulot, Maxime Pelcat, Jean-François Nezan, Yaset Oliva, Slaheddine Aridhi, Shuvra S. Bhattacharyya

Just-In-Time Scheduling Techniques for Multicore Signal Processing Systems

Julien Heulot*, Maxime Pelcat*, Jean-François Nezan*, Yaset Oliva*, Slaheddine Aridhi**, Shuvra S. Bhattacharyya§

*IETR, INSA Rennes, CNRS UMR 6164, UEB
Rennes, France
email: jheulot,mpelcat,jnezan,yoliva@insa-rennes.fr

§University of Maryland
College Park, USA
Email: ssb@umd.edu

**Texas Instruments France
5 Chemin Des Presses, Cagnes-Sur-Mer
email: saridhi@ti.com

***Abstract*—This paper introduces a novel multicore scheduling method that leverages a parameterized dataflow Model of Computation (MoC). This method, which we have named Just-In-Time Multicore Scheduling (JIT-MS), aims to efficiently schedule Parameterized and Interfaced Synchronous DataFlow (PiSDF) graphs on multicore architectures. This method exploits features of PiSDF to find locally static regions that exhibit predictable communications. This paper uses a multicore signal processing benchmark to demonstrate that the JIT-MS scheduler can exploit more parallelism than a conventional multicore task scheduler based on task creation and dispatch. Experimental results of the JIT-MS on an 8-core Texas Instruments Keystone Digital Signal Processor (DSP) are compared with those obtained from the OpenMP implementation provided by Texas Instruments. Results shows latency improvements of up to 26% for multicore signal processing systems.**

I. Introduction

An important evolution in embedded processing is the integration of increasingly more Processing Elements (PEs) in the Multiprocessor Systems-on-Chip (MPSoC) devices [1], [2], [3], [4]. This trend is mainly due to limitations in the processing power of individual PEs as a result of power consumption considerations.

Concurrently, signal processing applications are becoming increasingly dynamic in terms of hardware resource requirements. This tendency is due to the growing complexity of algorithms allowing higher levels of performance in aspects such as data compression, transmission efficiency, and precision of data analysis. For example, the Scalable High Efficiency Video Coding (SVC) video codec provides a mechanism to temporarily reduce the resolution of a transmitted video in order to match the instantaneous bandwidth of a network [5].

One of the main challenges of the design of multicore signal processing systems, is to distribute computational tasks efficiently onto the available PEs while taking into account dynamic changes. The process of assigning, ordering and timing actors on PEs in this context is referred to as *multicore scheduling*. Inefficient use of the PEs affects latency and energy consumption making multicore scheduling a very important problem to solve [6].

This paper describes a novel method called JIT-MS to address this challenge. JIT-MS is a flexible scheduling method that determines scheduling decisions at run-time to optimize the mapping of an application onto multicore processing resources. In relation to the scheduling taxonomy defined by Lee and Ha [7], JIT-MS is a *fully dynamic* scheduling strategy.

Singh presents a survey on multi/manycore mapping methodologies in [19]. In the context of the taxonomy used in Singh's survey, our methodology can be classified as "On-the-fly" mapping, targeting heterogeneous platforms with a centralized resource management strategy.

This work is supported by the ANR COMPA project.

Applications managed by the JIT-MS scheduler are described using the PiSDF dataflow Model of Computation (MoC), which is related to the general Dataflow Process Network (DPN) MoC. DPN MoCs are widely used in design of signal processing systems [8]. A distinguishing feature of PiSDF is the integration of a parameter tree to asynchronously transmit control values between actors [9].

The JIT-MS scheduling method is embedded in the Synchronous Parameterized and Interfaced Dataflow Embedded Runtime (SPIDER) [10]

This paper is organized as follows: Section II and Section III present related research, providing the context of current work, Section IV details our proposed new JIT-MS scheduling method, and Section V presents our experimental results with JIT-MS scheduling.

II. Related Works

Various frameworks based on OpenMP [11] and OpenCL [12] language extensions are currently proposed to address the multicore scheduling challenge. However, these extensions are based on imperative languages (e.g., C, C++, Fortran) that do not provide mechanisms to model specific signal flow graph topologies. On the contrary, signal processing oriented dataflow MoCs are widely used for specification of data-driven signal flow graphs in a wide range of application areas, including video decoding [13], telecommunication [14], [15], and computer vision [16]. The popularity of dataflow MoCs in design and implementation of signal processing systems is due largely to their analyzability and their natural expressivity of the concurrency in signal processing algorithms, which makes them suitable for exploiting the parallelism offered by MPSoCs.

Synchronous DataFlow (SDF) [17] is the most commonly used DPN MoC for signal processing systems. Production and consumption rates of *actors* (pieces of computation) are set by *firing rules*. These rates are fixed scalars in an SDF graph. Data values, encapsulated by *tokens*, are passed along the edges (First In, First Out data queues (FIFOs)) of a dataflow graph as it executes. Initial tokens, called *delays* can be set on FIFOs.

The PiSDF dataflow MoC [9] results from the addition of the Parameterized and Interfaced Meta Model (PiMM) to the SDF MoC. PiMM extends the semantics of a targeted dataflow MoC by introducing specific notions of hierarchy, interfaces, and parameters. Parameters in PiMM can influence, both statically and dynamically, different properties of a DPN, such as the firing rules of actors. The meta model introduces *configuration actors*, i.e. specific actors that can modify parameter values.

Neuendorffer, et al. define *quiescent points* as points where parameters influencing an execution are allowed to change [18]. Between two quiescent points, the application can be considered static. In this paper, decisions are taken Just-In-Time, immediately after the quiescent points are reached, unveiling new application parallelism.

JIT-MS is an evolution of the work in [14] presenting an adaptive scheduler of parameterized dataflow MoC. However, this work did not consider application hierarchy and was focused only on 3rd Generation Partnership Project (3GPP) Long Term Evolution (LTE) base stations. Our work on JIT-MS goes beyond the methods of [14] to take application hierarchy into account and address a broad class of signal processing applications through generalized scheduling techniques.

III. Context

A. Runtime Architecture

The method developed in this paper is applicable to heterogeneous platforms. In such platform, optimized local decision to start an actor computation (e.g., based on earliest availability of input data) can be inefficient in a global sense. In order to take effective decisions globally, a Master/Slave execution scheme is chosen for the system.

The JIT-MS method requires multiple (software or hardware) components (Figure 1). *Processing Elements (PEs)* are slave components that process actors. They can be of multiple types, such as General-Purpose Processors (GPPs), DSPs, or accelerators. The master of the JIT-MS system is called *Scheduling Element (SE)*. This is the only component that has access to the overall algorithm topology.

Jobs are used to communicate between the SE and PEs. Each PE has a job queue from which it pops jobs out prior to their execution. *Parameters* influence dataflow graph topology or execution timing of actors. When a parameter value is set by a configuration actor, its value is sent to the SE via a parameter queue. Finally, *Data FIFOs* are used by the PEs to exchange data tokens. A data FIFO can be implemented for instance over a shared memory or a network-on-chip.

B. Benchmark

We illustrate the JIT-MS scheduling algorithm by the scheduling of a benchmark application. This benchmark is an extension of the *MP-sched* benchmark [20].

The MP-sched benchmark can be viewed as a two-dimensional grid involving N channels, where each branch consists of M cascaded Finite Impulse Response (FIR) filters of NbS samples. Here, we extend the MP-sched benchmark by allowing the M parameter to vary across different branches. We refer to this extended version of the MP-sched benchmark as *heterogeneous-chain-length MP-sched* (*HCLM-sched*).

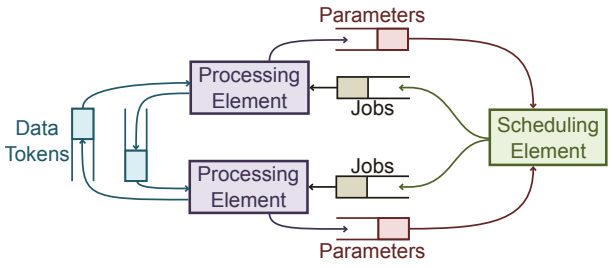

Fig. 1: Runtime execution scheme.

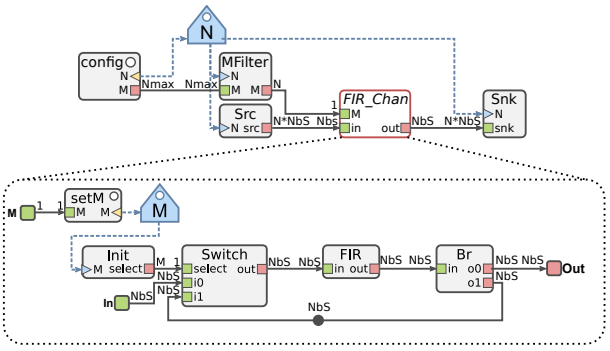

Fig. 2: A PiSDF model of the HCLM-sched benchmark.

A PiSDF representation of the HCLM-sched benchmark is shown in Figure 2. To represent the channels in the HCLM-sched benchmark, a hierarchical actor called FIR_Chan is introduced. The top level is designed to repeat N times this actor. In the subgraph describing the behavior of the FIR_Chan actor, M pipelined FIR filter repetitions in the branches are handled by a feedback loop and specific control actors ($Init$, $Switch$ and $Broadcast$).

C. Notations

To describe JIT-MS, the following notation is used. CA represents the set of configuration actors of the given PiSDF graph. Thus, $\overline{CA}$ represents all actors in the given PiSDF graph that are not configuration actors.

IV. Just-In-Time Multicore Scheduling (JIT-MS)

A. Multicore Scheduling of Static Subgraphs

JIT-MS involves decomposing the scheduling of a given PiSDF graph into the scheduling of a sequence $X_1, X_2, \ldots$ of SDF graphs. Different executions (with different sets of input data) can result in different sequences of SDF graphs for the same PiSDF graph. For a given execution, we refer to each X_i as a *step* of the JIT-MS scheduling process for that execution.

On each step, resolved parameters enable the transformation of the PiSDF graph into an SDF graph, which can be scheduled by any of the numerous existing SDF scheduling heuristics that are relevant for multicore architectures [21]. For example, see [22] for a set of techniques that can be applied upon transforming the resulting SDF graph into an single rate SDF (srSDF) graph. An srSDF graph is an SDF graph in which the production rate on each edge is equal to the consumption rate on that edge. A consistent SDF graph can be transformed into an equivalent srSDF graph by applying techniques that were introduced by Lee and Messerschmitt [23].

The Just-In-Time Multicore Scheduling (JIT-MS) method is based on the static multicore scheduling method which is composed of the following sequence of phases:

1) Computing the Basis Repetition Vector (BRV) of the *current graph* (the graph that is presently being scheduled). The BRV, also known as the SDF *repetition vector*, is a positive-integer vector and represents the number of firings of each actor in a minimal periodic scheduling iteration for the graph. We note however, that certain technical details of PiSDF require adaptations to the conventional repetitions vector computation process from [17].
2) Converting the SDF graph into an equivalent srSDF graph, where each actor is instantiated a number of times equal to its corresponding BRV component.

3) Scheduling actors and communications from a derived acyclic srSDF graph onto the targeted heterogeneous platform. Any scheduling heuristic that is applicable to acyclic srSDFs graphs can be chosen here — e.g., the applied schedule can be a list scheduler, fast scheduler, flow-shop or genetic scheduler [19], [24], [22]. Upon completing the scheduling process described, the resulting schedule S is executed.

A complete JIT-MS schedule of a PiSDF hierarchical graph consists of several of these phases, repeated as many times as needed (see Section IV-B).

In a PiSDF graph, some data FIFOs behave as Round Buffers (RBs) [9] — i.e., such FIFOs produce multiple copies of individual tokens as necessary to satisfy consumption demand. In particular, FIFOs at the interface of a hierarchical actor have RBs behavior to help ensure composability in hierarchical specifications. FIFOs connecting configuration actors to other actors also behave as RBs to ensure that configuration actors fire only once per subgraph. Application designers using the PiSDF model of computation need to take such RB behavior into account during the development process.

Configuration Actors and such RBs are excluded from the BRV computation as they are forced to fire only once.

B. Multicore Scheduling of Full Graphs

The JIT-MS method is based on the PiSDF runtime operational semantic. As shown in [9], the JIT-MS scheduler has to proceed in multiple steps, each one unveiling a new portion of srSDF graph for scheduling. In one step, configuration actors have to be fired first, they produce parameters needed to resolve the rest of the subgraph. When all parameters are solved at one hierarchy level, scheduling of other actors of this hierarchy level is made possible.

In a multicore system, the SE has to extract the parallelism of the application to send jobs to multiple PEs. Contrary to static applications, the difficulty of this process is to schedule actors efficiently without knowing the complete graphs. The complete srSDF graph is only known when all configuration actors have been executed.

Once an srSDF graph has been generated, it can be analyzed to exploit the parallelism of the application (Section IV-A). The JIT-MS runtime schedules the actors and communications and fires their execution on the platform. Newly instantiated hierarchical actors are added to a global srSDF graph, called *execution graph*, and the same process can be used until the whole graph has been processed.

To keep track of actor's execution, each actor of the execution graph is tagged with a flag representing its execution state. An actor can be *Run (R)*, *Not Executable (N)* or *Executable (E)*. An actor is *Executable* only when all its parameters are resolved and when all its predecessors are *Executable* or *Run*.

The procedure of JIT-MS scheduling is shown in Algorithm 1.

After initialization, the algorithm enters in a main while loop which computes scheduling steps until there is no more hierarchical actor in the execution graph. A single scheduling step is made of the 3 stages: *graph configuration*, *actor execution* and *graph resolution*.

The first stage (line 3 to 13) replaces each executable hierarchical actor of the execution graph by its configuration actors. As they are only fired once, there is no need to compute the BRV and the graph transformation to srSDF becomes trivial. If there is no configuration actor in this hierarchical actor, all the subgraph parameters can be resolved (using the 2 first phases presented in Section IV-A) and the subgraph can replace the hierarchical actor in the execution graph. In this stage, it is also important to add RBs at the interfaces of the hierarchical actors and between CA and $\overline{CA}$ to respect PiSDF semantics.

Algorithm 1: Multi-Step Algorithm procedure to schedule a PiSDF graph.

```
1  Procedure MultiStep()
2      while {∃ Hierarchical actor in execGraph } do
3          while {∃ Hierarchical actor in execGraph |
           actor.flag = E } do
4              currentPiSDF ← actor.pisdf ;
5              if CA ≠ {∅} then
6                  Replace actor with RBs in execGraph;
7                  Put CA in execGraph;
8                  Add RBs between CA and CA‾ ;
9                  push currentPiSDF → graphFifo;
10             else
11                 computeBRV(currentPiSDF) ;
12                 Add single rate CA‾ graph in execGraph;
13             Update flags in execGraph;
14         Schedule Executable actors ∈ execGraph;
15         Fire Actors and Wait parameter values ;
16         while graphFifo is not empty do
17             pop graphFifo → currentPiSDF;
18             computeBRV(currentPiSDF) ;
19             Add single rate CA‾ graph in execGraph;
20         Update flags in execGraph;
21     Schedule Executable actors ∈ execGraph;
22     Fire Actors ;
```

The second stage (line 14 to 15) assigns, orders and fires executable actors. It corresponds to phase 3 of Section IV-A.

The third stage (line 16 to 20) corresponds to the graph resolution. At this stage, the parameters resolved by configuration actors of the previous stages are used to solve the graph of each hierarchical step. The 2 first phases of Section IV-A can then be applied to fully replace the hierarchical actor in the execution graph with the corresponding child actors.

At the end of the algorithm (line 21 to 22), when no more hierarchical actor is present in the execution graph, a last phase of assignment, ordering and firing of executable actors has to be done to execute all non executed actors.

C. Applying JIT-MS to the Benchmark

The *execution graph* shape at each step of the HCLM-sched benchmark can be seen in Figure 3. In this figure, blue actors are not executable, green ones are executable and black ones are already run. Red dashed actors are hierarchical.

Figure 3.a corresponds to the execution graph state at the end of the first phase of the first iteration of the while loop (loop I.a). At this point, N is set to 2. Then Figure 3.b corresponds to execution graph state after the third phase of the first iteration (loop I.b). The hierarchical FIR_Chan actors are instanciated. Then, Figures 3.c and 3.d correspond to the execution, first of the internal configuration actors of FIR_Chan (SM), then of their actors with parameter $M = \{1, 2\}$.

V. Experimental Results

This paper describes a method called JIT-MS used to parallelize applications at runtime. In this context, experimental results will focus on the comparison between the JIT-MS approach and the OpenMP

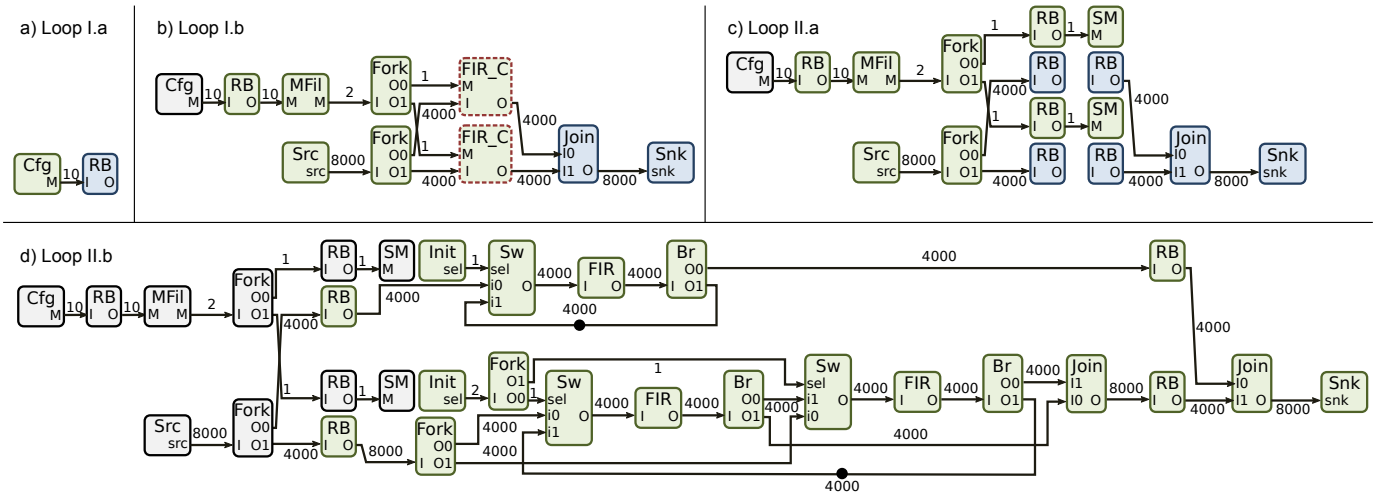

Fig. 3: single rate SDFs (srSDFs) graphs generated from the HCLM-sched benchmark.

framework. Results have been acquired by studying the latency of single and multiple iterations of the HCLM-sched benchmark on the Texas Instruments c6678 multicore DSP [1].

OpenMP is a framework designed for shared memory multiprocessing. It provides mechanisms for launching parallel teams of threads to execute efficiently an algorithm on a multicore architecture. OpenMP applications are designed with a succession of sequential code, executed by a master thread, and parallel code, distributed in a team of threads dispatched onto multiple cores [11].

The platform used for the current experiments is the Texas Instruments Keystone I architecture (EVM TMS320C6678). This multicore DSP platform is composed of 8 c66x DSP cores interconnected by a Network on Chip (NoC) called TeraNet with access to an internal shared memory. To perform synchronization between cores, hardware queues provided by the Multicore Navigator [25] have been used.

The OpenMP framework cannot implement the HCLM-sched as a double nested loop since FIRs are cascaded on each channels. So, OpenMP is used to parallelize channels by using a "parallel for".

For the first experiment, we fix the M value to 8 for all stages, FIR of 4000 samples and we measure latency of one graph iteration. Results on execution time are displayed in Figure 4.a.

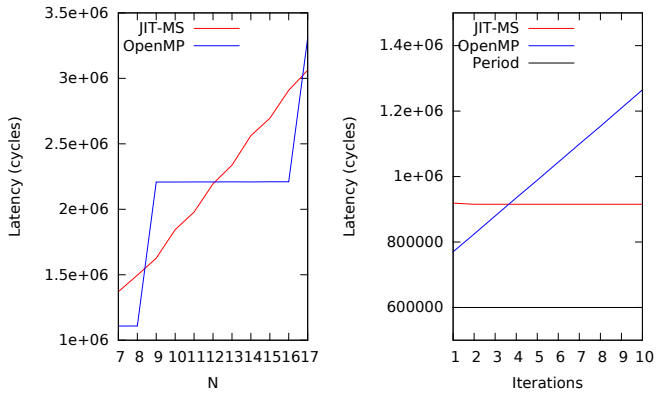

Fig. 4: a) Latency vs N values b) Latency vs multiple iterations

As can be seen on Figure 4.a, the OpenMP implementation latency curve displays a step shape when increasing N. This is due to channels distribution on the platform. Since each stage is implemented as a monolithic block with OpenMP, as soon as 9 channels are reached, 2 channels have to be completed on one PE making the overall latency double.

With the JIT-MS implementation, the graph transformation and scheduling phases introduce a visible overhead but the execution efficiency over varying parameters is smoother. The overhead can be observed on the figure when N equals to 7 or 8 as the resulting scheduling is the same as OpenMP. The transformation to srSDF extracts more parallelism than OpenMP from the subdivision of channels into multiple FIRs. These choices make JIT-MS suitable for unbalanced applications. In the HCLM-sched benchmark with 9 channels, the overall latency is reduced of up to 26%.

The second experiment is based on multiple iterations of the HCLM-sched benchmark. For this experiment, we fixed the number of channels to 8 and M values are linear between 1 to 8. Then, multiple iterations of the application are launched with a fixed period.

As we can see in Figure 4.b, if the latency of the OpenMP implementation is superior to the period, the latency is growing at each new iterations. This is due to the global synchronization at the end of each OpenMP parallelization blocks.

For the JIT-MS implementation, the latency remains constant over iterations. By having prior knowledge on how the application will behave, the Scheduling Element can start an execution on processing elements which have already finished the previous execution. It can then start the following iteration as soon as the next period tick occurs. With a better knowledge of the application execution, the JIT-MS can pipeline graph iterations.

VI. Conclusion

This paper presents a novel multicore scheduling method referred to as Just-In-Time Multicore Scheduling (JIT-MS). JIT-MS splits the scheduling of a PiSDF dataflow graph into steps to identify locally static regions. It enables efficient assignment and ordering of actors into PEs with a better knowledge of actor interactions. Experiments conducted on an 8-core Texas Instruments DSP demonstrate on a benchmark that the JIT-MS scheduler provides more parallelism to the execution than the job posting system based on pragmas, task creation and task dispatch of OpenMP. Results have shown that JIT-MS can reduce the execution latency up to 26% and can allow handling multiple executions.

REFERENCES

[1] *Multicore Fixed and Floating-Point Digital Signal Processor - SPRS691E*, Texas Instruments. [Online]. Available: http://www.ti.com/lit/pdf/sprs691e (accessed 04/2014)

[2] *MPPA MANYCORE: a multicore processors family*, Kalray. [Online]. Available: www.kalray.eu

[3] *Epiphany A breakthrough in parallel processing*, Adapteva. [Online]. Available: www.adapteva.com

[4] L. Benini, E. Flamand, D. Fuin, and D. Melpignano, "P2012: Building an ecosystem for a scalable, modular and high-efficiency embedded computing accelerator," in *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium, 2012, pp. 983–987.

[5] P. Helle, H. Lakshman, M. Siekmann, J. Stegemann, T. Hinz, H. Schwarz, D. Marpe, and T. Wiegand, "A scalable video coding extension of HEVC," in *Data Compression Conference (DCC), 2013*, 2013, pp. 201–210.

[6] P. Marwedel, J. Teich, G. Kouveli, I. Bacivarov, L. Thiele, S. Ha, C. Lee, Q. Xu, and L. Huang, "Mapping of applications to MPSoCs," in *Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. ACM, 2011, pp. 109–118.

[7] E. Lee and S. Ha, "Scheduling strategies for multiprocessor real-time DSP," in *Global Telecommunications Conference and Exhibition'Communications Technology for the 1990s and Beyond'(GLOBECOM), 1989. IEEE*. IEEE, 1989, pp. 1279–1283.

[8] S. S. Bhattacharyya, E. Deprettere, R. Leupers, and J. Takala, Eds., *Handbook of Signal Processing Systems*, 2nd ed. Springer, 2013, iSBN: 978-1-4614-6858-5 (Print); 978-1-4614-6859-2 (Online). [Online]. Available: http://dx.doi.org/10.1007/978-1-4614-6859-2

[9] K. Desnos, M. Pelcat, J.-F. Nezan, S. S. Bhattacharyya, and S. Aridhi, "Pimm: Parameterized and interfaced dataflow meta-model for mpsocs runtime reconfiguration," in *Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIII), 2013 International Conference on*. IEEE, 2013, pp. 41–48.

[10] J. Heulot, M. Pelcat, K. Desnos, J. F. Nezan, S. Aridhi *et al.*, "Spider: A synchronous parameterized and interfaced dataflow-based rtos for multicore dsps," *EDERC 2014 Proceedings*, 2014.

[11] L. Dagum and R. Menon, "Openmp: an industry standard api for shared-memory programming," *Computational Science & Engineering, IEEE*, vol. 5, no. 1, pp. 46–55, 1998.

[12] J. E. Stone, D. Gohara, and G. Shi, "Opencl: A parallel programming standard for heterogeneous computing systems," *Computing in science & engineering*, vol. 12, no. 3, p. 66, 2010.

[13] C. Lucarz, M. Mattavelli, M. Wipliez, G. Roquier, M. Raulet, J. W. Janneck, I. D. Miller, D. B. Parlour *et al.*, "Dataflow/actor-oriented language for the design of complex signal processing systems," in *Conference on Design and Architectures for Signal and Image Processing (DASIP 2008) Proceedings*, 2008, pp. 1–8.

[14] M. Pelcat, J.-F. Nezan, and S. Aridhi, "Adaptive multicore scheduling for the LTE uplink," in *NASA/ESA Conference on Adaptive Hardware and Systems*, 2010, pp. 36–43.

[15] C.-J. Hsu, J. L. Pino, and F.-J. Hu, "A mixed-mode vector-based dataflow approach for modeling and simulating lte physical layer," in *Design Automation Conference (DAC), 2010 47th ACM/IEEE*. IEEE, 2010, pp. 18–23.

[16] M. Sen, I. Corretjer, F. Haim, S. Saha, J. Schlessman, T. Lv, S. S. Bhattacharyya, and W. Wolf, "Dataflow-based mapping of computer vision algorithms onto fpgas," *EURASIP Journal on Embedded Systems*, vol. 2007, no. 1, pp. 29–29, 2007.

[17] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, 1987.

[18] S. Neuendorffer and E. Lee, "Hierarchical reconfiguration of dataflow models," in *Formal Methods and Models for Co-Design, 2004. MEMOCODE'04. Proceedings. Second ACM and IEEE International Conference on*. IEEE, 2004, p. 179188.

[19] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel, "Mapping on multi/many-core systems: Survey of current and emerging trends," in *Proceedings of the 50th Annual Design Automation Conference*. ACM, 2013, p. 1.

[20] G. F. Zaki, W. Plishker, S. S. Bhattacharyya, C. Clancy, and J. Kuykendall, "Integration of dataflow-based heterogeneous multiprocessor scheduling techniques in gnu radio," *Journal of Signal Processing Systems*, vol. 70, no. 2, pp. 177–191, 2013.

[21] S. Sriram and S. S. Bhattacharyya, *Embedded multiprocessors: Scheduling and synchronization*. CRC press, 2012.

[22] Y.-K. Kwok, "High-performance algorithms for compile-time scheduling of parallel processors," 1997.

[23] E. Lee and D. G. Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing," *Computers, IEEE Transactions on*, vol. 100, pp. 24–35, 1987.

[24] J. Boutellier, S. S. Bhattacharyya, and O. Silvén, "A low-overhead scheduling methodology for fine-grained acceleration of signal processing systems," *Journal of Signal Processing Systems*, vol. 60, no. 3, pp. 333–343, 2010.

[25] *KeyStone Architecture Multicore Navigator*, Texas Instruments. [Online]. Available: http://www.ti.com/lit/pdf/sprugr9 (accessed 04/2014)