SPIDER: A Synchronous Parameterized and Interfaced Dataflow-Based RTOS for Multicore DSPs

Julien Heulot, Maxime Pelcat, Karol Desnos, Jean François Nezan, Slaheddine Aridhi

SPIDER: A SYNCHRONOUS PARAMETERIZED AND INTERFACED DATAFLOW-BASED RTOS FOR MULTICORE DSPS

Julien Heulot, Maxime Pelcat*, Karol Desnos*, Jean-François Nezan*, Slaheddine Aridhi***

* IETR, INSA Rennes, UMR CNRS 6164, UEB
20 Avenue des Buttes de Coësmes, 35708, Rennes, France
email: jheulot, mpelcat, kdesnos, jnezan@insa-rennes.fr
web: www.ietr.fr

** Texas Instrument France
5 Chemin Des Presses, 4 Allée Technopolis,
Cagnes-Sur-Mer
email: saridhi@ti.com
web: www.ti.com

ABSTRACT

This paper introduces a novel Real-Time Operating System (RTOS) based on a parameterized dataflow Model of Computation (MoC). This RTOS, called Synchronous Parameterized and Interfaced Dataflow Embedded Runtime (SPiDER), aims at efficiently scheduling Parameterized and Interfaced Synchronous Dataflow (PiSDF) graphs on multicore architectures. It exploits features of PiSDF to locate locally static regions that exhibit predictable application behavior. This paper uses a multicore signal processing benchmark to demonstrate that the SPiDER runtime can exploit more parallelism than a conventional multicore task scheduler. By comparing experimental results of the SPiDER runtime on an 8-core Texas Instruments Keystone I Digital Signal Processor (DSP) with those obtained from the OpenMP framework, latency improvements of up to 26% are demonstrated.

1. INTRODUCTION

The current limitation of the processing power of individual Processing Element (PE) due to power consumption considerations is fostering the integration of more and more PEs into Multiprocessor System-on-Chip (MPSoC) devices such as Texas Instruments' Keystones and other multicore devices. This trend is even more marked because of the rising complexity of applications in the signal processing systems domain.

Concurrently, signal processing applications are becoming increasingly dynamic which leads to more complex hardware resource requirements. Data dependencies between different subsections of an algorithm can also change over time. This fact is due to the growing number of conditional operations in algorithms so as to achieve better performance in terms of latency and reliability. For example, the variation of the number of User Equipments (UEs) scheduled to be transmitted by a base station implementing the 3rd Generation Partnership Project (3GPP) Long Term Evolution (LTE) needs to be taken into account to process each UEs data reliably and efficiently. The modification in the number of scheduled UEs between iterations (in this case with a period of 1ms) leads to high variations in the complexity of uplink and downlink data stream processing [10].

When designing and implementing multicore signal processing systems, one of the main challenges is to dispatch computational tasks efficiently onto the available PEs while also considering dynamic changes in application functionalities and resource requirements. The process of assigning, ordering and timing actors on PEs in this context is referred to as *multicore scheduling*.

In multicore scheduling, inefficient use of the PEs leads to additional costs ranging from longer processing times to higher energy consumption. Data dependencies and dynamic signal processing algorithms make multicore scheduling an interesting and difficult challenge [6].

This paper describes a new RTOS called SPiDER runtime to address these challenges. The SPiDER runtime takes scheduling decisions at runtime as soon as it gets information on the current algorithm topology. In relation to the scheduling taxonomy defined by Lee and Ha [4], this RTOS implements a *fully dynamic* scheduling strategy.

Instead of Tasks and Threads as in other RTOS, applications managed by SPiDER runtime are described using the PiSDF dataflow MoC, which is a specialization of the general Dataflow Process Network (DPN) MoC. The DPN MoC decomposes algorithms into pieces of computation, called *actors*, that exchange data, decomposed into *tokens* (atomic data element), through *First In, First Out data queues (FIFOs)*. Actors and FIFOs compose a directed graph called *dataflow graph*.

The PiSDF model is obtained through the use of Parameterized and Interfaced dataflow Meta-Model (PiMM) over a Synchronous DataFlow (SDF) graph. PiMM integrates two main features: an acyclic graph of parameters to transmit control values between actors, and an interfaced hierarchical scheme ensuring schedulability over hierarchy exploration [2]. The SPiDER runtime exploits these features of the PiSDF model to efficiently extract parallelism and reduce the overall latency of a dataflow graph execution. The SPiDER runtime is an open source project[1].

This paper is organized as follows. After Section 2 which presents related works, Section 3 describes the SPiDER runtime components and Section 4 details its implementation on the TMS320C6678 platform. Finally, Section 5 presents our experimental results.

2. RELATED WORKS

To address the multicore scheduling challenge, various frameworks based on OpenMP [1] and OpenCL [13] language extensions are currently proposed. However, these extensions are based on imperative languages (e.g., C, C++, Fortran) that do not provide mechanisms to specify signal algorithms with complex task parallelism.

The Open Event Machine (OpenEM)[2] is a multicore runtime for Texas Instruments Keystone platforms. It allows dispatching tasks on several working cores following prioritized queues of tasks called *events*. It can be seen as hardware abstraction layer for the SPiDER runtime as it mainly proposes tools for multicore programming but does not provide complex scheduling strategies. The OpenMP framework implementation for Keystone platforms is implemented over OpenEM.

Dataflow MoCS are widely used for specification and implementation of data-driven signal algorithms in many application areas [5], telecommunication [10], and computer vision [11]. The popularity of dataflow MoCS in the design and implementation of signal processing systems is largely due to their analyzability, their predictability and their natural expressivity of task parallelism in signal processing algorithms.

In [2], Desnos and al. define a meta model called PiMM that can be applied to an SDF MoC to obtain PiSDF. This meta model

This work is supported by the ANR COMPA project.

[1] http://github.com/COMPA-Runtime/COMPA-Runtime
[2] http://sourceforge.net/projects/eventmachine/

brings multiple features such as hierarchy interfaces and parameterization. Hierarchy interfaces make possible to explore one hierarchy level entirely without any information on the inner parameterization of the subgraphs. Parameterization introduced by PiMM can mix data flow and parameter flow making possible complex parameterization of the application. The meta model also introduces *configuration actors* having a dedicated firing rule and able to set parameter values dynamically for the current subgraph.

The SPiDER runtime is an evolution of the work in [10]. In [9], Oliva and al. propose an RTOS based on this previous work. It uses the μC/OS II Operating System (OS) and a Master/Slave pattern for the runtime architecture. This work takes as input dataflow MoC a Parameterized Cyclo-Static Directed Acyclic Graph (PCSDAG) that does not consider application hierarchy and feedback loops. This work was also focused only on a specific application, the 3GPP LTE Uplink Physical Layer data processing (PUSCH) algorithm. The method presented in this paper is based on another dataflow MoC called PiSDF. This MoC allows more flexible parameterization schemes, not only as preprocessing. This MoC also handles hierarchical programming and feedback loops.

Nollet and al. define a taxonomy of MPSoC runtime architectures in [8]. They define terms such as *Quality Manager*, and *Runtime Library*. The SPiDER runtime introduces a Quality Manager called *Global Runtime* based on PiSDF MoC and a Runtime Library called *Local Runtime*. These both components are platform independent.

In [7], Neuendorffer, et al. define *quiescent points* as points where parameters influencing an execution are allowed to change. Between two quiescent points, the application can be considered static. In this paper, decisions taken by the SPiDER runtime on actor ordering and mapping are taken after the quiescent points are reached.

The Jade (Just-in-time adaptive decoder engine) scheduler from Gorin, et al. [3] is a scheduler based on dataflow methods. The difference between the Jade scheduling method and the SPiDER runtime is that Jade is based on the CAL language implementing the dynamic dataflow MoC. This model gives knowledge on quiescent points only after graph execution, providing a posteriori informations rather than predictability information.

In [12], Singh presents a survey on multi/manycore mapping methodologies. The SPiDER runtime can be classified as "On-the-fly" mapping, targeting heterogeneous platform with a centralized resource management strategy. This survey does not reference any "On-the-fly" mapping method based on dataflow.

3. SPIDER RUNTIME

3.1 DataFlow MoC

The SPiDER runtime is based on a Dataflow MoC. The MoC chosen for this RTOS is PiSDF for its parameterization and hierarchy properties. The predictability of this MoC allows taking relevant scheduling decisions a priori without a strong penalty on its expressivity.

An example of a PiSDF graph can be seen in Figure 1. The parameters are variables that can influence production and consumption of data tokens by actors in the graph. They can be static such as NbS which is constant for the whole graph execution or they can be set dynamically; e.g. N and M are set by the *config* and $setM$ actors respectively. These actors are called *configure actors* because they have special firing rules related to their use in updating parameter values. Configuration actors (the ones marked a small white circle in Figure 1) are fired only once at the beginning of the execution of the hierarchical graph containing them. The FIR_Chan actor is hierarchical and contains, as a nested subgraph, the graph shown under it. More details about PiSDF can be found in [2].

In order to use most of the parallelism of a PiSDF representation, the method developed in this paper transforms the PiSDF graph into an single rate Synchronous DataFlow (srSDF) graph at runtime as soon as parameters are resolved. An srSDF graph is an SDF graph where token production and consumption rates are equal on each edge. This is achieved by replicating actors that need to be fired more than once, thus meaning that each actor of srSDF graph is executed only once in the graph execution. Doing so, the scheduler has a global view of actors dependencies for the whole graph iteration.

Since parameters are resolved after all configuration actors have been executed, the PiSDF to srSDF transformation cannot be completed at once. Each hierarchy level has to be configured sequentially as the lower hierarchy level depends on parameters and/or data coming from the upper one. To perform this transformation, multiple steps of scheduling need to be completed, each step configuring a hierarchy level and then revealing a part of the srSDF graph.

Once this srSDF graph has been generated, the scheduler can dispatch actors onto the MPSoC. To do so, the process is separated into: *task ordering* and *mapping*. The task ordering consists of sorting all non-executed tasks of the srSDF graph into one list. This list is then used to dispatch tasks onto each PE of the MPSoC. This task is called mapping. Both the tasks of ordering and mapping can be optimized to improve different metrics such as: latency, throughput, memory utilization or energy efficiency.

3.2 RTOS Topology

One of the uses of the SPiDER runtime is on heterogeneous platforms. In this usage, a local decision on actor firing may lead to a bad global decision, as there may be another PE available for this actor which could have permitted earlier completion of the task. In order to ensure efficient global decisions, a Master/Slave execution scheme is thus preferred for heterogeneous platforms.

The SPiDER runtime uses a PiSDF graph as an input algorithm graph. This MoC defines parameters as integer values that influence algorithm execution. Since configuration actors can be executed on any PE of the MPSoC, it is important to send parameters back to the Master. It will then take scheduling decisions based on these parameters.

The runtime (Figure 2) requires the following elements:

- *Local RunTime (LRT)*: low footprint OS that processes actors. It can be implemented over multiple types of PE: General-Purpose Processor (GPP), DSP, accelerator, etc...
- *Global RunTime (GRT)*: This is the master of the system, and knows the algorithm topology and takes multicore scheduling decisions. It is usually implemented over a GPP core but can also be a DSP core. The GRT can also process actors.
- *Data tokens*: atomic data exchanged by actors. A data token has a predefined size: it can be one bit, one byte, a data structure, etc... A data FIFO can be implemented over any data medium (e.g. a shared memory or a network on chip).
- *Jobs*: a job embeds all data required to execute one instance of an actor. In particular, it includes information on actor code location, which FIFO receives input data and which FIFO sends output data. Each LRT has a job queue.

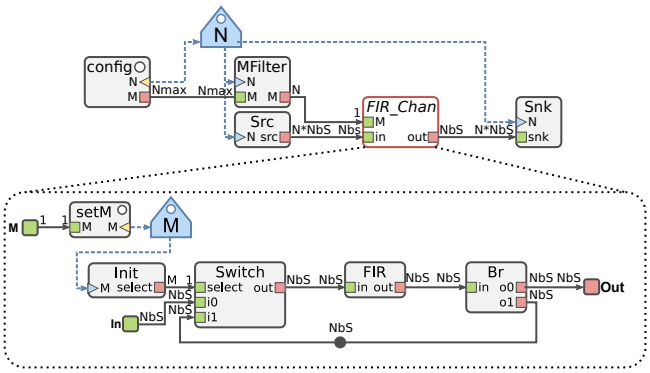

Figure 1: PiSDF representation of HCLM-sched benchmark

- *Parameters*: A parameter influences the algorithm graph topology or the execution timing of actors. When it is set by a configuration actor processed by a LRT, its value is sent to the GRT via a dedicated queue.
- *Timings*: To have timing feedback on the previous and the current execution, LRTs send back actor start and end times through low priority timing FIFOs. All timing information is based on the same timing reference.

4. IMPLEMENTATION ON C6678

The experimental platform used is the Texas Instruments Keystone I architecture (EVM TMS320C6678) composed of 8 c66x DSP cores. They are interconnected by a Network-on-Chip (NoC) called TeraNet which accesses an internal shared memory called MSMC. The Keystone platform also provides an external memory access to DDR memory.

Control queues for parameters, timings and jobs are implemented using the hardware queues present in the Keystone Multicore Navigator [14]. It embeds 8192 hardware queues for multicore synchronization. These queues exchange data through descriptors stored in memory regions. We allocate one memory region in cached MSMC for the descriptors of these control queues. Since control data sent through these queues are relatively small, descriptors are configured monolythic, i.e. all data is present in the descriptor.

For data token communication, data is stored in MSMC or DDR (depending on the memory allocation). Synchronizations between cores is performed using one hardware queue for each data transfer. A descriptor present in a hardware queue means that the corresponding data is available in shared memory. This makes access to shared memory predictable and suitable for enabling caches.

For timing information, the keystone architecture provides 16 shared timers. One of these shared timers is used for global timestamp information. This ensures relevant (global) timing information on current and previous executions.

Figure 3 represents the implementation of the SPiDER runtime on a c6678 keystone architecture. The hardware independent GRT and LRT need services from hardware which are provided by an abstraction layer called *Platform Library*.

5. EXPERIMENTS

This paper describes an RTOS used to distribute efficiently at runtime signal processing applications. In this context, experiments will focus on comparison with another widely used framework called OpenMP. Results have been acquired by studying single and multi-iteration latencies of a benchmark application on the Texas Instruments c6678 multi-core DSP platform.

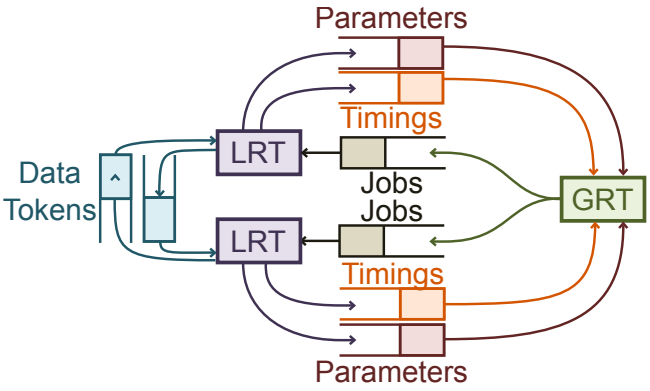

Figure 2: Runtime execution scheme

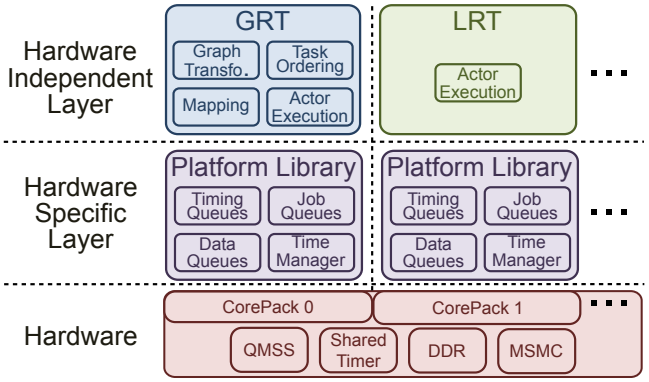

Figure 3: Hardware arch. of SPiDER Runtime on c6678

5.1 Benchmark

To compare our approach and the OpenMP frameworks, we chose a generic benchmark of signal processing. This benchmark is an extension of the *MP-sched* benchmark [15]. The MP-sched benchmark can be viewed as a two-dimensional grid involving N channels, where each channel 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 channels. We refer to this extended version of the MP-sched benchmark as *heterogeneous-chain-length MP-sched* (*HCLM-sched*).

The OpenMP framework cannot implement the HCLM-sched as a double nested loop since FIRs are pipelined on each channels, this data dependencies is not suitable for an OpenMP "parallel for". However, OpenMP framework is used to parallelize channels making them monolithic tasks.

In PiSDF, the HCLM-sched description can be found in Figure 1. To handle the versatility of the application, two parameters, called N and M, are used. The N parameter corresponds to the number of channels of FIRs. The M parameter corresponds to the number of FIRs in each channel. Following the PiSDF semantic, as M parameter is inside the hierarchical actor FIR_Chan, it can be different for each channel of FIR. To represent the HCLM-sched application, many control actors have been added:

- $config$: Configuration actor defining the parameters of the whole graph. The N parameter is set and a list of corresponding M values is sent to $MFilter$.
- $MFilter$: This actor is used to filter the M values to only N values. This will lead to the N executions of the FIR_Chan hierarchical actor.
- Src and Snk: These actors are used respectively to retrieve raw data and to send results.
- $setM$: A basic configure actor used to set the M value of the current chain of FIRs.
- $Switch$: This actor is used to select the input data of the FIR actor. Depending of the $select$ input, it chooses data from interface or feedbacked data.
- $Init$: This actor sets $select$ values. It will make the $Switch$ actor choose input data from the interface on the first iteration and feedbacked data on other iterations.
- $Broadcast(Br)$: This actor duplicates data for the output interface and for the feedback edge.

This implementation of the HCLM-sched algorithm exploits the Round Buffer (RB) behavior of interfaces in PiSDF MoC. As explained in [2], to maintain schedulability of the upper graph without regarding a lower graph, input and output interfaces behave like RBs. If too much data is sent into an output interface, only the last one will be returned to the upper graph. If not enough data is produced by an input interface, data will be duplicated. However, repetition number in one graph iteration for each actor is computed in

such a way that all data from an input interface is consumed at least once and at least enough data are produced to output interfaces. In the HCLM-sched graph, the RB behavior of the output interface of the lower graph is used to keep only the result of the last execution of the FIR actor.

For these experiments, and to be compliant with the default implementation of the OpenMP framework for this platform, both cached and uncached shared L2 memory have been used by the SPiDER runtime to allocate data FIFOs.

5.2 Results

We have proceeded with two experiments in our comparison between SPiDER runtime and OpenMP.

For the first experiment (Exp. 1), we fixed $M = 12$ for all stages. 512-tap FIR filters of $NbS = 4000$ samples have been implemented using the dsplib library of Texas Instruments. Latencies for each iteration have been measured for N varying from 6 to 17 and displayed in Figure 4.

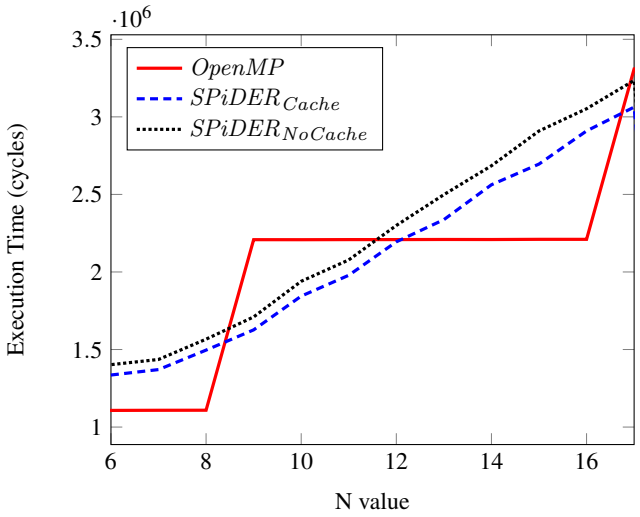

Figure 4: Exp. 1: Latency vs N

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. The execution Gantt chart for $N = 9$ can be found at Figure 5. In the Gantt charts, FIR of same level on all channels have the same color.

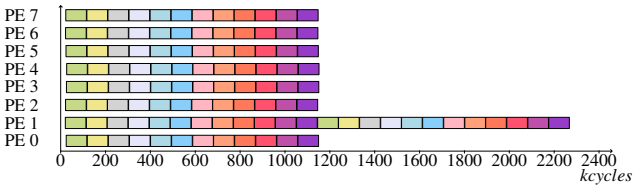

Figure 5: OpenMP Gantt chart for Exp. 1 ($N = 9$)

With the SPiDER runtime, the graph transformation and scheduling phases introduce a visible overhead. This overhead can be seen in Figure 6 where GRT first red tasks represent scheduling overhead. However, the transformation to srSDF extracts more parallelism than OpenMP from the subdivision of channels into multiple FIRs. These choices make SPiDER runtime suitable for applications that do not fit to the architecture topology. In the HCLM-sched benchmark with 9 channels, the overall latency is reduced of up to 26%.

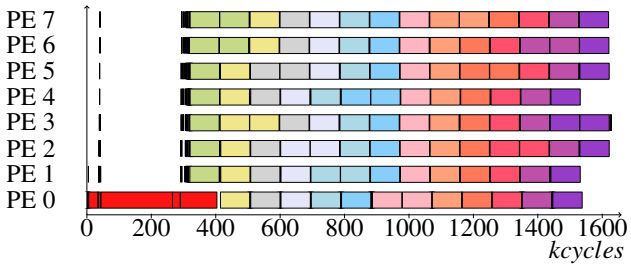

Figure 6: SPiDER Gantt chart for Exp. 1 ($N = 9$)

The second experiment (Exp. 2) is based on multiple iterations of the HCLM-sched benchmark. As the first experiment, 512-tap FIRs of 4000 samples have been used. We fix the number of channels $N = 8$ and the number of FIRs pipelined in each channel varies with : $M = 8 - Chan\_Id$. Then, multiple iterations of the application are launched with a fixed period of 700 kcycles.

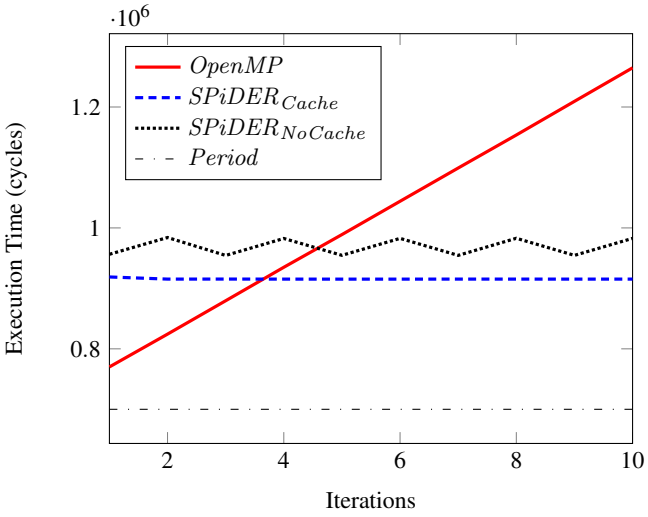

Figure 7: Exp. 2: Latency vs iterations

As we can see in Figure 7, if the latency of the OpenMP implementation is superior to the period, the latency is growing at each new iteration. This is due to the global synchronization at the end of each OpenMP parallel blocks which occurs on the Gantt chart at Figure 8 around 800 kcycles.

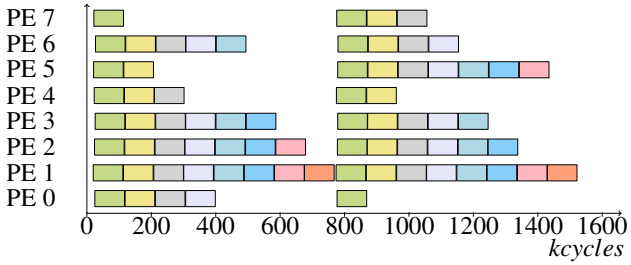

Figure 8: OpenMP Gantt chart for Exp. 2

For the SPiDER runtime, the latency remains constant over iterations. By having prior knowledge on how the application will behave, the GRT can start an execution on LRTs which have already finished the previous execution. It can then start the following iteration as soon as the next period tick occurs, see Figure 9. With a knowledge of the application execution, the SPiDER Runtime can pipeline iterations.

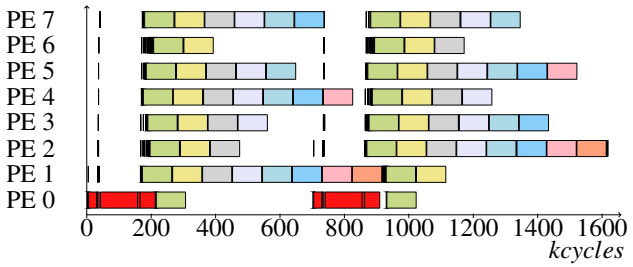

Figure 9: SPiDER Gantt chart for Exp. 2

6. CONCLUSION AND FUTURE WORKS

This paper presents a novel multicore RTOS called SPiDER runtime. SPiDER runtime exploits parallelism from a PiSDF dataflow graph for a multicore execution. 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 SPiDER runtime provides more parallelism to the execution than the OpenMP framework. Results have shown that the SPiDER Runtime reduces the execution latency by up to 26% and allows handling multiple executions.

REFERENCES

[1] Dagum, L., Menon, R.: Openmp: an industry standard api for shared-memory programming. Computational Science & Engineering, IEEE **5**(1), 46–55 (1998)

[2] Desnos, K., Pelcat, M., Nezan, J.F., Bhattacharyya, S.S., Aridhi, S.: 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, pp. 41–48. IEEE (2013)

[3] Gorin, J., Yviquel, H., Prêteux, F., Raulet, M.: Just-in-time adaptive decoder engine: a universal video decoder based on mpeg rvc. In: Proceedings of the 19th ACM international conference on Multimedia, pp. 711–714. ACM (2011)

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

[5] Lucarz, C., Mattavelli, M., Wipliez, M., Roquier, G., Raulet, M., Janneck, J.W., Miller, I.D., Parlour, D.B., 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, pp. 1–8 (2008)

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

[7] Neuendorffer, S., Lee, E.: Hierarchical reconfiguration of dataflow models. In: Formal Methods and Models for Co-Design, 2004. MEMOCODE'04. Proceedings. Second ACM and IEEE International Conference on, p. 179188. IEEE (2004)

[8] Nollet, V., Verkestt, D.: A quick safari through the mpsoc runtime management jungle. In: Embedded Systems for Real-Time Multimedia, 2007. ESTIMedia 2007. IEEE/ACM/IFIP Workshop on, pp. 41–46. IEEE (2007)

[9] Oliva, Y., Pelcat, M., Nezan, J.F., Prevotet, J.C., Aridhi, S.: Building a rtos for mpsoc dataflow programming. In: System on Chip (SoC), 2011 International Symposium on, pp. 143–146. IEEE (2011)

[10] Pelcat, M., Nezan, J.F., Aridhi, S.: Adaptive multicore scheduling for the LTE uplink. In: NASA/ESA Conference on Adaptive Hardware and Systems, pp. 36–43 (2010). DOI 10.1109/AHS.2010.5546233

[11] Sen, M., Corretjer, I., Haim, F., Saha, S., Schlessman, J., Lv, T., Bhattacharyya, S.S., Wolf, W.: Dataflow-based mapping of computer vision algorithms onto fpgas. EURASIP Journal on Embedded Systems **2007**(1), 29–29 (2007)

[12] Singh, A., Shafique, M., Kumar, A., Henkel, J.: Mapping on multi/many-core systems: survey of current and emerging trends. In: Proceedings of the 50th Annual Design Automation Conference, p. 1. ACM (2013)

[13] Stone, J.E., Gohara, D., Shi, G.: Opencl: A parallel programming standard for heterogeneous computing systems. Computing in science & engineering **12**(3), 66 (2010)

[14] Texas Instruments: KeyStone Architecture Multicore Navigator. URL http://www.ti.com/lit/pdf/sprugr9. (accessed 06/2014)

[15] Zaki, G.F., Plishker, W., Bhattacharyya, S.S., Clancy, C., Kuykendall, J.: Integration of dataflow-based heterogeneous multiprocessor scheduling techniques in gnu radio. Journal of Signal Processing Systems **70**(2), 177–191 (2013)