Demonstrating the SPIDER Runtime for Reconfigurable Dataflow Graphs Execution onto a DMA-based Manycore Processor

Hugo Miomandre[1], Julien Hascoët[1,2], Karol Desnos[1], Kevin Martin[3],
Benoît Dupont de Dinechin[2], Jean-François Nezan[1]
[1] IETR, INSA Rennes, CNRS UMR 6164, UBL, Rennes, France
[2] Kalray, Montbonnot-Saint-Martin, France
[3] Lab-STICC, Université Bretagne-Sud, CNRS UMR 6285, Lorient, France
email: first.last@{insa-rennes.fr, kalray.eu, univ-ubs.fr}

***Abstract*—Embedded manycore architectures offer energy-efficient super-computing capabilities but are notoriously difficult to program with traditional parallel programming Application Programming Interfaces (APIs). To address this challenge, dataflow Models of Computation (MoCs) are increasingly used as their high-level of abstraction eases the automation of computation mapping, memory allocation, and communication management. Reconfigurable dataflow is a class of dataflow MoC that fosters a unique trade-off between application dynamicity and predictability. This demonstration presents the first embedded runtime manager enabling the execution of reconfigurable dataflow graphs on a Non-Uniform Memory Access (NUMA) architecture. The proposed runtime dynamically deploys reconfigurable dataflow graphs onto clustered Processing Elements (PEs) through the Networks-on-Chips (NoCs) of the manycore architecture. An open-source implementation on the Kalray MPPA® processor demonstrates the feasibility and the great potential of such a runtime.**

I. INTRODUCTION

THE ever increasing performance of embedded systems is driven by the introduction of more and more parallel architectures. Following this trend, the era of manycore architectures, that massively embed several processor cores, has now begun. Unlike classic multicore architectures, which integrate tens of complex high-performance Processing Elements (PEs) in a single chip, the idea behind manycore architectures is to sacrifice the processing capability of individual PEs, in order to gain silicon area to integrate more PEs. Hence, manycore architectures integrating several hundreds of PEs have been commercialized, offering competitive performance for embedded systems, especially in terms of energy efficiency [1].

At the same time, more than 80% of embedded software is still written using procedural languages such as C/C++. Procedural language are based on control-dependent sequences of instructions manipulating a pool of shared variables. These characteristics make procedural languages inherently ill-suited for programming manycore architectures, where hundreds of PE communicate through complex on-chip interconnects and distributed Non-Uniform Memory Access (NUMA) architectures. Hence, there is a widening *software productivity gap* between the developer productivity, and the increasing complexity of code that must be written to fully exploit the computing power offered by latest embedded processors.

This work is supported by the GdR ISIS MORDRED project.

Dataflow Models of Computation (MoCs) have been introduced in an effort to bridge this software productivity gap. An application specified with a dataflow graph consists of a set of processing entities, named actors, connected by a set of First-In First-Out queues (FIFOs) transmitting data quanta, named data-tokens, between actors. An actor starts its preemption-free execution (it fires) when its input FIFOs contain enough data-tokens. The number of data-tokens consumed and produced during the execution of an actor is specified by a set of firing rules. The dataflow semantics naturally captures parallel and data-driven computations, which makes dataflow MoCs naturally suitable for programming modern parallel architectures.

This demonstration focuses on the reconfigurable class of dataflow MoCs, like the Parameterized and Interfaced Synchronous Dataflow (PiSDF) MoC [2], which allows firing rules of actors to be reconfigured non-deterministically at restricted points in the application execution. The software component responsible for managing the reconfigurations of the graph is called an embedded runtime. The embedded runtime is notably responsible for adapting the mapping of the actors computations and the allocation of the data FIFOs on the computing resources of the target architecture when a reconfiguration of firing rules occurs.

This demonstrator shows the first implementations of an embedded runtime for reconfigurable dataflow graphs, namely the SPIDER runtime [3], on an MPPA®.

II. CONTEXT

A. MPPA® Processor Architecture

The Kalray Multi-Purpose Processor Array (MPPA)® manycore architecture is designed to achieve high energy efficiency and deterministic response times for compute-intensive embedded applications. Figure 1 shows the hierar-

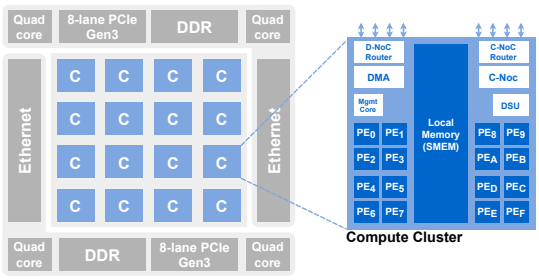


Fig. 1: MPPA® Processor

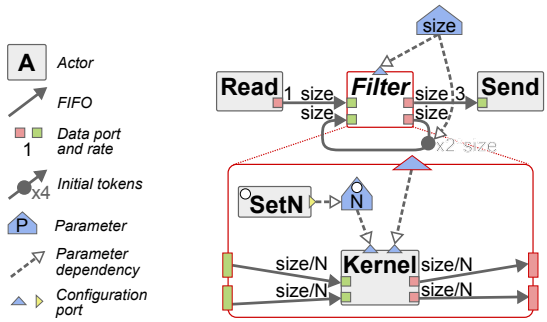
chical computing resources of MPPA featuring the VLIW-core (5-issue), the compute cluster which is equipped with 2 MB of *scratchpad-only-memory* and the manycore processor which integrates the Network-on-Chip (NoC) inter-connnecting the 18 NUMA nodes (16 compute clusters and 2 IOs). The MPPA is a DMA-based manycore architecture. Any computations are driven by DMA data transfers over the NoC and the software runtime is in charge of configuring the DMA NoC interface. Indeed the compute clusters do not have access to the main memory (DDR) and to other compute clusters' memory making the MPPA a challenge to be programmed as all communications are managed by the software.

B. PiSDF

This demonstrator relies on the Parameterized and Interfaced Synchronous Dataflow (PiSDF) [2] MoC. The semantics of the PiSDF MoC is depicted in Figure 2a. In addition to dataflow actors, FIFOs, and data ports, the PiSDF MoC defines a hierarchy mechanism, explicit parameters, parametrization dependencies, and configuration ports as elements of the model semantics. The hierarchy mechanisms makes it possible to specify the internal behavior of an actor with a PiSDF subgraph, instead of source code (generally C/C++). A parameter is a node of the PiSDF graph associated to an integer value. Integer production and consumption rates of actors can be specified with expressions depending on parameters connected to the actor through parameterization dependencies and configuration ports. The output of an actor is able to modify at runtime the input parameter of another actor; therefore, enabling dynamicity if needed by the application (remains static if not used). The dynamicity on an input parameter of an actor will influence the graph transformation, the parallelism, the memory allocation and the scheduling at runtime. Such a challenge is resolved by the SPIDER runtime at the beginning of each iteration of a (sub)graph, if a reconfiguration of a parameter value was triggered. A graph iteration designates a sequence of actor firings during which all actors of the (sub)graph are executed at least once, and the final number of data tokens in each FIFO is equal to the initial state.

C. SPIDER

The Synchronous Parameterized and Interfaced Dataflow Embedded Runtime (SPIDER) runtime exhibits a master/slave organization which eases its porting on a manycore architecture. Each PE of the target architecture is managed by a slave Local Runtime (LRT) process with a dedicated job queue. As



(a) PiSDF semantics (b) PiSDF graph example

Fig. 2: The PiSDF MoC.

all of the in-chip scratchpad-memories of MPPA is limited and managed at runtime by the software, the memory footprint of each LRT must be kept as low as possible. Indeed for efficient graph executions, the local scratchpad-memory of the node should always have enough free space to accommodate the data tokens and the code of actors executed by the associated LRTs. The master Global Runtime (GRT) or host process, which handles graph reconfigurations, mapping and scheduling heuristics, and application profiling, requires a substantially larger memory footprint than LRTs to handle these tasks. Thus this master process is mapped onto the IO subsystem (host processor) of MPPA which has direct access to the external DDR memory. Our SPIDER implementation onto MPPA was done using a multi-core execution model with a master core. The SPIDER communication runtime relies on the Kalray specific low-level one-sided communication API providing the required mechanisms (RDMA, Queues and Remote Atomics) to build complex parallel systems onto the MPPA.

III. DEMONSTRATOR

The demonstrator shows the first running implementation of SPIDER onto the MPPA. The proof of concept shows a self-adaptive implementation of an image filtering application based on a Sobel filter and two morphological operators. At runtime, the application reconfigures itself by changing the number of slices the processed images are split into before being processed in parallel. By monitoring its own execution time, the application is able to select the configuration achieving the highest throughput, depending on the video resolution and the number of available PEs.

REFERENCES

[1] B. D. de Dinechin, R. Ayrignac, P.-E. Beaucamps, P. Couvert, B. Ganne, P. G. de Massas, F. Jacquet, S. Jones, N. M. Chaisemartin, F. Riss *et al.*, "A clustered manycore processor architecture for embedded and accelerated applications," in *High Performance Extreme Computing Conference (HPEC), 2013 IEEE*. IEEE, 2013, pp. 1–6.

[2] K. Desnos, M. Pelcat, J.-F. Nezan, 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.

[3] J. Heulot, M. Pelcat, K. Desnos, J. F. Nezan, and S. Aridhi, "Spider: A synchronous parameterized and interfaced dataflow-based rtos for multicore dsps," in *Education and Research Conference (EDERC), 2014 6th European Embedded Design in*, Sept 2014, pp. 167–171.