A Hybrid Execution Approach to Improve the Performance of Dataflow Applications

Mattis Hasler, Robert Wittig, Emil Matúš and Gerhard Fettweis
Vodafone Chair for Mobile Communication Systems & Center for Advancing Electronics Dresden (cfAED),
Technische Universität Dresden, Germany
Email: {mattis.hasler,robert.wittig,emil.matus,gerhard.fettweis}@tu-dresden.de

***Abstract*—Exploiting the full computation power of a modern Multi-Processor System is a challenging problem. The traditional multithreading programming paradigm assumes a coherent view of multiple (hardware) threads to a shared memory, which is expensive in terms of hardware complexity and memory access and is not compatible with distributed systems like tiled MPSoCs. Application models like SDF deal with this problem by isolating each thread into their own memory space and define communication channels between them. We propose an SDF execution framework that is designed with tiled and embedded systems in mind. Our system is able to execute an SDF like application in an overhead minimizing process network. Part of the graph can be unrolled to a directed acyclic graph that has more overhead but allows for more parallelism exploitation. We show that in some topological situation the benefit of this hybrid execution is bigger than the drawback by the extended overhead.**

I. Introduction

Modern computing systems tend to have more and more parallel processors. The traditional model of computation (MoC) with one or more control flows running in parallel with a coherent view on a single shared memory becomes less attractive for the programming on multiprocessor system on chips (MPSoCs). Instead, the tiled structure of MPSoCs [1]–[5] demand for MoCs that embrace the concept of distributed memory and computation, the isolation of processing units and data distributing network on chips [6], [7]. At the same time, an execution environment must be able to utilize a multicore shared memory (sub)-system since even in embedded systems the tiles are becoming more powerful and often consist of multiple processors sharing one memory [1], [3].

With this work, we propose an execution environment for SDF graph like applications. It is designed with a tiled system architecture in mind while it is not limited to such. Our SDF runner implementation combines a Khan Process Network like process network (PN) [8] based execution strategy with a task graph (TG) [8] approach to exploiting the benefits of both ideas in a hybrid execution application. We will evaluate the system with randomly generated SDF graphs and apply our methodology to baseband signal processing of an LTE-A base station receiver.

The two execution strategies have their strength and weaknesses. The PN approach encounters a low management overhead since once created the network works on its own. On the other side, the TG approach has to constantly create and destroy tasks and data containers which leads to a constant overhead workload. We will explore the boundaries that limit the PNs performance because of the sequential nature of each actor. Further, we will explain how the TG approach breaks exactly this limitation by introducing parallelism into the execution of one actor and hence can increase performance. Whether the performance increase outweighs the additional overhead costs for the TG management decides if the task graph approach or the PN approach is faster. The combination into a hybrid execution strategy allows us to minimize the overhead costs while maximizing parallelism and performance.

This work was partly supported by the German Research Foundation (DFG) within the Cluster of Excellence "Center for Advancing Electronics Dresden" (cfaed). This work has further received funding from the ECSEL Joint Undertaking under grant agreement No. 692519 (PRIME).

II. Runtime Environment

The execution environment developed for this work is able to describe and process graphs of both PN and TG structure. To be able to do that the graphs are composed of three basic building blocks. The "execution unit" (EU) is similar to an operating system's process or thread. It describes the instantiation of a control flow to a processing unit using a binary and defines communication capabilities in the form of IO ports connected to instances of the communication structures. These can either be "channels" or "blobs", both defining a data transfer between two EUs, with one EU producing and the other consuming data. Where a channel realizes a connection between two EUs running at the same time a blob implicitly defines a dependency of execution order, assuring the serial execution of the two. The channel is therefore suited for PN networks where all processes are running and communicating in parallel while blobs fit the concept of TG that depend on a dependency mechanism to define execution order.

III. SDF execution

A SDF graph like shown in Fig. 1 consists of actors that hold the functional description of the algorithm to be executed. An actor can fire, which causes the function to be executed once while consuming and producing predefined numbers of tokens on the actors in- and output ports. Prequesite for a firing is that at all input ports enough token are available. The ports are connected to channels that can hold and transport token from one actor to another.

For the PN execution strategy the SDF actors and channels are mapped one-to-one to PN processes and channels. To implement the TG execution model the SDF actos and channels are split up into single firings and the token sets that a firing

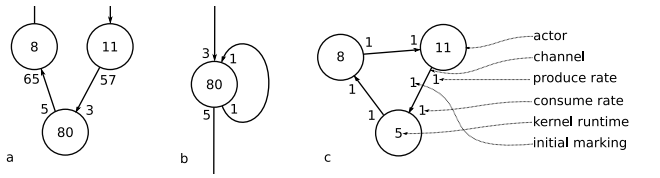

Fig. 1: Example of bottleneck situations in SDF. Left: actor with a potential to introduce parallelism in task mode. Middle: stateful actor depending on itself unable to parallelize execution. Right: feedback loop forcing sequential execution of all three actors.

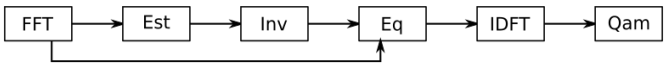

Fig. 2: Block diagram of the LTE application

accesses. The firing and token sets are then mapped to EUs and blobs. Endlessly creating new EUs from new firings at one end of the graph while finishing EUs on the oder side creates a sliding window over an infinite task graph. The hybrid strategy, that is proposed in this work, combines both strategies by running most of the SDF actors in PN mode while single actors are selected to run in TG mode. Wether the selection of an actor for task mode will cause an speedup depends on the graphs topology.

To be able to produce an performance increase an actor must fullfill two conditions. (1) The single iterations must be independent from each other. In Fig. 1.b the actors firings each depend on the prior firing, which prevents parallel executions of any of the actors firings. (2) The runtime of the actors firing must be big enough to compensate for the additional overhead costs from creating the task and the blob. The relatively high kernel runtime is a hint that the bottom actor in Fig. 1.a is a good candidate to fullfills this condition.

IV. Evaluation

First, we test the proposed strategy with randomly generated SDF graphs that are assured to be deadlock-free by the used generator [9]. In the second part, we showcase the method on an LTE-A uplink SC-FDMA baseband receiver for channel bandwidth range from 1.4 to 20MHz. We used a system with 2x 10-Core Intel Xeon E5-2650 v3 @2.3GHz allowing 20 simultaneous threads.

Fig. 3 shows that, even for randomly generated graphs, the hybrid execution strategy is almost always faster than the PN

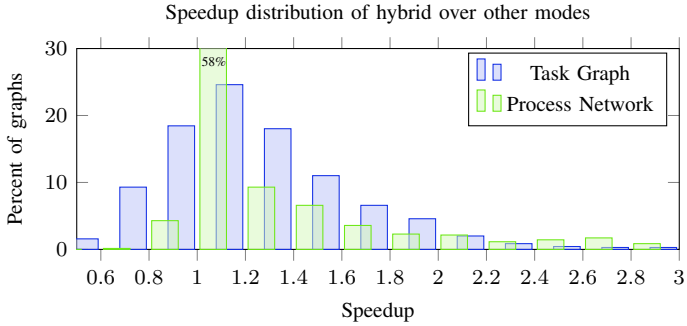

Fig. 3: Speedup distribution of hybrid execution mode over alternative execution modes. Randomly generated SDF graphs of sizes between four and 37 actors. No coherence of graph size and speedup could be observed.

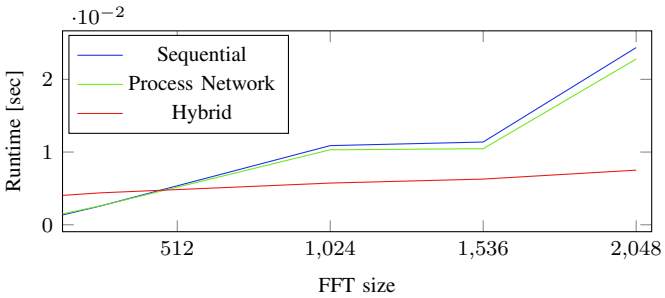

Fig. 4: LTE application runtime measure with different FFT sizes and execution modes.

and the TG strategy. Although most graphs see a speedup of around 1.2 there are cases where the speedup reaches 3.0. Being a signal processing application it is not surprising that the LTE application has an SDF graph like structure (Fig. 2). From a simple serial implementation without any framework, we learn that around 75% of the computation effort is spent in the FFT actor, which is executed serially under the PN strategy. Since the iterations of the FFT block are independent of each other the FFT actor is a good candidate for task graph execution. We ran the LTE application as a sequential application without framework as well as in PN mode and in hybrid mode with the FFT actor selected for TG mode. Since the profitability of the selective TG mode depends on the ratio of overhead and payload computation, it depends on the FFT size. In Fig. 4 the results of our test with different FFT sizes are shown. For our setup at an FFT size of 512 points, the TG mode starts to pay off. The break-even point may be different for other systems and is expected to be lower in an embedded system that may have lesser overhead cost (i.e. in lack of a full Linux stack).

References

[1] J. Ceng, "A methodology for efficient multiprocessor system-on-chip software development," Ph.D. dissertation, 2011.

[2] P. S. Paolucci, A. A. Jerraya, R. Leupers, L. Thiele, and P. Vicini, "Shapes:: a tiled scalable software hardware architecture platform for embedded systems," in *Proceedings of the 4th international conference on Hardware/software codesign and system synthesis*. ACM, 2006, pp. 167–172.

[3] N. P. Carter, A. Agrawal, S. Borkar, R. Cledat, H. David, D. Dunning, J. Fryman, I. Ganev, R. A. Golliver, R. Knauerhase *et al.*, "Runnemede: An architecture for ubiquitous high-performance computing," in *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on*. IEEE, 2013, pp. 198–209.

[4] M. Gschwind, "The cell broadband engine: exploiting multiple levels of parallelism in a chip multiprocessor," *International Journal of Parallel Programming*, vol. 35, no. 3, pp. 233–262, 2007.

[5] S. Haas, T. Seifert, B. Nöthen, S. Scholze, S. Höppner, A. Dixius, E. P. Adeva, T. Augustin, F. Pauls, S. Moriam *et al.*, "A heterogeneous sdr mpsoc in 28 nm cmos for low-latency wireless applications," in *Proceedings of the 54th Annual Design Automation Conference 2017*. ACM, 2017, p. 47.

[6] L. Benini and G. De Micheli, "Networks on chips: A new soc paradigm," *Computer-IEEE Computer Society-*, vol. 35, no. EPFL-ARTICLE-165542, pp. 70–78, 2002.

[7] D. Wingard, "Micronetwork-based integration for socs," in *Design Automation Conference, 2001. Proceedings*. IEEE, 2001, pp. 673–677.

[8] S. S. Bhattacharyya, E. F. Deprettere, and B. D. Theelen, "Dynamic dataflow graphs," in *Handbook of Signal Processing Systems*. Springer, 2013, pp. 905–944.

[9] B. Bodin, Y. Lesparre, J.-M. Delosme, and A. Munier-Kordon, "Fast and efficient dataflow graph generation," *Proceedings of the 17th International Workshop on Software and Compilers for Embedded Systems, SCOPES 2014*, 06 2014.