Toward Efficient Many-core Scheduling of Partial Expansion Graphs

Hai Nam Tran, Shuvra Bhattacharyya, Jean-Pierre Talpin, Thierry Gautier

Toward Efficient Many-core Scheduling of Partial Expansion Graphs

Hai Nam Tran
hai-nam.tran@inria.fr
INRIA
Rennes, France

Shuvra S. Bhattacharyya
ssb@umd.edu
University of Maryland, USA and
Tampere University of Technology, Finland

Jean-Pierre Talpin
jean-pierre.talpin@inria.fr
INRIA
Rennes, France

Thierry Gautier
thierry.gautier@inria.fr
INRIA
Rennes, France

ABSTRACT

Transformation of synchronous data flow graphs (SDF) into equivalent homogeneous SDF representations has been extensively applied as a pre-processing stage when mapping signal processing algorithms onto parallel platforms. While this transformation helps fully expose task and data parallelism, it also presents several limitations such as an exponential increase in the number of actors and excessive communication overhead. Partial expansion graphs were introduced to address these limitations for multi-core platforms. However, existing solutions are not well-suited to achieve efficient scheduling on many-core architectures. In this article, we develop a new approach that employs cyclo-static data flow techniques to provide a simple but efficient method of coordinating the data production and consumption in the expanded graphs. We demonstrate the advantage of our approach through experiments on real application models.

ACM Reference Format:
Hai Nam Tran, Shuvra S. Bhattacharyya, Jean-Pierre Talpin, and Thierry Gautier. 2018. Toward Efficient Many-core Scheduling of Partial Expansion Graphs. In *SCOPES '18: 21st International Workshop on Software and Compilers for Embedded Systems, May 28–30, 2018, Sankt Goar, Germany*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3207719.3207734

1 INTRODUCTION

The data flow programming paradigm is commonly used to model signal processing or control applications in embedded system design. Its simplicity allows implementing code generation techniques to limit the problematic and error-prone tasks of programming real-time parallel applications. Among data flow models of computation, synchronous data flow (SDF) [6] is one of the most popular in the community. This model is deterministic regarding communications: the topology and the amount of data sent and received are known (fixed). Communications between processing elements are isolated through well identified FIFO channels. Task parallelism [4] in SDF can be exploited systematically: tasks can be mapped to available processing elements (cores), dependencies can be enforced, and independent tasks can run in parallel.

Transformation of synchronous data flow graphs (SDF) into their equivalent homogeneous SDF (HSDF) forms has been applied extensively as a preprocessing stage when mapping signal processing algorithms onto parallel platforms. While HSDF expansion helps fully expose both *task* and *data* parallelism [4], it also presents several limitations. First, there is no polynomial bound for the size of the produced graph in terms of the size of the original graph. In particular, highly multirate SDF graphs can lead to exponential increases in size [8]. In some cases, this expansion may not be beneficial because it may overwhelm the number of available processors. Besides, the increase in data parallelism has the hazard of increasing buffering requirements and introducing excessive communication overhead.

Partial expansion graphs (PEG) [4, 5, 10] are proposed to address these limitations. This approach allows the designer or design tool to control the degree to which each actor is expanded rather than expanding actors based on their SDF repetition counts. It presents an interesting design space exploration problem with the objective of finding an optimal mapping from actors into expansion rates.

Problem statement: Existing PEG solutions are not well adapted to achieve efficient scheduling on many-core architectures. They involve the use of shared buffers [10] or the addition of split/join actors [5]. The first solution requires the implementation of a buffer manager to coordinate data production and consumption of expanded actors on shared buffers. It can introduce additional communication overhead and result in a bottleneck (communication through the buffer manager) that gets worse as the number of processors increases. The second solution increases the number of actors and the buffer size requirement significantly because of split/join actors. Besides, one has to account for the overhead introduced by adding these actors.

Contribution: In this article, we present an approach that employs cyclo-static data flow techniques to coordinate the production and consumption of expanded actors. Our PEG solution involves a transformation of the original SDF graph to a cyclo-static data flow (CSDF [2]) graph. The advantage of our solution is that it neither

SCOPES '18, May 28–30, 2018, Sankt Goar, Germany

ACM ISBN 978-1-4503-5780-7/18/05…$15.00
https://doi.org/10.1145/3207719.3207734

requires the use of buffer managers nor split/join actors. Experiment results have shown that our approach can have a significantly lower buffer space and number of added actors when compared to the split/join solution.

2 BACKGROUND AND RELATED WORKS

A synchronous data flow graph (SDF) is a directed graph $G = (V, E)$ consisting of a finite set of actors $V = \{v_1, ..., v_N\}$ and a finite set of edges (or channels) connecting them: E. A channel $e_{ab} = (v_a, v_b, p, q) \in E$ connects the producer v_a to the consumer v_b. The production and consumption rates of e_{ab} are given by the integers $p \in \mathbb{N}$ and $q \in \mathbb{N}$, respectively.

In the form of SDF that we consider in this paper, an actor v_i has a worst-case execution time (WCET) of C_i representing the workload of one firing. An essential property of SDF graphs is that every time an actor fires, it consumes a fixed number of tokens from incoming channels and produces another fixed number of tokens on its outgoing channels. An actor can only fire when there are sufficient tokens to consume and buffer spaces to produce.

In cyclo-static data flow graphs (CSDF), the production (resp. consumption) rate is defined by an infinite periodic sequence $p^\omega = (v)$ in which v is a finite sequence of integers. The term "()" indicates that v is repeated infinitely. The number of tokens produced (consumed) by the j^{th} firing of an actor is given by the value at position $[j \bmod |v|]$ of the sequence ($|v|$ denotes the length of v) .

The *buffer size* (δ_{ab}) of a channel is in principle unbounded, i.e., it can contain arbitrarily many tokens. However, in practice storage space must be bounded. There can be a number of initial tokens that induces an offset in the execution of the consumer to the producer [1]. In this article, we study only *delayless* SDF graphs — that is, graphs in which the number of initial tokens on all channels is 0.

We consider a model of many-core architecture that consists of several compute clusters such as the commercial platform Kalray MPPA-256 [3]. In this Kalray platform, each cluster consists of 16 homogeneous processing cores. Each core has access to a private cache memory and to a limited shared memory of 2 MB. Actors on the same cluster can communicate rapidly via shared memory while tasks that run on different clusters communicate via more expensive means such as Network-on-Chip. Ideally, SDF applications should have a low buffer size requirement so that they can fit in the local shared memory.

Tasks are instantiations of actors when we schedule SDF graphs on many-core architectures. One assumption is that tasks are statically mapped to a processing core, and that task migration between cores is not allowed. This context is considered as non-auto concurrency: two firings of an actor cannot execute in parallel. Non-auto concurrency limits the exploitation of data parallelism. However, allowing auto-concurrency would lead to a complex scheduling problem because of contention on memory accesses and task migration costs.

The PEG approach has been introduced as a solution to exploit data parallelism in the context of non-auto concurrency. Expanded actors can be mapped to tasks that have their own software code blocks. Gordon et al. [4] present a heuristic to adjust the granularity of an SDF graph by fusing actors with small loads, and then expanding the fused regions based on data parallelism. Then, software pipelining is implemented to balance the load on different processors. Kudlur and Mahlke [5] describe an integrated expanding and partitioning approach based on integer linear programming that expands actors as needed with the objective of maximizing the total workload on processors. Both solutions include the addition of split/join actors. They require adding channels to coordinate expanded actors and split/join actors, which, in general, leads to increased buffer size requirements.

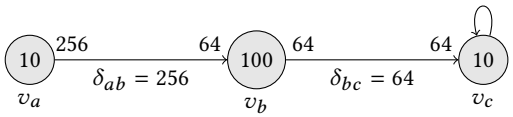

Figure 1: An SDF of 3 actors: v_a and v_b are stateless actors, v_c is a stateful actor.

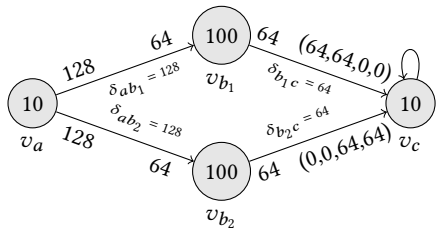

Figure 2: PEG solution.

The two methods above target applications where offline scheduling is performed and the execution time of the actors can be accurately modeled. In the dynamic scheduling context, Zaki et al. [10] present a PEG solution in which each data parallel actor is expanded to a number of instances that is bounded by the number of cores in the platform. Instead of introducing split/join actors as the previous work, *buffer managers* are implemented to coordinate actor firings.

3 APPROACH

The approach we propose employs CSDF techniques to provide a method for coordinating data production and consumption in expanded graphs. We exploit periodic sequences in order to allow each actor firing to consume from or produce onto different channels. Section 3.1 presents an example to motivate our approach and illustrate the application of cyclo-static rates. Then, we formulate our solution in section 3.2.

3.1 Motivational example

Let us start by considering the scheduling of an SDF graph on two processing cores (P_1 and P_2). Figure 1 illustrates the graph of the example, buffer sizes of channels and WCETs, production rates and consumption rates of actors. The graph consists of two stateless actors (v_a, v_b) and one stateful actor (v_c). Figure 3a illustrates the only valid schedule for a single iteration of the graph given the assumption on non-auto concurrency. We can see that this schedule cannot take advantage of two processing cores.

We partially expand the original SDF graph as illustrated in Figure 2. The following observations can be made regarding our solution. Actor v_a is the source actor thus it cannot be expanded. We assume that source actors represent data acquisitions with fixed frequencies. Actor v_b can be expanded with the expansion rate of

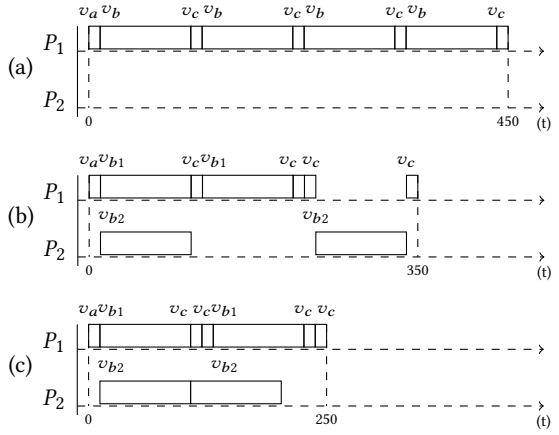

Figure 3: (a) Schedule of the SDF with makespan = 450. (b) Schedule of the PEG with makespan=350. (c) Schedule of the PEG with $\delta_{b_2c} = 128$ with makespan=250.

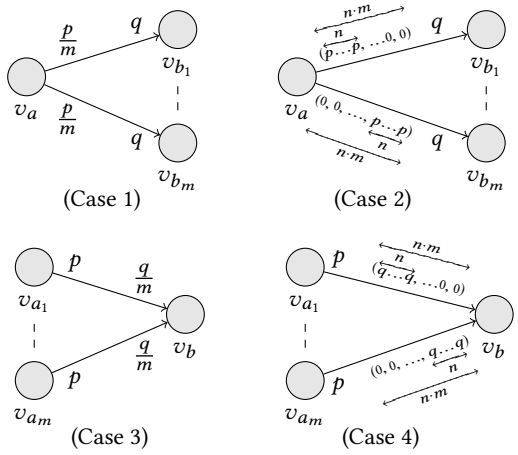

Figure 4: Four cases of PEG.

two to take advantage of 2 cores. A higher expansion rate leads to poor trade-off concerning buffer size and performance gain. Actor v_c cannot be expanded because it is a stateful actor. In other words, a firing of v_c must be completed before the next firing.

Tokens produced by actor v_a are split into two channels. Actor v_a produces the first half of the original data to the top channel (e_{ab_1}) then the second half to the bottom channel (e_{ab_2}). For every firing of v_a, actor v_c must consume data produced by two firings of v_{b_1} before consuming any data from firings of v_{b_2}.

We model this type of scheduling constraint using cyclo-static data flow rates, as illustrated in Figure 3. Our PEG solution effectively transforms the original SDF graph into a CSDF graph.

The makespan is shortened by 100 units as we can see in Figure 3b. In this Figure, we also observe that the second firing of v_{b_2} must wait for the third firing of v_c because the buffer for channel e_{b_2c} is full. If we double the buffer size of channel: $\delta_{b_2c} = 128$, we can have a shorter makespan as illustrated in Figure 3c. It allows the second firing of v_{b_2} to be fired immediately after the first firing.

In this example, we have shown that the application of cyclo-static rates can help us achieve data flow control without the need to add split/join actors or buffer managers. In the next section, we formulate rules for expanding actors in our proposed PEG construction approach, and we show how to derive the resulting cyclo-static production/consumption rates.

3.2 Expansion formulation

This section focuses on PEG solution for the simplest acyclic delayless SDF graph made of a single edge $G = v_a \xrightarrow{p \quad q} v_b$. We present constraints for expansion rates and provide exact formulas for expanding either the producer (v_a) or the consumer (v_b). We show that the application of cyclo-static data flow techniques, even for such simple graphs, is quite involved. Later in the section, we discuss how these techniques can be applied to general SDF graphs.

We apply the following rules to our PEG solution. First, actors in expanded graphs must produce and consume the same number of tokens per firing as the original actors that they are expanded from. For instance, in the previous example, the number of tokens produced and consumed per firing of all actors is not changed. Second, the order of data consumed by stateful actors in the expanded graphs is identical to the order in the original SDF graph. In addition, we assume that there is a predefined ordering of accesses across an actor's ports. In our illustrations, an actor produces and consumes tokens from its incident channels in a top-to-bottom order.

Our expansion techniques are illustrated in Figure 4. We study firstly the expansion of actor v_b while keeping actor v_a intact (unexpanded). The expansion depends on the relation between the production rate p and the consumption rate q. There are two cases:

Case 1: $p > q, p = q \cdot n, n \geq 2$. We expand actor v_b by an expansion rate m such that $m \mid n$ (n is divisible by m). By the rule of transitivity: $m \mid n, n \mid p \implies m \mid p$. We can divide p tokens produced by v_a to m expanded actors of v_b equally. In the expanded graph, one firing of v_a still produces p tokens in total, and one firing of expanded v_b consumes q tokens. In this case, we exploit data parallelism by allowing data produced by v_a to be consumed faster.

Case 2: $p \leq q, q = p \cdot n, n \in \mathbb{N}^*$. We expand actor v_b by an expansion rate $m \in \mathbb{N}^*$. Here, we apply cyclo-static rates to v_a in the expanded graph For each expanded actor of v_b, v_a produces $q = p \cdot n$ tokens after n firings. The length of the periodic sequence is $m \cdot n$ as corresponding to m expanded actors v_b. In this case, the application of PEG can be an advantage if C_b (the WCET of v_b) is significantly larger than C_a. The reason is that it allows data to be produced and consumed without waiting for the completion of actor v_b in the original SDF graph.

Next, we study the expansion of actor v_a while keeping actor v_b intact. We have two cases that are the duals of cases 1 and 2.

Case 3: $p < q, q = p \cdot n, n \geq 2$. This case is the dual of Case 1. We expand actor v_a by an expansion rate m such that $m \mid n$.

Case 4: $p \geq q, p = q \cdot n, n \in \mathbb{N}^*$. This case is the dual of Case 2. We expand actor v_a by an expansion rate $m \in \mathbb{N}^*$ and apply cyclo-static rates to v_b.

In all cases, we limit the expansion rate m to be no greater than number of processors available to schedule the SDF graph Our observation is that expanding an actor beyond the number of processors does not provide better exploitation of parallelism.

	SDF	PEG Split/Join		PEG CSDF	
	NB (#)	NB (#)	BS (byte)	NB (#)	BS (byte)
CFAR	4	9	1792	7	1296
dcal	84	128	14144	110	7632
des	423	591	57772	523	27144
FFT2	26	111	53760	79	24576
matmult	23	69	131488	49	67968

Table 1: Comparison between PEG solutions.

Cases 1, 2 are applied to the expansion of the consumer while cases 3, 4 are applied to the producer. In practice, for general SDF graphs, an actor can be both a producer and a consumer with respect to other actors As a result, we need to apply the combinations of the four cases. There are only four combinations (1-3), (1-4), (2-3) and (2-4). For each combination, the expansion rate must satisfy the conditions of both cases.

A limitation of our solution is that, if $p \nmid q$ or $q \nmid p$, we cannot divide tokens equally to expanded actors. As a result, we can neither expand v_a or v_b. This limitation also applies to the PEG solution with split/join actors. A possible solution would be to associate an actor with a vectorization factor [7] to get divisible rates.

If the order of data consumed by actors does not need to be enforced, we can apply the expansion formulas presented in the four cases. Otherwise, we need to take into account the top-to-bottom ordering and adjust the rates accordingly. The exact formulas are not presented in this article due to space limit. The intuition is to employ cyclo-static rates to delay the production or consumption of tokens on specific channels.

4 EVALUATION

We have conducted experiments with a subset of signal processing applications taken from the STR2RTS benchmark [9], which is a refactored version of the StreamIT benchmark. We evaluate our PEG solution by two metrics: number of actors (NB) and buffer size requirement (BS). The PEG solution with split/join actors is generated by the StreamIt compiler. Then, we implement our PEG solution which allows us to effectively remove split/join actors. A small maximum expansion rate of 4 was chosen to allow fast prototyping.

The results of our experiments are shown in Table 1. The first column shows the number of actors in the original SDF graphs. We also give the number of actors and buffer size requirements of expanded graphs produced by the split/join solution and ours.

The results show that our PEG solution has a smaller number of actors and buffer size requirements compared to the split/join solution. On average, our PEG approach results in 21% fewer actors and requires 41% less buffer cost. Comparing to the original SDF graph, the addition of split/join actors can be quite costly. One of our observations is that the increase in the number of actors also depends on the implementation of the original SDF graphs For example, the SDF graphs of "des" and "dcal" in the benchmark are partly parallelized with the existence of split/join actors. As a result, there is not a significant increase in the number of actors with these applications compared to "FFT2" or "matmult".

We also evaluate the overhead of our PEG solution by manually implementing the expanded graphs of "cfar" and "FFT2". We use the WCET analyzer Heptane [5] to derive a bound on the execution time of one iteration of the graph on one processing core. For each of the two benchmarks, we derive this bound both with and without the use of cyclo-static data flow control, as defined by our expansion approach. From this experiment, we find that the overhead due to the implementation of cyclo-static rates is less than 5%. Indeed, manual implementation of expanded graphs is time-consuming and only applicable to small applications. We plan to integrate the proposed PEG solution in a code generator for SDF graphs.

5 CONCLUSIONS & FUTURE WORK

In this article, we present an approach that employs cyclo-static data flow techniques in partial expansion graphs. Our approach requires neither buffer managers nor split/join actors to coordinate the production and consumption of actors in expanded graphs. Experimental results have shown that our approach has a lower buffer space requirement and a lower number of actors when compared to the split/join solution. In the context of clustered many-core architectures, a key advantage of our PEG solution is that it requires less memory, scheduler and communication overhead because of a smaller number of actors. In addition, the lower buffer space requirement implies that it is easier to fit the expanded graphs in a local shared memory of a compute cluster.

For future works, we are developing an algorithm that allows us to optimizing the assignment of expansion rates to actors. In addition, we plan to integrate the proposed approach in a code generator with the aim of automatically generating expanded graphs from SDF graphs. Furthermore, we intend to study the problem of scheduling PEGs on many-core architectures by integrating our approach in a scheduling framework.

REFERENCES

[1] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee. 2012. *Software synthesis from dataflow graphs*. Vol. 360. Springer Science & Business Media.

[2] G. Bilsen, M. Engels, R. Lauwereins, and J. Peperstraete. 1996. Cycle-static dataflow. *IEEE Transactions on signal processing* 44, 2 (1996), 397–408.

[3] B. D. De Dinechin, D. Van Amstel, M. Poulhiès, and G. Lager. 2014. Time-critical computing on a single-chip massively parallel processor. In *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*. IEEE, 1–6.

[4] M. I. Gordon, W. Thies, and S. Amarasinghe. 2006. Exploiting coarse-grained task, data, and pipeline parallelism in stream programs. *ACM SIGOPS Operating Systems Review* 40, 5 (2006), 151–162.

[5] M. Kudlur and S. Mahlke. 2008. Orchestrating the execution of stream programs on multicore platforms. In *ACM SIGPLAN Notices*, Vol. 43. ACM, 114–124.

[6] E. A. Lee and D. G. Messerschmitt. 1987. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Transactions on computers* 100, 1 (1987), 24–35.

[7] S. Lin, J. Wu, and S. S. Bhattacharyya. 2018. Memory-constrained Vectorization and Scheduling of Dataflow Graphs for Hybrid CPU-GPU Platforms. *ACM Transactions on Embedded Computing Systems* 17, 2 (January 2018), 50:1–50:25.

[8] J. L. Pino, S. S. Bhattacharyya, and E. A. Lee. 1995. A Hierarchical multiprocessor Scheduling System for DSP Applications. In *Proceedings of the IEEE Asilomar Conference on Signals, Systems, and Computers*. 122–126 vol.1.

[9] B. Rouxel and I. Puaut. 2017. STR2RTS: Refactored StreamIT benchmarks into statically analysable parallel benchmarks for WCET estimation & real-time scheduling. In *OASIcs-OpenAccess Series in Informatics*, Vol. 57. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[10] G. F. Zaki, W. Plishker, S. S. Bhattacharyya, and F. Fruth. 2017. Implementation, scheduling, and adaptation of partial expansion graphs on multicore platforms. *Journal of Signal Processing Systems* 87, 1 (2017), 107–125.