Influence of Dataflow Graph Moldable Parameters on Optimization Criteria

Alexandre Honorat, Thomas Bourgoin, Hugo Miomandre, Karol Desnos, Daniel Menard, Jean-François Nezan

Influence of Dataflow Graph Moldable Parameters on Optimization Criteria*

Alexandre Honorat[1][0000−0001−5875−7258], Thomas Bourgoin[2], Hugo Miomandre[2], Karol Desnos[2], Daniel Menard[2], and Jean-François Nezan[2]

[1] Univ. Grenoble Alpes, INRIA, CNRS, Grenoble INP, LIG, 38000 Grenoble, France
{firstname}.{lastname}@inria.fr
[2] Univ Rennes, INSA Rennes, CNRS, IETR - UMR 6164, F-35000 Rennes, France
{firstname}.{lastname}@insa-rennes.fr

Abstract. The integration of static parameters into Synchronous Dataflow (SDF) models enables the customization of an application functional and non-functional behaviours. However, these parameter values are generally set by the developer for a manual Design Space Exploration (DSE). Instead of a single value, moldable parameters accept a set of alternative values, representing all possible configurations of the application. The DSE is responsible for selecting the best parameter values to optimize a set of criteria such as latency, energy, or memory footprint. However, the DSE process explodes in complexity with the number of parameters and their possible values.
In this paper, we study an automated DSE algorithm exploring multiple configurations of a dataflow application. Our experiments show that: 1) Only limited sets of configurations lead to Pareto-optimal solutions in a multi-criteria optimization scenario. 2) How individual parameters impact on optimization criteria are determined accurately from a limited subset of design points. The approach was evaluated on three image processing applications having from hundreds to thousands configurations.

Keywords: Design Space Exploration · Moldable Parameter · SDF

1 Introduction

Designing signal processing applications requires an ever-increasing amount of time and resources, as well as a careful choice of the appropriate target hardware architecture, along with the corresponding software optimizations. On the hardware side, embedded systems are limited by their memory and processing power capabilities, as well as by power consumption and heat dissipation constraints. On the software side, applications for embedded systems are usually written with

* This work was supported by DARK-ERA (ANR-20-CE46-0001-01).
A. Honorat, T. Bourgoin, H. Miomandre: Equal contribution.

procedural languages such as `C`. As `C` is a relatively low-level language, it offers the possibility to maximize the utilization of a given hardware resource through hardware-specific optimization, but at the cost of specialized cumbersome code. This widens the *software productivity gap* between the developer productivity and the code complexity required to fully exploit hardware resources [4, 8].

Dataflow Models of Computation (MoCs) and associated design tools exist to bridge the *software productivity gap*. A piece of software described with a dataflow graph [11] is composed of a set of *actors*, representing computational entities, connected by First-In First-Out queues (FIFOs). A FIFO transports data tokens between *actors* which consume, process and produce said data tokens. An actor is executed when its input FIFOs contain the required number of data tokens. The production and consumption of data tokens for each actor execution is specified by a set of *firing rules*. Thus, dataflow semantics exposes task and data parallelism of data-driven computations. Design tools use the dataflow representation of an application to efficiently handle the allocation of hardware resources when deploying software on specific target hardware architectures, such as Multi-Processor System-on-Chips (MPSoCs).

This paper focuses on statically parameterized dataflow MoCs which allow both functional and non-functional behaviour of the application to be customized at compile time. We propose *moldable parameters*, that we have implemented for the first time within a dataflow MoC. Contrary to regular parameters, which hold a unique value or expression, moldable parameters are associated with a set of alternative values, each resulting in a different configuration of the application. The main contribution of this work lies in the study of how these moldable parameters impact on criteria such as latency, throughput, energy consumption and memory footprint.

In order to study the influence of moldable parameters, we have implemented a Design Space Exploration (DSE) algorithm to find the Pareto-front of multiple application configurations of the same dataflow graph. Considering a multi-objective optimization problem with independent criteria, the Pareto-front is the set of configurations providing the best trade-offs between the optimized criteria. This DSE is completed by a set of scripts used for analyzing and classifying how the different moldable parameters impact on each optimization criterion. The DSE and parameter analysis have been executed on three real-world computer vision applications: Sobel filtering, stereo-matching and Scale-Invariant Feature Transform (SIFT) [12]. Results of these analyses show that: 1) only a limited set of configurations belong to the Pareto-front, and 2) how moldable parameters impact on optimization criteria are determined accurately from a limited subset of configurations. These results lay the ground for the design of low-complexity DSE heuristics responsible for finding automatically the Pareto-efficient configurations of a set of functional and non-functional application parameters.

Related works on parameterized dataflow MoCs and resource allocation for static dataflow MoCs are presented in Section 2. Section 3 presents the concept of moldable parameters. Finally, Section 4 studies how moldable parameters impact on optimization criteria, and Section 5 concludes this paper.

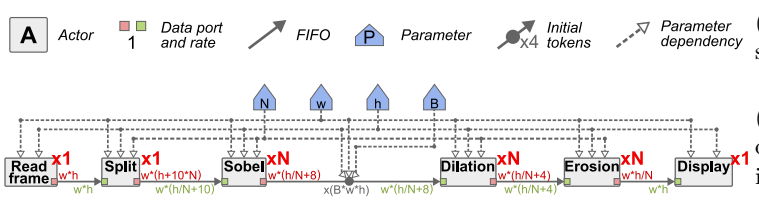
2 Context & Related work

In the semantics of most static dataflow MoCs, the production or consumption rate on each data port of an actor is defined by a fixed integer value [11], or sometimes by a sequence of integer values [2]. In practice, when editing a dataflow graph, it is easier to specify the production and consumption rates of actors by using symbolic expressions made of mathematical operators and functions applied to a list of pre-defined parameters associated to the graph. While the use of such parameterization mechanism is common in many dataflow frameworks, parameters are generally not part of the dataflow MoC semantics, and symbolic expressions are replaced by their resolved values before any analysis or execution of the graph. There exists a few *parametric* dataflow MoCs with a well defined parameterization semantics, as surveyed in [3]. Nevertheless, these parametric MoCs mostly integrate parameters in their semantics to support dynamic reconfiguration of the application graph during its execution. In this paper, we focus on *static* parameters whose values can be resolved statically without executing the application. After describing the model used in the present work, this section presents related work about the resource allocation for static dataflow MoCs.

2.1 Static PiSDF MoC

The parameterized dataflow MoC studied in this paper is the Parameterized and Interfaced Synchronous Dataflow (PiSDF) MoC [7] whose semantics is depicted in Figure 1a. Configuration in the Parameterized and Interfaced Synchronous Dataflow (PiSDF) MoC is based on explicit parameters, which are nodes of the graph associated to a scalar value. The integer production and consumption rates of actors and the number of initial data token in FIFOs, known as *delays*, can be specified with expressions depending on those parameters. If all rates of an actor are evaluated to zero, it is not executed. Moreover, the Parallel and Real-time Embedded Executives Scheduling Method (PREESM) tool [13] implementing PiSDF also supports parameterized expressions of the actor execution times and energy specifications; and parameter values can be used as input argument to actor function calls in its code generation process.

By changing the value of a parameter, it is possible to modify the functional and non-functional behaviour of an application. Non-functional parameters only impact on intrinsic properties of an application such as its degree of



(a) PiSDF semantics

(b) PiSDF graph of an image filtering application

Fig. 1: An example of PiSDF graphical representation.

parallelism and granularity, tuned by data production and consumption rates; and its pipeline depth, tuned by the size of delays. Functional parameters impact on the extrinsic properties of an application, such as its Quality of Service (QoS). For example, a functional parameter can control the resolution of processed images or the bounds of an iterative process, through the modification of data production and consumption rates or number of delays. Functional parameter may also cause the execution of alternative code implementations for actors with different time or energy properties, through time and energy expressions associated to each actor and through actor parameter input. While the semantics of the PiSDF MoC also allows actors to set dynamically the values of graph parameters, in this paper we will focus only on static parameters whose alternative values are known at compile time.

Figure 1b depicts a graph implementing a simple video filtering algorithm, using elements from the PiSDF semantics in Figure 1a. Each iteration of the graph, starting by an execution of the `ReadFrame` actor and ending by an execution of the `Display` actor, corresponds to the processing of a new frame.

Four parameters influence the behaviour of this application:

- $h, w \in \mathbb{N}^*$ are two functional parameters controlling the height and width, respectively, of the images processed by the algorithm.
- $\{N \in \mathbb{N}^* \mid h \mod N = 0\}$ is a non-functional integer parameter controlling the number of slices in which the input image is split before being processed by the N data-parallel firings of the `Sobel`, `Dilation` and `Erosion` actors.
- $B \in \{0, 1\}$ is a non-functional Boolean parameter controlling the presence of delays between the `Sobel` and `Dilation` actors. When enabled, these delays separate the computations of the graphs into two pipeline stages, thus increasing the parallelism and throughput of the application, at the expense of greater memory requirements and latency.

2.2 Resource Allocation for Static Dataflow MoCs

Software synthesis is the process translating a dataflow graph into executable code for a complex computing platform, such as a heterogeneous MPSoC [1]. To do so, software synthesis allocates all the hardware resources needed to support the execution of the dataflow graph. Among other tasks, the software synthesis is in charge of:

- *Scheduling and mapping*, which orders the individual firings of actors, and assigns these firings to the processing element handling them [5];
- *Allocating the memory* needed to store the data produced and consumed by actors near their processing elements [6];
- *Communication routing* which ensures the availability of data and the synchronization of computations [10];
- *Configuring the computing platform* appropriately to optimize the application execution, for example by tuning the Dynamic Voltage and Frequency Scaling (DVFS) configuration of the cores, or by selecting the appropriate scheduling strategy of the supporting operating system [15].

Each one of the aforementioned tasks is a complex, often NP-hard, optimization problem. Indeed, each resource allocation choice made during the software synthesis potentially impacts many optimization criteria such as the latency and throughput of the application, its energy consumption or its memory footprint. Because the resource allocation problem and optimization criteria are deeply entangled, each design decision, or each change in the dataflow graph can have intricate consequences on the different optimization criteria. For example, augmenting the parallelism of a dataflow graph by pipelining it will increase the throughput of the application, at the expanse of larger latency and memory footprint. To make all the resource allocation choices, the DSE process relies on abstract models or on means of hardware simulations for predicting rapidly the optimized criteria depending on the design decisions made. Resource allocation heuristics produce their results in a time ranging from a fraction of a second to hours, depending on their complexity and the desired quality of their outcome.

Most related work on DSE for applications modelled with static dataflow graphs assumes that the dataflow graph is fixed before entering the DSE process, thus evaluating multiple solutions given by the resource allocation solvers. When exploring different application configurations with such a DSE process, the developer must manually modify the application graph, possibly by changing its static parameters, and re-start the whole DSE process for each configuration. Only few works consider exploring design choices on the application model itself, exploiting the dataflow MoC semantics. MASES [16] is one of them; it optimizes the throughput, latency and processor utilization of applications represented with a restriction of Synchronous Dataflow (SDF) where it automatically adds software pipelining. Another tool [14] supports DSE deciding to enable actors or not, for an extended version of SDF with dynamic actors. In our work, the DSE exclusively refers to the domain of parameter configurations: it explores the multiple configurations of an application while considering a single target hardware architecture and a single resource allocation solution to each configuration. Next section introduces the moldable parameters supporting this DSE process.

3 Moldable Parameters

This section introduces *moldable* parameters, which can hold multiple expressions. Once evaluated, those expressions provide the different possible application configurations. To the best of our knowledge, this work represents the first attempt to define and integrate such moldable parameters in a dataflow MoC. After motivating the use of moldable parameters, this section presents their semantics and discusses their influence on DSE and multi-criteria optimization.

3.1 Moldable Parameters Semantics

Parameters in the PiSDF MoC may be used to set various characteristics of the application: data production and consumption rates on FIFOs, delay sizes,

execution times and energy per actor firing, and even actor static integer input argument. Tuning PiSDF *regular* parameters holding a single expression is cumbersome for developers since they have to set the right expression of a parameter manually in order to run the application analysis or code generation for a specific application configuration. Instead, moldable parameters hold multiple alternative expressions so that developers do not have to set them multiple times. Most importantly, moldable parameters make it possible to automatically run analyses on all possible application configurations.

Moldable parameters are a simple extension of parameters as defined in the PiSDF semantics [7]. Each moldable parameter holds a list of symbolic expressions, separated by semi-colons. The first expression is the default one, so that moldable parameters can always be used as regular parameters. Much as regular parameters, symbolic expressions held by moldable parameters can be a mere static integer value or a complex expression depending on other parameters and using mathematical operators and functions.

In this work, we consider only *static* parameters, which means that parameter values never depend on any actor output. Moreover as parameters may depend on other parameters, cyclic dependencies are forbidden, and thus parameters and their dependencies eventually form a tree whose root parameters only hold integer values. A *parameter configuration* of the application dataflow graph is obtained by selecting and evaluating for each moldable parameter a single expression among the list of available ones. When not specified, the word *parameter* refers to both regular and moldable parameters.

3.2 Relation with Multi-Criteria Optimization Problem

In order to select the most suitable parameter configuration, a developer will often consider multiple criteria: throughput, end-to-end latency, memory footprint, energy consumption, or any QoS metric. Evaluating the optimization criteria for each configuration takes from a fraction of a second to hours whereas most configurations are irrelevant, as we shall see in Section 4. Moreover the domain of possible configurations is the Cartesian product of the expressions of moldable parameters, so the size C of the configuration domain to explore explodes with the number of expressions held by moldable parameters. If denoting $\mathcal{P}$ the set of moldable parameters and $|p|$ the number of expressions held by $p \in \mathcal{P}$, then $C = \prod_{p \in \mathcal{P}} |p|$. Hence, there is a critical need for algorithms automating the search for the best configuration, while exploring only a subset of all possible configurations. When a moldable parameter holds only integer values, an option is to sort these values and explore only a representative sample of it.

As moldable parameters may be either functional or non-functional, their influence on the criteria to optimize are not always clear and they might compensate each other even when looking at a single criterion. In a multi-criteria optimization problem, only the points of the Pareto-front are relevant and multiple ones can be considered *optimal*. In the PREESM tool [13] supporting moldable parameters, it is also possible to automatically select a single best configuration if given a priority ordering of the aforementioned criteria. The criteria will be

minimized[3], or forced to stay below a given threshold. However in the context of this paper, the criteria are neither ordered nor weighted, thus there is a priori no single best configuration. Yet the criteria to consider for the Pareto-front should be picked carefully: they should not be entirely dependent on other ones. For example energy and power are not considered together with the throughput since the power is computed by multiplying the energy by the throughput in PREESM. Next section experimentally studies the influence of moldable parameters on all criteria, except QoS ones for practical reasons.

4 Multi-Criteria DSE with Moldable Parameters

This section presents experiments on three use-cases. The influence of moldable parameters on the behaviour of the chosen criteria is first evaluated through an exhaustive DSE; results are then compared with a non-exhaustive DSE.

4.1 Use-Cases: Sobel, stereo and SIFT Applications

Three common computer vision applications have been used for experiments: Sobel, stereo and SIFT[4]. They all contain the following moldable parameters:

- `image_width`: QoS parameter holding 2 possible values for the resolution;
- `AspectRatioDenominator`: QoS parameter holding 2 possible values (but fixed for stereo);
- `parallelismLevel`: non-functional parameter holding 3 values and setting the degree of parallelism of most compute intensive actors;
- `delayRead` and `delayDisplay`: non-functional parameters enabling a pipeline stage after the image read and the result display respectively, it is used only in the corresponding delay size expressions;
- `NumeratorFrequency`: non-functional parameter simulating 12 processor frequencies and used only in expression specifying each actor timing and energy.

The execution times of actors have been measured on a JetsonTX2 board for their default configuration: maximum image size and processor frequency, no pipeline, and minimum degree of parallelism. In this work, the latency is measured as the strictly positive number of graph iterations required to process a bundle of data tokens from end-to-end, that is the full software pipeline depth controlled by the `delayRead` and `delayDisplay` parameters. The Power criterion is the sum of all actor energy specifications weighted by their number of firings, and then multiplied by the throughput. The actor timing expressions are linear to the amount of processed data and inversely proportional to the frequency set by the `NumeratorFrequency` parameter. The actor energy expressions are linear to the amount of processed data and quadratic to the frequency.

[3] For the throughput, its reciprocal is considered so that it can be minimized.

[4] Code is available upon request. For SIFT, see a similar version here: https://github.com/preesm/preesm-apps/tree/master/SIFT

While Sobel contains no other moldable parameter, stereo and SIFT have extra ones to enable some specific actors in the data path or to specify a QoS metric. In particular stereo can be configured with 10 different numbers of disparities used to control the accuracy of the computed depth map. The SIFT application is the most complex one containing 57 actors and 121 FIFOs dispatched into 4 levels of hierarchy and representing between 200 and 550 actor firings for each processed image, depending on the graph configuration. The moldable parameters specific to SIFT are:

- `nKeypointsMaxUser`: QoS parameter holding 9 possible values for the maximum number of keypoints to detect;
- `imgDouble`: QoS parameter enabling one resolution upsampling.

4.2 Raw DSE results

For each configuration, the application is scheduled on an homogeneous architecture with 4 cores. A list scheduling algorithm [9] is used, followed by a static memory allocation [6]. The DSE takes a few seconds to 6 hours to sequentially explore all the configurations within the PREESM framework, respectively for Sobel and SIFT. The Pareto-front is defined for 4 criteria: either Power or Energy, plus Latency, Throughput^{-1} and Memory, respectively denoted PLTS and ELTS. The Pareto-front of SIFT in the domain (Power, Throughput^{-1}, Memory) is represented in Figure 2 for a latency value of 2, that is 2 pipeline stages.

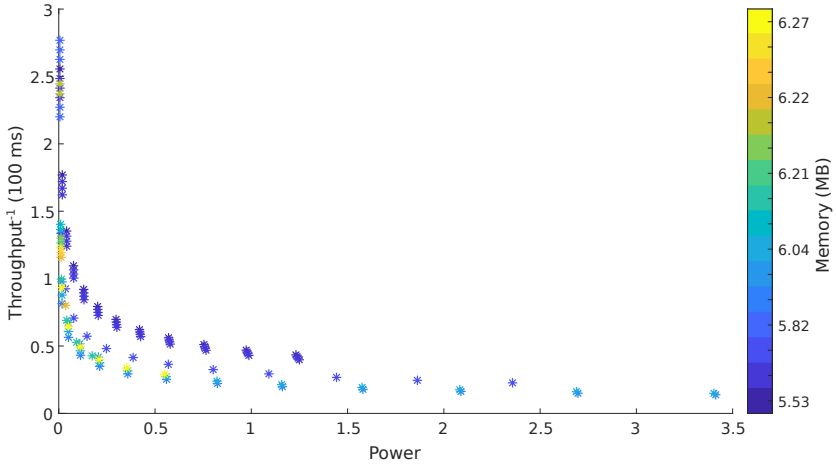


Fig. 2: Pareto-front (Power, Throughput^{-1}, Memory) of SIFT for a latency of 2.

While some purples clusters, especially around the $(0.5, 0.5)$ coordinates, seem to be strictly dominated by blue and yellow clusters below them, they actually boast a smaller memory footprint; the Pareto-front is correct. This slightly smaller memory footprint of purple clusters comes at the cost of worse

Throughput^{-1} and power consumption. Besides, the developer is most probably interested in only one point of each cluster, thus the number of relevant DSE points on the Pareto-front is even more reduced. A manual analysis of the point configurations reveals that all the clusters of points of the same colour are produced by the parameter `nKeypointsMaxUser`. Also, the parameter `NumeratorFrequency` creates the twelve clusters of each colour. Similar results were observed for other latency values and for other applications, but are not presented due to lack of space. The automatic discovery of clusters and of their relation to a specific parameter is an interesting direction for future work.

As the DSE takes up to 6 hours (for SIFT), it is important to know if it really worth it to explore all the configurations. To answer this main question, we answer two corollary ones: **Q1** only the points on the Pareto-front are relevant for the developer, but how many are they among all configurations? **Q2** is it possible to explore only a subset of the configurations to get the influence of each moldable parameter on each criteria?

Q2 will be answered in Section 4.4 by applying the analysis detailed in Section 4.3 on a subset of the expressions held by moldable parameters: if expressions are mere integer, only $Npts$ are retained for the analysis. Regarding Q1, Table 1 gives the total number of configurations and the number of points on the Pareto-front for all applications and different values of $Npts$. While 20% of the points belong to the Pareto-front of the smallest application (Sobel), this percentage goes down to 1% and 2% for stereo and SIFT respectively. Note that the aforementioned percentages are for the PLTS criteria and considering $Npts = all$; they would be even lower for the ELTS criteria.

	Sobel			stereo			SIFT		
$Npts$	ELTS	PLTS	total	ELTS	PLTS	total	ELTS	PLTS	total
all	81	119	576	24	63	5760	91	244	10368
4	–	–	–	–	–	–	31	73	2048
3	19	36	144	8	29	768	23	56	864
2	11	19	64	4	25	128	12	26	384

Table 1: Number of points on the ELTS and PLTS Pareto-fronts and total number of configurations for each application.

4.3 Exhaustive Parameter Analysis

To evaluate the influence of a specific moldable parameter on a given criterion, the variation of this criterion is classified depending on the variation of the moldable parameter, with all other parameters being fixed. We classify moldable parameters in 4 categories depending on their influence on the EPLTS criteria:

- **Same** $\rightarrow$: the criterion is constant as the parameter changes.
- **Increase** $\nearrow$: the criterion strictly increases as parameter value increases.
- **Decrease** $\searrow$: the criterion strictly decreases as parameter value increases.

– **Inconsistent** ⤫: the variation of the criterion is not strictly monotonic while the variation of the parameter value is monotonic.

Each parameter-criterion pair is evaluated on all parameter configurations. If the behaviour of a criterion is consistent for all configurations, the appropriate class is assigned to the parameter-criterion pair. Otherwise, the class is **Inconsistent**. It is implemented as `Matlab` scripts which run in less than a second despite a high complexity: linear to the number of moldable parameters, to the number of criteria and to the number of configurations.

Moldable parameters	Power	Latency	Throughput^{-1}	Memory	Energy
nKeypointsMaxUser	⤫	→	⤫ (↗)	⤫	↗
image_width	⤫	→	↗	↗	↗
parallelismLevel	↗	→	↘	⤫	⤫
AspectRatioDenominator	⤫	→	↗	↗	↗
delayRead	⤫	↗	⤫	⤫	→
delayDisplay	↗	↗	↘	⤫	→
NumeratorFrequency	↗	→	↘	⤫	↗
imgDouble	⤫	→	↗	↗	↗

Table 2: Influence of each moldable parameter on each criterion for exhaustive DSE of SIFT. Circled results are erroneous classifications with $Npts \leqslant 3$.

Results of this evaluation are shown in Table 2 for SIFT. Some criteria are mostly independent of parameters, such as the Latency criterion, which only depends of the `delayRead` and `delayDisplay` moldable parameters. This particular result is expected as only these parameters impact on the pipelining of the application. At the opposite the energy evolves most of the time in the same direction as the parameters, except for the two parameters adding delays, which is also expected. On the other hand, the power and memory criteria exhibit a mostly **Inconsistent** behaviour. So power and memory are not classified the same way and while the power might be more relevant for the developer, its intricate relation with the throughput makes it harder to classify. Results on Sobel and stereo have a similar amount of **Inconsistent** parameter-criterion pairs, also linked to the power and the delays.

The parameter `imgDouble` {0,1} controls the execution of an optional branch of the PiSDF graph. This branch upsamples the input images to find keypoints with a better accuracy. Larger input images imply more data to store and process, hence the mostly negative influence of this parameter on the criteria shown in Table 2. Consequently, as the QoS is not part of the studied criteria, the Pareto-front does not contain any points featuring the parameter `imgDouble` with a value of 1. The same phenomenon is observed for `image_width` and `aspectRatioDenominator`: these parameters worsen the throughput and the memory and energy footprints, so there is no point having the maximum resolution on the Pareto-front. Similar results are obtained for the two other applications, with only a few exceptions about the resolution.

4.4 Faster Parameter Analysis

Here the goal is to reduce the number of configurations required to evaluate the influence of a moldable parameter on a given criterion. We observe that despite less configurations being evaluated, the results are similar to the ones described in Section 4.3. The same set of 4 classes defined in Section 4.3 is used, and the analysis is performed with the same `Matlab` scripts.

Instead of performing an exhaustive evaluation with every configuration, the number of possible expressions that a moldable parameter can be set to is capped arbitrarily according to $Npts$. Doing so, the DSE run time is greatly decreased as the domain of configurations to evaluate, being the Cartesian product of less expressions, decreases from the same ratio. This technique benefits the applications having moldable parameters holding numerous expressions. The class corresponding to each parameter-criterion pair is then evaluated on the subset of possible configurations. As all moldable parameters hold mere integer expressions in our experiments, we arbitrarily select the minimum, median and maximum values if $Npts = 3$ and equally distributed values otherwise. The choice of which expressions to keep in the general case is yet an open question.

Table 2 also shows the results of the evaluation with $Npts \leqslant 3$. For $Npts \leqslant 3$, all results are identical to the classification from exhaustive data, except for the only erroneous parameter-criterion pair displayed within a circle. The number of tested configurations for SIFT is reduced from 10368 to 864 for $Npts \leqslant 3$. For stereo, only 4 errors occur with $Npts = 2$. No error occurs for Sobel.

The classification of a parameter-criterion pair as **Inconsistent** requires an explicit divergent behaviour across multiple configurations. Consequently, limiting the DSE to a smaller sublist of moldable parameter expressions, such as $Npts = 2$, increases the likelihood of parameter-criterion pairs misclassification away from the **Inconsistent** class. Increasing $Npts$ to 4 yields a classification identical to the exhaustive analysis for SIFT, while testing only 1536 configuration out of 10368, representing 15% of the design space and an equivalent analysis speedup of almost 7x. No error occurs for stereo with $Npts = 3$, testing 432 configurations out of 5760.

5 Conclusion

This paper introduces the first use of moldable parameters in the semantics of dataflow MoCs as a way to automatically explore different configurations of an application in a multi-criteria optimization context. Results on three computer vision applications reveal that only 1 to 20% of configurations obtained with this technique belong to the Pareto-front. Finding these Pareto-efficient configurations is crucial for the developer. An analysis on how moldable parameters impact on DSE criteria shows that only a limited subset of all configuration is needed to classify accurately the influence of each parameter. This observation gives credit to the design of smart DSE heuristics capable of finding Pareto-efficient configurations without resorting to an exhaustive analysis.

References

1. Bhattacharyya, S.S., Murthy, P.K., Lee, E.A.: Software synthesis from dataflow graphs, vol. 360. Springer Science & Business Media (1996)
2. Bilsen, G., Engels, M., Lauwereins, R., Peperstraete, J.: Cycle-static dataflow. IEEE Transactions on signal processing **44**(2), 397–408 (1996)
3. Bouakaz, A., Fradet, P., Girault, A.: A survey of parametric dataflow models of computation. ACM Transactions on Design Automation of Electronic Systems (TODAES) **22**(2), 1–25 (2017)
4. Castrillón, J.: Programming heterogeneous MPSoCs : tool flows to close the software productivity gap. Ph.D. thesis, RWTH Aachen University, Aachen (2013), aachen, Techn. Hochsch., Diss., 2013
5. Castrillon, J., Leupers, R., Ascheid, G.: Maps: Mapping concurrent dataflow applications to heterogeneous mpsocs. IEEE Transactions on Industrial Informatics **9**(1), 527–545 (Feb 2013)
6. Desnos, K., Pelcat, M., Nezan, J., Aridhi, S.: Pre- and post-scheduling memory allocation strategies on mpsocs. In: Proceedings of the 2013 Electronic System Level Synthesis Conference (ESLsyn). pp. 1–6 (2013)
7. Desnos, K., Pelcat, M., Nezan, J.F., Bhattacharyya, S., Aridhi, S.: PiMM: Parameterized and interfaced dataflow meta-model for MPSoCs runtime reconfiguration. In: Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS). pp. 41–48. IEEE (2013)
8. Ecker, W., Müller, W., Dömer, R.: Hardware-dependent Software, pp. 1–13. Springer Netherlands, Dordrecht (2009)
9. Honorat, A., Desnos, K., Bhattacharyya, S.S., Nezan, J.F.: Scheduling of Synchronous Dataflow Graphs with Partially Periodic Real-Time Constraints. In: Real-Time Networks and Systems. Paris, France (Jun 2020)
10. Kang, S., Yang, H., Schor, L., Bacivarov, I., Ha, S., Thiele, L.: Multi-objective mapping optimization via problem decomposition for many-core systems. In: 2012 IEEE 10th Symposium on Embedded Systems for Real-time Multimedia. pp. 28–37
11. Lee, E.A., Messerschmitt, D.G.: Synchronous data flow. Proceedings of the IEEE **75**(9), 1235–1245 (1987)
12. Lowe, D.G.: Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision **60**(2), 91–110 (Nov 2004)
13. Pelcat, M., Desnos, K., Heulot, J., Guy, C., Nezan, J., Aridhi, S.: Preesm: A dataflow-based rapid prototyping framework for simplifying multicore dsp programming. In: 2014 6th European Embedded Design in Education and Research Conference (EDERC). pp. 36–40 (2014)
14. Schwarzer, T., Falk, J., Müller, S., Letras, M., Heidorn, C., Wildermann, S., Teich, J.: Compilation of dataflow applications for multi-cores using adaptive multi-objective optimization. ACM Trans. Des. Autom. Electron. Syst. **24**(3), 29:1–29:23
15. Wang, J., Roop, P.S., Girault, A.: Energy and timing aware synchronous programming. In: International Conference on Embedded Software, EMSOFT'16. p. 10. ACM, Pittsburgh, United States (Oct 2016)
16. Yu, W., Kornerup, J., Gerstlauer, A.: Mases: Mobility and slack enhanced scheduling for latency-optimized pipelined dataflow graphs. In: Proceedings of the 21st International Workshop on Software and Compilers for Embedded Systems. pp. 104–109. SCOPES '18, ACM, New York, NY, USA (2018)