DB4HLS: A Database of High-Level Synthesis Design Space Explorations

Lorenzo Ferretti[1], Jihye Kwon[2], Giovanni Ansaloni[3], Giuseppe Di Guglielmo[2], Luca Carloni[2], Laura Pozzi[1]

[1]Università della Svizzera italiana, Lugano, Switzerland, [2]Columbia University, New York, United States
[3]EPFL, Lousanne, Switzerland

Abstract—**High-Level Synthesis (HLS) frameworks allow to easily specify a large number of variants of the same hardware design by only acting on optimization directives. Nonetheless, the hardware synthesis of implementations for all possible combinations of directive values is impractical even for simple designs. Addressing this shortcoming, many HLS Design Space Exploration (DSE) strategies have been proposed to devise directive settings leading to high-quality implementations while limiting the number of synthesis runs. All these works require considerable efforts to validate the proposed strategies and/or to build the knowledge base employed to tune abstract models, as both tasks mandate the syntheses of large collections of implementations. Currently, such data gathering is performed ad-hoc, a) leading to a lack of standardization, hampering comparisons between DSE alternatives, and b) posing a very high burden to researchers willing to develop novel DSE strategies. Against this backdrop, we here introduce DB4HLS, a database of exhaustive HLS explorations comprising more than 100000 design points collected over 4 years of synthesis time. The open structure of DB4HLS allows the incremental integration of new DSEs, which can be easily defined with a dedicated domain-specific language. We think that of our database, available at `https://www.db4hls.inf.usi.ch/`, will be a valuable tool for the research community investigating automated strategies for the optimization of HLS-based hardware designs.**

Index Terms—**High-Level Synthesis, Databases, Machine Learning, Big Data, Design Space Exploration.**

I. INTRODUCTION

High-Level Synthesis (HLS) fostered a revolution in hardware design. HLS frameworks allow the specification of hardware components in languages such as C, C++, or SystemC. As opposed to traditional Register Transfer Level (RTL) approaches, HLS flows do not require detailed descriptions of the logic gates, memory elements and interconnects comprising hardware implementations. Instead, these are automatically generated, based on the high-level specifications and on a set of directive values specifying optimizations such as the unrolling factor of loops and the inlining of functions. By decoupling specification from implementation, HLS allows unprecedented productivity, leading to considerable reductions in non-recurring engineering costs.

Nonetheless, while HLS allows to easily define vast design spaces for a given hardware specification, determining the performance (latency) and resource requirements (area, power) of each implementation still requires time-consuming syntheses. The amount of possible implementations of a design explodes exponentially with the number of applied directives, while, in general, only a few of them are Pareto-optimal from a performance/resources perspective. Exhaustive explorations are therefore wasteful (since only Pareto implementations are of interest) and impractical beyond very simple cases.

Various strategies, which we summarize in Section II, have been proposed to identify (or approximate) the set of Pareto-implementations while minimising the number of synthesis runs [1] [2] [3]. This problem is named *HLS-driven Design Space Exploration (DSE)*. The proposed DSEs strategies are typically validated against exhaustive explorations, which the authors performed ad-hoc. Moreover, works such as [4] [5] rely on prior knowledge to steer the HLS exploration process. Performing the huge number of synthesis required for validation or for generating a high-quality knowledge base entails a very high effort, which at present must be repeated ex-novo when investigating the performance of a novel DSE methodology.

Against this backdrop, we introduce DB4HLS, a database of high-level synthesis design space explorations. The database comprises more than 100000 design points, reporting the synthesis outcomes of exhaustive explorations performed on 39 designs from the MachSuite [6] benchmark suite. In addition, we define a simple domain-specific language to define design spaces, resulting in an open infrastructure that can be enriched by further contributions from the research community.

We believe that, by providing standardized synthesis data sets, our effort will allow easier comparisons among DSE strategies, enabling fairer evaluations of the strengths and weaknesses of each approach. It will also facilitate the development and assessment of future design exploration frameworks, spurring research in this challenging field.

II. RELATED WORKS

State of the art DSE frameworks for HLS follow three main approaches. Black-box methodologies aim, after an initial phase, at iteratively refining explorations by smartly selecting additional design points. To this end, they employ unsupervised learning strategies such as clustering [2], random forest [1], lattice traversing [3] and response surface models [7]. Model-based strategies, on the other hand, estimate performance and resource requirements of implementations by developing an analytical formulation of the effect of directives when applied to a design. Typically, they can well approximate the Pareto set of best-performing implementations with few synthesis, but are restricted in the type of targeted optimizations (e.g., loop unrolling and dataflow in [8]). The authors of

TABLE I: DSEs available in the database. Each entry reports benchmark, function name, and number of configurations ($|CS|$). All functions are from Machsuite [6].

Benchmark	Function name	$|CS|$
spmv ellpack	ellpack	1600
bfs bulk	bulk	2352
md knn	md_kernel	1600
viterbi	viterbi	1152
gemm ncubed	gemm	2744
gemm blocked	bbgemm	1600
fft strided	fft	64
sort merge	ms_mergesort	4096
	merge	4096
stencil stencil2d	stencil	1344
stencil stencil3d	stencil3d	1536
radix sort	update	2400
	hist	4704
	init	484
	sum_scan	1280
	last_step_scan	800
	local_scan	704
	ss_sort	1792
aes	aes_addRoundKey	500
	aes_subBytes	50
	aes_addRoundKey_cpy	625
	aes_shiftRows	20
	aes_mixColumns	18
	aes_expandEncKey	216
	aes256_encrypt_ecb	1944
backprop	get_delta_matrix_weights1	21952
	get_delta_matrix_weights2	31213
	get_delta_matrix_weights3	21952
	get_oracle_activations1	2401
	get_oracle_activations2	1372
	product_with_bias_input_layer	1372
	product_with_bias_second_layer	686
	product_with_bias_output_layer	392
	backprop	2048
	add_bias_to_activations	1372
	soft_max	64
	take_difference	512
	update_weights	1024

all these works adopt as figure of merit either the Hypervolume or the Average Distance from Reference Set (ADRS) for validation, and both require the computation of true Pareto frontiers from exhaustive explorations. Recently, a promising research avenue has focused, instead, on exploiting prior knowledge in order to perform Design Space Exploration in hardware design. These works [4] [5] leverage the availability of a comprehensive knowledge base, such as the one we describe in our paper, to achieve exploration results close to that of model-based strategies while being much more flexible in the number and type of supported directives.

While benchmark suites dedicated to hardware design are available, such as CHStone [9], MachSuite [6], Rosetta [10] and S2CBench [11], they only provide specifications (in the form C/C++ code) as benchmarks. Conversely, our DB4HLS suite offers rich and well-defined design spaces and related synthesis outcomes, greatly easing the burden of performing comparative evaluations of exploration methodologies. To the best of our knowledge, this is the first database of HLS implementation made publicly available with the intent of standardize the evaluation process, and provide a source of knowledge for ML strategies.
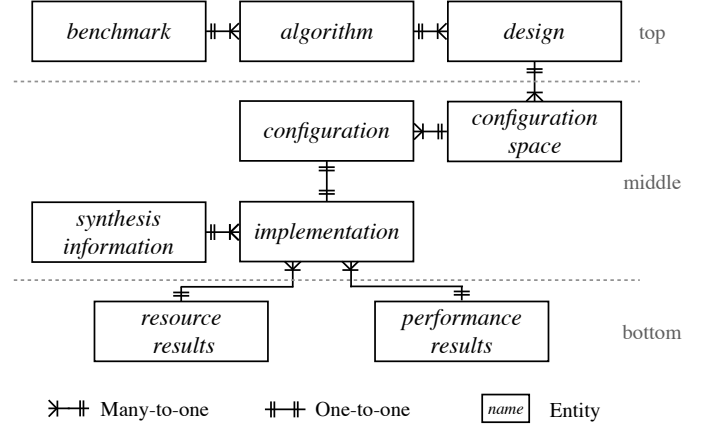

Fig. 1: Simplified scheme of the Entity-Relationship Diagram (ERD) of the DB4HLS syntheses database.

III. AVAILABLE DESIGN SPACE EXPLORATIONS

We provide a rich set of DSEs by targeting the benchmarks of the MachSuite collection of designs [6]. We have performed DSEs for 39 out of 50 functions in the benchmark suite, discarding those having a variable latency due to input-dependent control flows, and those having very small design spaces. The considered functions present on average 40 lines of code, with the biggest having 308 lines of code.

We performed an exhaustive exploration of each design–according to the configuration space defined by the user–running more than 100000 synthesis. Table I lists all the designs explored and their configuration space size.

We used Vivado HLS [12] version 2018.2 to perform the syntheses , and we targeted a ZynqMP Ultrascale+ (`xczu9eg`) FPGA chip, with a target clock of $10ns$.

To restrain the design spaces sizes, we have constrained directive set values with a numerical range (e.g., the unrolling factor) to power-of-two or integer divisor of the maximum admissible values (e.g., number of loop iterations). Moreover, for some designs, different optimizations are forced to have the same values when intuitively such choice would lead to better cost/performance trade-offs (e.g., binding the loop unrolling factor to the array partitioning one).

Even when considering these constraints, the data collection required more than 4 years of single-core machine time. To speed up this process, GNU Parallel was adopted to collect synthesis results from 60 parallel Vivado HLS instances, allowing us to populate the database in approximately 25 days of wall-clock time.

IV. DB4HLS INFRASTRUCTURE

In addition to the DSE data, the DB4HLS framework offers a) a database infrastructure hosting DSE in a structured and easy-to-access way, b) a domain-specific language used to describe a configuration space for a target design, c) an interface to generate new explorations and further enrich the database. The remaining of this section describes these further contributions in details.

A. A database for DSEs

Snippet 1: `last_step_scan` design (C code).

```
1  void last_step_scan(int bucket[SIZE], int sum[RADIX]){
2    int i, j, k;
3    loop_1:for(i = 0; i < RADIX;i++){
4      loop_2:for(j = 0; j < BLOCK; j++) {
5        k = (i * BLOCK) + j;
6        bucket[k] = bucket[k] + sum[i];
7      }
8    }
9  }
```

Snippet 2: Configuration Space of `last_step_scan`.

```
1  resource;last_step_scan;bucket;{RAM_2P_BRAM}
2  resource;last_step_scan;sum;{RAM_2P_BRAM}
3  array_partition;last_step_scan;bucket;1;{cyclic,block
       };{1->512,pow_2}
4  array_partition;last_step_scan;sum;1;{cyclic,block};{1->128,
       pow_2}@bind_a
5  unroll;last_step_scan;last_1;{1->128,pow_2}@bind_a
6  unroll;last_step_scan;last_2;{1,2,4,8,16}
7  clock;{10}
```

Fig. 2: Left: Snippet of the `last_step_scan` C code function from MachSuite [6]. We rewrote the code to increase the readability without affecting its functionality. Right: An associated Configuration Space Descriptor (CSD).

The database structure, implemented in MySQL, comprises a description of the design targeted for exploration (top part of Figure 1), and that of the explored HLS optimizations applied to each design (middle part of Figure 1). Finally, it reports the resource and performance results obtained by synthesis (as described in the bottom part of the figure). Each of these components is described more in detail in the following.

Similarly to the taxonomy adopted in MachSuite [6], applications are identified by the *benchmark* they belong to (e.g.: aes), by the *algorithm* they realize (e.g.: `aes256_encrypt`) and by the *design* implementing such algorithms. As an example, two variants are provided by MachSuite for the `aes256_encrypt` algorithm (one using lookup tables to store encryption keys and one generating the values online), each corresponding to a separate design specified as C++ code.

A descriptor of the HLS optimizations considered for the DSEs are stored as entries in *configuration space* table. Multiple explorations (hence, rows in the configuration space table) for the same design are possible, corresponding to different choices of optimizations, or explorations targeting different tools/FPGAs, or even contributions from different researchers. An entry in the *configuration space* table is linked to many entries of the *configuration* table, where each entry indicates a specific element of the design space.

A line in the *configuration* table (that indicates the set of HLS optimizations defining a design space element) is linked to an entry in the *implementation* table. Furthermore, the *synthesis information* table provides additional information on each performed synthesis: the synthesis timestamp, the contributor that originated the data, the employed synthesis tool and version, and the targeted FPGA. Finally, each implementation links to one or more entries in the *resources* and *performance* tables, which report the synthesis outcomes. Resources are expressed as employed Flip-Flops, Look-Up Tables, Block RAMs (BRAM) and DSP blocks, while performances are reported in terms of effective latency.

B. A domain-specific language for DSEs

Generating the different configurations associated with an DSE is a tedious and error-prone process when performed by hand. We therefore developed a Domain-Specific Language (DSL) to automatically and concisely define configuration spaces by employing Configuration Space Descriptors (CSDs).

Each line of a descriptor encodes a *knob*, which comprises a directive type, a label corresponding to its location in the

design C/C++ code, and one or multiple sets of values. The number of sets is equal to the number of parameters required by the directive type. Values can be numerical when expressing optimizations such as loop unrolling or array partitioning factors, or categorical when determining the type of employed FPGA resources such as BRAM types. A shorthand is provided for expressing regular value series (e.g., a succession of power-of-two values). Finally, we provide a `@bind` decorator, which constraints the values associated with different directives.

Figure 2 shows, for the `last_step_scan` function in Snippet 1, an example of DSL descriptor created to define its configuration space (Snippet 2) created using the DSL. The DSL descriptor defines seven different knobs. Lines 1 and 2 of Snippet 2 show two knobs associating a dual-port BRAM to the input array `bucket`, and `sum` respectively. Lines 3 and 4 define knobs specifying the array_partitioning directive. These directives are created as combinations of partitioning strategies and partitioning factors. Both line 3 and 4 combine two partitioning strategies (`cyclic` and `block`) with the associated directive values set for the partitioning factors–all the powers of two from 1 up to 512 for knob 3, and all the powers of two from 1 up to 128 for knob 4. Then line 5 and 6 define for `loop_1` and `loop_2` the associated set of unrolling factors to consider during the exploration, all the powers of two from 1 up to 128 and 16, respectively. Both line 4 and 5 have a binding decorator (`@bind_a`), that specifies that the array partitioning directive and the unrolling one must have the same partitioning and unrolling factor for all the configurations described by the CSD. Finally line 7 defines the target clock.

The configuration space resulting from a DSL descriptor having N different knobs is the Cartesian product of all knob values: $CS = K_1 \times K_2 \times \cdots \times K_N$; where K_i is the directive values set related to knob i, taking into account the restrictions imposed by the `bind` decorator. In case of directives requiring multiple parameters, the knob K_i is itself the Cartesian product among each set of values associated to the knob. Lastly, the total number of configurations, i.e., the configuration space size, is given by its cardinality ($|CS|$).

The configuration space descriptor of Figure 2 in Snippet 2 describes a configuration space with 1600 different configurations. Without the binding decorator, the cardinality of the configuration space would be 12800.
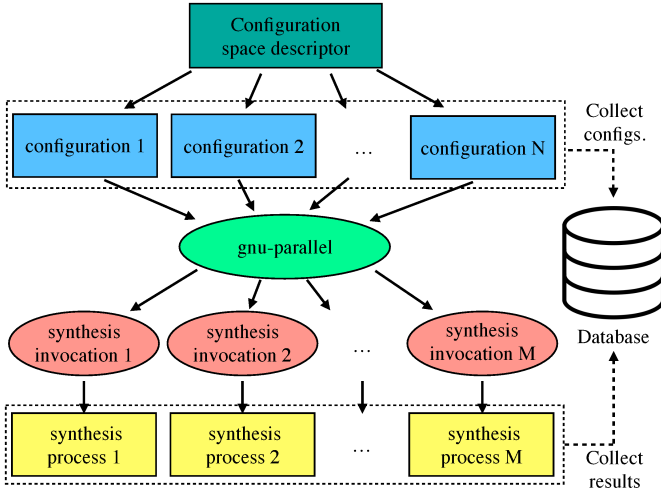
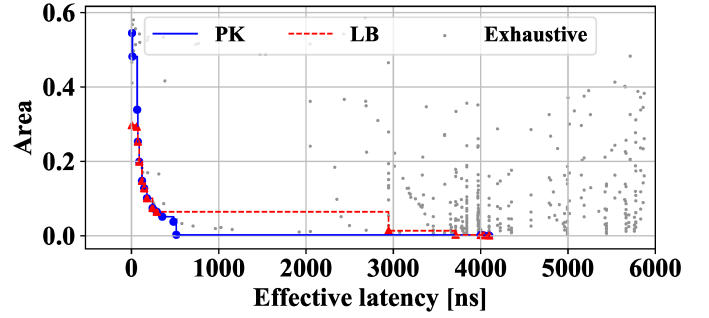Fig. 3: Adding DSE to DB4HLS with parallel processing.

Fig. 4: Example of DSEs comparison employing DB4HLS.

VI. CONCLUSIONS

DB4HLS offers an extensive set of DSEs targeting functions from MachSuite [6]. The data collection is made publicly available and will be will be updated increasing the number of design explorations and targeted benchmarks. In addition, further design spaces can be effectively defined through a novel domain-specific language and a framework to efficiently contribute novel explorations to DB4HLS. Both the DB4HLS database and the framework for DSE generation are publicly available at `https://www.db4hls.inf.usi.ch/`.

C. A framework for parallelizing HLS runs

Figure 3 gives a high-level view of the infrastructure, realized through Bash and Python scripts, which we provide to automate DSE and commit their outcomes in DB4HLS. Starting from a user-provided design and Configuration Space Descriptor (CSD), configuration files are automatically generated and stored in the database. Then, using GNU Parallel [13], a tunable number of instances of an employed HLS tool (we use Vivado HLS for the data collection described in Section III) are concurrently and independently executed, one for each configuration. As synthesis runs terminate, the retrieved performance and resources information are also stored in DB4HLS, and new HLS processes are launched until all configurations have been explored.

MySQL statements can then be used to retrieve data from the tables in the database and to access the design's implementations and the associated performance and resources results.

V. CASE STUDY

Herein, we showcase two possible uses of DB4HLS. We use the database both to compare the results of two DSE methodologies, and as a source of knowledge for one of them. We employed a lattice-based strategy (LB) from [3], and one leveraging prior knowledge (PK) [5], to perform DSEs for the `local_scan` design space available in DB4HLS. Figure 4 reports the Pareto curve obtained by LB and PK for the `local_scan` design space. Grey dots represent the area and latency of the 704 implementations belonging to the `local_scan` design space provided by DB4HLS. The figure also reports the approximated Pareto fronts retrieved by the lattice methodology described in [3] (LB) and by the prior-knowledge strategy in [5] (PK).

In this scenario, DB4HLS is employed to comparatively evaluate the two strategies, without requiring to re-run ex-novo a large number of time-consuming synthesis runs. Besides, for PK, the database mandates the availability of a set of *source* design spaces in order to extract previous knowledge. In fact, DB4HLS can be effectively employed in these cases, or in similar ML-based methods [4], to provide the required knowledge base.

REFERENCES

[1] H.-Y. Liu and L. P. Carloni, "On learning-based methods for design-space exploration with high-level synthesis," in *Proceedings of the 50th Design Automation Conference*, Jun. 2013, pp. 1–6.

[2] L. Ferretti, G. Ansaloni, and L. Pozzi, "Cluster-based heuristic for high level synthesis design space exploration," *IEEE Transactions on Emerging Topics in Computing*, no. 99, pp. 1–9, Jan. 2018.

[3] ——, "Lattice-traversing design space exploration for high level synthesis," in *Proceedings of the International Conference on Computer Design*, Oct. 2018, pp. 210–217.

[4] Z. Wang, J. Chen, and B. C. Schafer, "Efficient and robust high-level synthesis design space exploration through offline micro-kernels pre-characterization," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2020, pp. 145–150.

[5] L. Ferretti, J. Kwon, G. Ansaloni, G. Di Guglielmo, L. Carloni, and L. Pozzi, "Leveraging prior knowledge for effective design-space exploration in high-level synthesis," in *CASES*. IEEE, 2020, pp. 145–150.

[6] B. Reagen, R. Adolf, Y. S. Shao, G.-Y. Wei, and D. Brooks, "Machsuite: Benchmarks for accelerator design and customized architectures," in *Proceedings of the IEEE International Symposium on Workload Characterization*, Oct. 2014, pp. 110–119.

[7] S. Xydis, G. Palermo, V. Zaccaria, and C. Silvano, "SPIRIT: Spectral-Aware Pareto Iterative Refinement Optimization for Supervised High-Level Synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 1, pp. 155–159, Oct. 2015.

[8] G. Zhong, V. Venkataramani, Y. Liang, T. Mitra, and S. Niar, "Design Space Exploration of Multiple Loops on FPGAs Using High Level Synthesis," in *Proceedings of the International Conference on Computer Design*, Dec. 2014, pp. 456–463.

[9] Y. Hara, H. Tomiyama, S. Honda, H. Takada, and K. Ishii, "Chstone: A benchmark program suite for practical c-based high-level synthesis," in *2008 IEEE International Symposium on Circuits and Systems*. IEEE, 2008, pp. 1192–1195.

[10] Y. Zhou, U. Gupta, S. Dai, R. Zhao, N. Srivastava, H. Jin, J. Featherston, Y.-H. Lai, G. Liu, G. A. Velasquez *et al.*, "Rosetta: A realistic high-level synthesis benchmark suite for software programmable fpgas," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2018, pp. 269–278.

[11] B. C. Schafer and A. Mahapatra, "S2cbench: Synthesizable systemc benchmark suite for high-level synthesis," *IEEE Embedded Systems Letters*, vol. 6, no. 3, pp. 53–56, 2014.

[12] "Vivado high-level synthesis." [Online]. Available: https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html

[13] O. Tange, "Gnu parallel - the command-line power tool," February 2011.