Design space exploration of heterogeneous MPSoCs with variable number of hardware accelerators

Siyuan Xu [a], Shuangnan Liu [b], Yidi Liu [b], Anushree Mahapatra [b], Mónica Villaverde [c], Félix Moreno [c], Benjamin Carrion Schafer [a, *]

[a] Department of Electrical and Computer Engineering, The University of Texas at Dallas, USA
[b] Department of Electronic and Information Engineering, The Hong Kong Polytechnic University, HK SAR, China
[c] Centro del Electrónica Industrial, Universidad Politécnica de Madrid, Spain

A B S T R A C T

This work proposes three different methods to automatically characterize heterogeneous MPSoCs com- posed of a variable number of masters (in the form of processors) and hardware accelerators (HWaccs). These hardware accelerators are given as Behavioral IPs (BIPs) mapped as loosely coupled accelerators on a shared bus system (*i.e.* AHB, AXI). BIPs have a distinct advantage over traditional RT-level based IPs given VHDL or Verilog: The ability to generate micro-architectures with different area vs. perfor- mance trade-offs from the same description. This is usually done by specifying different synthesis direc- tives in the form of pragmas. This in turn implies that using different mixes of the accelerators' micro- architectures lead to SoCs with unique area vs. performance trade-offs.

Two of the three methods proposed are based on cycle-accurate simulations of the complete MPSoC, while the third method accelerates this exploration by performing it on a Configurable SoC FPGA. Exten- sive experimental results compare these three methods and highlight their strengths and weaknesses.

Keywords:
Design space exploration
Heterogeneous SoCs
Hardware accelerators
High-level synthesis
In-situ exploration
Simulation acceleration

1. Introduction

VLSI circuits are reaching complexities never seen before. One solution that is being proposed is to customize the computing platforms to the application domain, also known as domain-specific computing. At the chip level, this is accomplished by designing heterogeneous Multiprocessor Systems-on-Chips (MPSoCs) with dedicated hardware accelerators (HWAcc). These dedicated accelerators execute dedicated tasks faster than general purpose architectures by exploiting the inherent parallelism of some application (*e.g.* image processing applications), while consuming a fraction of the power.

Due to the pressure to tape-out these chips at shorter design cycles, companies often rely on third party Intellectual Properties (3PIPs) to meet their tight schedules. Companies have also started to rely on High-Level Synthesis (HLS) to increase their design productivity. Thus, third party behavioral IPs (3PBIPs) are often used as HWAccs on these heterogeneous MPSoCs. The International Technology Roadmap for Semiconductors (ITRS) already suggested in 2013 that by 2020 a 10x productivity increase for designing complex SoCs was needed [1]. Two main factors were predicted to help to achieve this goal. The first is the re-use of components. ITRS estimates that around 90% of the SoCs will be composed of re-used components. Secondly, the use of new design methodologies to raise the level of abstraction *i.e.* HLS. The use of HLS has led to a new market for 3PBIPs. One of the main advantages of BIPs is that they are much easier to re-use than traditional RT-level IPs. Moreover, micro-architectures of different area vs. performance trade-offs can be easily obtained by synthesizing the behavioral description with different synthesis options. This is typically done by setting different synthesis options in the form of pragmas (comments) inserted directly into the source code or through global synthesis options. For example, these options can control how to synthesize arrays (register or RAM), if a function should be inlined or not and if loops should be fully unrolled, partially unrolled, not unrolled or pipelined.

FPGA vendors have also embraced this new paradigm and have released their own Programmable or Configurable SoCs (CSoCs), *e.g.* Altera's Cyclone V SoC and Xilinx's Zynq FPGA. These CSoCs contain multiple embedded cores mainly in the form of ARM Cortex A9 and reconfigurable fabric onto which to map the

* Corresponding author.
E-mail addresses: siyuan.xu@utdallas.edu (S. Xu), shuangnan.liu@connect.polyu.hk (S. Liu), dylan@connect.polyu.hk (Y. Liu), anushree.mahapatra@connect.polyu.hk (A. Mahapatra), monica.villaverde@upm.es (M. Villaverde), felix.moreno@upm.es (F. Moreno), schaferb@utdallas.edu (B. Carrion Schafer).

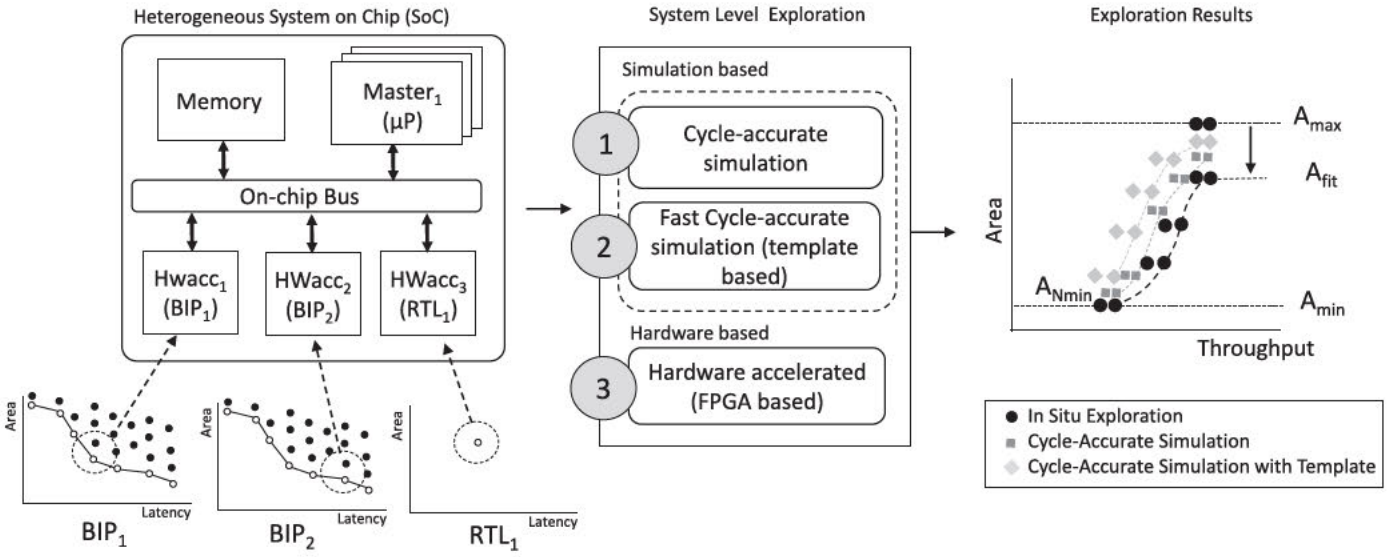

Fig. 1. Shared bus heterogenous MPSoC target platform and overview and the different system-level design space exploration methods proposed in this work.

accelerators, all interconnected through a standard bus e.g. AMBA AHB/AXI. Concurrently, these FPGA vendors have also extended their design flows to include HLS as a vehicle to program their FPGAs and to simplify the creation of these heterogeneous systems using high-level languages (e.g. Xilinx's Vivado HLS and Altera's OpenCL SDK).

It should be noted that HLS is still far away from a push-button solution. Thus, not any software description can be synthesized. The design flow advocated in this work is that hardware designers will now spend time optimizing a behavioral description in any high-level language (e.g. C/C++) to make it synthesizable instead of writing RTL code. The hardware designer can then either manually or automatically perform a HLS Design Space Exploration (DSE) setting different synthesis attributes to obtain a trade-off curve of Pareto-optimal designs. These optimal designs are in turn passed to the system integrator who will then choose one of these micro-architectures for each accelerator for that particular MPSoC. The problem that arises now for the system integrator is to determine which micro-architecture is the best in terms of area, performance and/or power efficiency when mapped onto the SoC. This work addresses this issue and compares three characterization methods previously introduced [2–4], each with different characteristics, to automatically find system configurations with unique area vs. performance trade-offs composed of different mixes of micro-architectures for each hardware accelerator.[1] In particular, this paper makes the following contributions:

- Propose a fully automatic flow to generate system configurations composed of a variable number of hardware accelerators each with a unique micro-architecture with unique area vs. performance trade-offs.
- Introduces three different exploration methods, each with its own strengths and weaknesses. The first is based on a detailed cycle-accurate simulation of the complete systems [2], the second is also simulation based and accelerates the simulation by substituting each accelerator with a template that only mimics the IO timing, but is *empty* inside [3], and the last one further accelerates the exploration by performing it on a configurable SoC FPGA [4].
- Present extensive experimental results comparing the three methods.

2. Motivational example

Fig. 1 shows the target MPSoC platform used throughout this work. It is an MPSoC with variable number of processors and a variable number of slaves mapped as loosely coupled HWAccs on a memory mapped shared bus.[2]

This figure also highlights the benefit of designing the hardware accelerators at the behavioral level instead of traditional RT-level (Verilog or VHDL). A variety of micro-architectures with unique trade-offs can be generated without the need to modify the input description. Out of all the different micro-architecture, we are only interested in the Pareto-optimal, which confirm the dominating trade-off curve. In the RTL case, the micro-architecture is fixed and corresponds to a single design. In this case it is impractical to manually generate all the different design variants. To circumvent this problem, some research work has focused on converting RTL descriptions into behavioral descriptions to maximize HLS DSE [5,6]. As shown in Fig. 1 too, using different mixes of micro-architectures for each accelerator leads to system configurations with unique area vs. performance trade-offs, where in this case the area only indicates the area of the slaves (excluding the masters' area as this does not change). Each point on the diagram represents a unique task mapping within the specified number of masters. In particular, when using for every accelerator the fastest micro-architecture, but also the largest a system with area A_{max} is obtained. At the same time, if the smallest, but slowest micro-architectures for all the accelerators are used, then a system of area A_{min} is created. As shown, different mixes of accelerators lead to different area vs. performance trade-offs. In particular it can be seen that a smaller system (A_{fit}) with the same performance as A_{max} can be found, mainly because of bus contention issues.

In addition to the micro-architecture chosen for each accelerator, the tasks' mapping onto the different masters also leads to systems with different performances, but the same area. This is clearly shown in Fig. 1, by a row of points with same area, but different

[1] In this work a system is a SoC composed of M masters and N accelerators interconnected through a shared bus.

[2] In this work the term master and processor will be used interchangeably to denote a system component which originates the data for the slaves and initiates the communication sequence. This work also makes use of the term slaves, BIP, HW kernel or HW accelerator interchangeably.

Table 1
Exploration result and task mappings on different number of masters (1–3) in motivational example in Fig. 1 with 3 slaves and effect on the area.

M	Mappings	Num	$Area_{orig}$	$Area_{fit}$	Δ[%]
1	{(1,2,3}	1	1321	992	26.1
2	{(1),(2,3)}, {(2),(1,3)}, {(3),(1,2)}	3	1321	1002	25.3
3	{(1),(2),(3)}	1	1321	1048	21.9

throughputs. Table 1 shows the different possible tasks mappings on the motivational example with different number of masters, ranging from 1 to 3, and 3 slaves, and how the area savings that can be achieve by choosing different mixes of micro-architectures as compared to only using the fastest micro-architectures but also largest.

Several observations can be made from these results:

Observation 1: Different task mappings for the same accelerator implementations lead to different system performance, while consuming the same area. Hence, there is a task mapping which dominates the others. This fact is mainly due to the fact that the masters cannot feed the slaves continuously with data due to bus congestion problems. The bus arbitration policy also affects this. In this work the bus arbiter is set in all cases to round robin arbitration as this is the most widely used arbitration policy.

Observation 2: Based on observation 1 it can be further observed that for each BIP, there are smaller designs, which can lead to the same performance of the entire system, while consuming less area than an equivalent system composed of only BIPs of highest performance and largest area. It is therefore not needed to fully parallelize the BIPs to achieve the highest performance. Hence a slower, but smaller version of these BIPs can be used in each of the MPSoC configurations. The smallest micro-architecture for each BIP depends on the number of masters on the MPSoC and on the mapping of tasks on each master.

Observation 3: The number of mappings follows the Stirling numbers of the second kind sequence. In this work we do not consider the task execution order once the tasks are mapped onto the same master. For the first case only a single task mapping is possible, because there is only a single master available ({1, 2, 3}) as shown in Table 1. Similarly, only one task assignment is possible in the case that 3 masters are available as each task is mapped onto its own master ({(1), (2), (3)}). For the case of two masters, 3 task mappings are possible. This will be explained in more detail in the next sections as this impacts the running time of our technique.

The features that enable our work to identify the amount of performance degradation allowed by each accelerator and the ability to generate smaller designs are: First, the use of BIPs for each of the accelerators and second, the ability to generate fast cycle-accurate models for the entire MPSoC to accurately estimate the idle time of each slave and the performance of the entire system. Other works make use of virtual platforms which model the communication part loosely through payloads. The problem with this approach is that the exact idle time of each HW module cannot be accurately measured and hence previous work cannot exactly determine the idle time of each module. This combined with the fact that our method takes as inputs BIPs which can be synthesized into different micro-architectures automatically are key differentiating elements in this work.

This work will make use of these observations in order to find the dominating configurations trade-off curves efficiently. Thus, the problem that this work deals with can be formulated as:

Problem definition: Given N BIPs to be mapped onto a memory mapped shared bus MPSoC as loosely-coupled HWAccs, each with a testbench $BIP_1/TB_1, BIP_2/TB_2, \ldots, BIP_N/TB_N$ firstly explore each BIP_i to obtain a trade-off curve (TDC) of Pareto-optimal micro-architectures for each BIP $TDC(BIP_i) = \{micro_1, micro_2, \ldots, micro_p\}$ with the following area $\{A(micro_1) > A(micro_2), \ldots, > A(micro_p)\}$ and latencies $\{L(micro_1) < L(micro_2), \ldots, < L(micro_p)\}$. Secondly, given M masters find a Pareto-optimal system-level trade-off curve, such that the TDC is composed of unique micro-architectures for each BIP $TDC_i = \{BIP_1(micro_x), BIP_2(micro_y), \ldots, BIP_N(micro(z))\}$.

3. Previous work

On-chip hardware accelerators can be coarsely classified as tightly coupled accelerators and loosely coupled. In the first case, the accelerator is directly attached to a specific core. Some examples of this include [7,8]. In the latter case, the accelerator is directly attached to a global bus and is shared among multiple cores, the authors in [9] developed MorphoSys, a reconfigurable computing system which combines reconfigurable hardware with general-purpose processors for word-level, computation-intensive applications. In [10] the authors proposed a loosely coupled retargetable Loop Accelerator (LA). To ensure that the accelerator design is broad enough to accelerate different applications the authors perform a DSE on the LA. However, the architecture does not fit all situations, requiring the re-design of the LA for different applications. Tabkhi et al. [11] introduced Function-Level Processors (FLPs) to fill the gap between Instruction Level Processors (ILPs) and dedicated HWAccs. FLPs are comprised of configurable Function Blocks (FBs) implementing selected functions which are then interconnected via programmable point-to-point connections constructing an extensible/configurable macro data-path. This work focuses on these second type of accelerators, which can be accessed by different masters in the system, which in [12] was shown to lead to very good results for particular systems (e.g. applications with clear memory access patterns).

Some work on loosely coupled accelerators connect these accelerators through custom interconnects e.g. NoCs combined with DMAs for quick data transfer. In [13] the authors present a global management of NoCs in accelerator-rich architectures. In our case, the accelerator is connected to the system through a standard AMBA AHB/AXI bus.

Regarding MPSoC Design Space Exploration (DSE), much work has been done in the past. We thus, only highlight some representative work. Most previous works can be classified into three different categories: (a) Using aggressive pruning techniques to reduce the search space [14,15] (b) make use of meta-heuristics to search the design space [16,17] or (c) use static analytical techniques to guide the explorer [18,19]. Once the candidate solutions have been generated, these have to be evaluated either through simulation (i.e. [19]) or through predictive models (i.e. [18]). Closer to this work, the authors in [20,21], use compositional techniques to explore the design space of SoCs composed of multiple accelerators generated from HLS which have a set of Pareto-optimal configurations to choose from. The main idea is to use compositional techniques to reduce the number of invocations of the HLS tools as the explore the accelerators' micro-architecture and the system at the same time. The same authors extended this work in [22] to deal with memory optimizations. The authors in [23] presented a flow that allows to generate and explore complete SoCs at the behavioral level by creating a synthesizable library of components and present a prototype of an SoC interconnected through a Networks-on-Chip (NoCs) architecture. The main problem with all this previous work as that they still require to perform a full system-level simulation to measure the performance of each newly generated SoC.

With regard to the use of higher levels of abstraction to create HWAccs more efficiently, Corre et al. [24,25] proposed the use of HLS for HWAccs in heterogeneous MPSoCs, similar to this proposal, but for tightly coupled HWAccs. The authors make use of fast analytical estimators for the area and performance, hence, different

workloads, bus congestion and bus arbitration policies cannot be modeled by their work. Their work assumes fixed regular static access patterns, which is not too realistic. The authors in [26] propose a virtual platform for accelerating the exploration of many-accelerator systems.

Previous work, vary based on the type of simulation abstraction used to model the MPSoC ranging from sequential simulators (e.g. QEMU [27] and SimpleScalar [28]), transaction level models (TLM) (e.g. OVP [29]) to cycle-accurate modeling (e.g. HORNET [30]). In most of this previous work, the main objective is to explore system parameters like e.g. cache sizes, number of processors, bus bitwidth, memory latency.

Similar to our work [31] uses HLS to design the HWaccs in MPSoCs and develop an HLS DSE method to obtain MPSoCs with unique area vs. latencies. In this work a fast static area and latency estimator is used, hence the bus congestion is not taken into account in their work. In [12] the authors use HLS to generate a set of dominating micro-architectures mapped as loosely coupled HWaccs in a SoC, similar to this work. They then propose a system-level exploration method based on a pre-defined system template and emulate these configurations on an FPGA.

In these previous works, the access pattern and workload was always considered regular. The authors in [32] showed that workloads in modern MPSoC-based embedded systems are becoming increasingly dynamic, which can cause changes in the nature of the workload demand over time. They introduce the concept of system scenarios, which group system behaviors that are similar in such a way that the system can be configured to exploit this cost similarity. Quan et al. [33] extended this work by introducing a hybrid task mapping method that combines static mapping exploration and a dynamic mapping optimizer.

A hybrid approach is taken by Renesas Electronics, which commercializes a run-time reconfigurable coarse grain FPGA IP called Stream Transpose Processor (STP) which can be embedded into SoCs to accelerate any datapath [34]. To facilitate the programmability of this IP, the STP is configured using HLS technology [35]. A set of contexts are generated from an untimed C description that are loaded onto the reconfigurable fabric every clock cycle. This also helps saving area as the STP can be configured to execute different tasks.

Finally, with regard to in situ characterization, the authors in [36] propose an in situ characterization to perform DSE for multi-core systems targeting Intel Core i7-2600 with 4 cores and Xilinx Zynq-7000 with two ARM cores. In this previous work, no hardware accelerators are used. Closer to this work, the authors in [37] proposes a method called *hArtes*, which is a toolchain that maps applications specified in high-level languages into platforms composed of general-purpose processor, DSPs and FPGAs. This work makes extensive use of pragmas to control what is executed where.

Our work is different from the above previous works in various aspects. First, we assume that the overall system architecture has already been fixed. This implies that the bus structure, memories, HWAccs have already been fixed. This also implies that the HW/SW partition is fixed. Previous work done in the area of automatic HW/SW partitioning can be fully leveraged to further automate the complete flow [38–40]. Hence, our proposed flow is fully orthogonal to this work. Secondly, we take as inputs BIPs in the form of explorable C/SystemC inputs. This allows us to generated a variety of micro-architectures of unique area vs. performance trade-offs for each BIP. Finally, we propose and compare three methods. Two are simulation based, while the last one is in-situ based, where the exploration runs on a Configurable SoC FPGA. If this hardware platform is also the final target product, then this method also leads to 100% accurate. On another dimension, this work also considers how different micro-architectures of individual accelerator impacts the entire system. Thus, the input to our system explorer is a set of trade-off curves for each slave and the objective of our work is to find unique combinations of micro-architectures of each BIP which lead to Pareto-optimal system configurations, once the overall architecture, including number of masters, bus type, arbitration policy, etc., has already been fixed.

We believe that this work is extremely important for system integrators to help them decide either which micro-architecture to choose from when integrating BIPs in complex SoCs or which method to use. It should be noted that previous work on system-level design could be extended by adding our proposed BIP micro-architecture search method as another search dimension. We thus believe that our work is fully orthogonal to previous work.

4. Proposed system exploration methods

Fig. 2 shows the complete flow diagram of our three different proposed methods, called *Fast Explorer for Behavioral Systems FEBS*. All three methods takes as inputs N behavioral IPs (BIPs) given in synthesizable ANSI-C or SystemC and their testbenches (*TB*), which form a complete System S, $S = \{BIP_1/TB_1, BIP_2/TB_2, \ldots, BIP_N/TB_N\}$. Therefore the HW/SW partition is already decided a priori. The TB will be executed on the SoC as a master and acts as traffic generator, while the *BIPs* are synthesized and mapped as slaves in the system. The output of our method is a set of dominating systems with unique area vs. performance (throughout) and different task mapping combinations. Based on the number of masters in the system, this means that all the tasks can be either mapped onto a single master or each task can have its own master or any combination in-between. The core of the system explorer is the optimization of each slave's micro-architecture for each unique task mapping. The task mapping is important because it decides upon the ultimate workload pattern in the SoC. The next subsection describes this optimization first.

4.1. Single mapping behavioral IPs optimization

The core of our fast exploration method is the optimization of the individual IPs for a specific task mapping. This step is composed of 4 main steps and 1 pre-characterization phase as follow:

Pre-step: BIP design space exploration. As a pre-characterization step, our method starts by performing an HLS DSE for each individual BIP. As mentioned previously, C-based design has the advantage over traditional RTL-based design that micro-architectures with unique area vs. performances (in this case latencies) can be obtained by setting different synthesis options. The pre-characterization step of our flow is based on a genetic algorithm to explore just the synthesis directives of each BIP. The HLS DSE used in this stage is a modified genetic algorithm presented in [41]. Other methods have been proposed in the past using other meta-heuristics [42]. These methods could be easily incorporated into our flow, but are out of the scope of this paper as we only use the dominating designs as an input to our main flow. We will not go into details about how the GA works as it has been presented in previous work. Basically, each *explorable* operation *OP* is represented as a gene to which a synthesis attribute (pragma) p or global option opt is assigned. These *OP* include arrays, functions and loops. The list of all genes built a chromosome Cr, which is then combined and mutated based on pre-defined crossover and mutation probabilities (pc and pm). Each new configuration is synthesized using as many FUs as needed to fully parallelize the generated micro-architecture. The result of this step is a trade-off curve of dominating designs for each BIP with unique micro-architectures, $TDC(BIP_i) = \{micro_1, micro_2, \ldots, micro_p\}$.

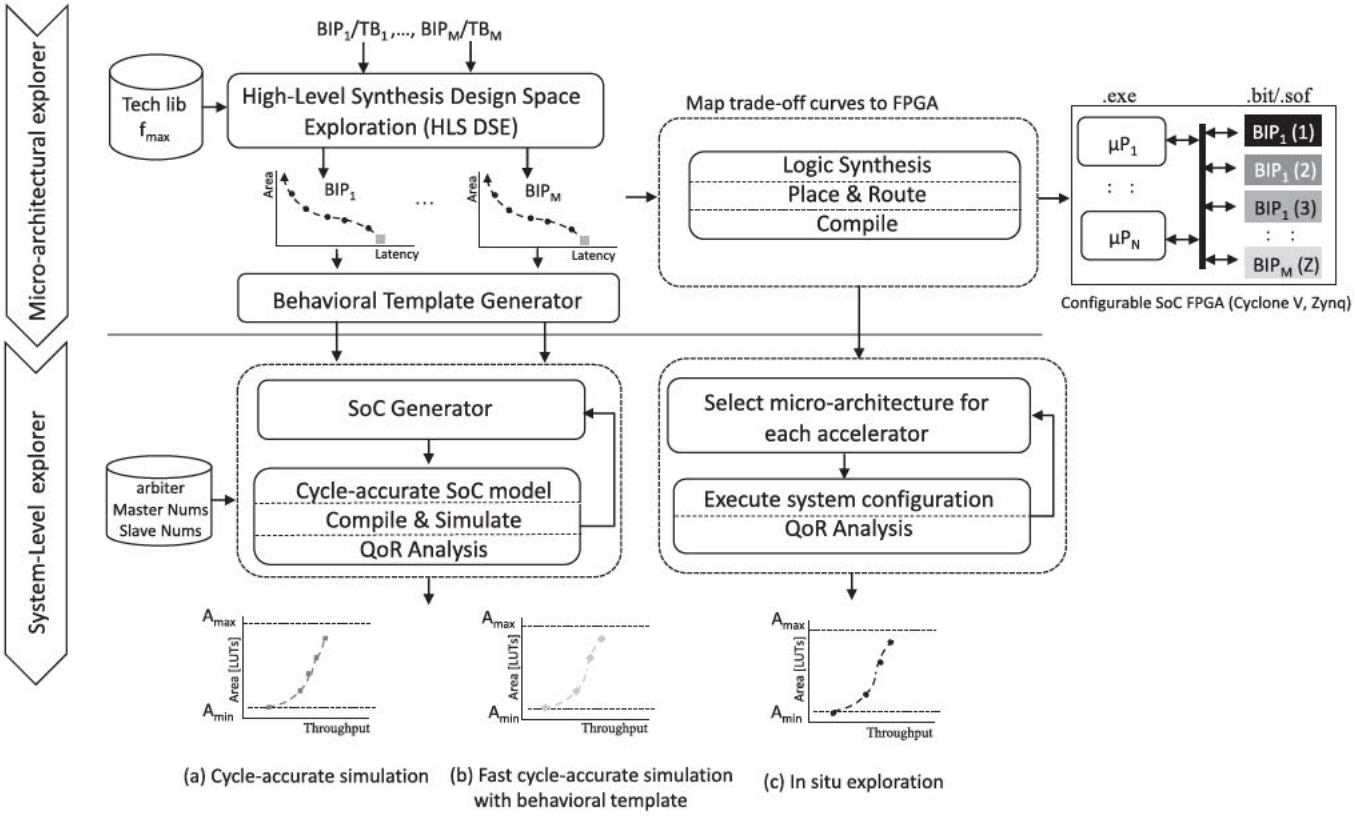

Fig. 2. Proposed methods flow diagram. (a) Offline simulation based method without template. (b) Offline simulation based method with template. (c) In situ system level exploration.

Step 1: SoC generation. This first step automatically generates the SoCs based on the number of masters and slaves specified and for each new configuration generates all possible task mappings. The order in which the tasks are executed on each master is not considered as all tasks are completely independent of each other. This work also considers that tasks are repeatedly executed.

The number of mappings follows the Stirling numbers of the second kind sequence if the task execution order is not considered. If it is considered, all possible permutations would be required. Based on preliminary simulation results, we could observe that for the case of periodically repeating tasks this was not the case and thus the task execution order is not considered. The Stirling numbers of the second kind $S(n, k)$ count the ways to divide a set of n objects into k nonempty subsets. In our case $n = N$, and $k = [1, N]$ where N is equal to the total number of slaves (BIPs).

Fig. 1 illustrated the effect of different tasks mappings on the area and overall system throughput as well as on the number of combinations. When the system only has 1 master ($M = 1$) only one mapping exists, which also leads to the slowest of all system configurations because the master now executes all the tasks, as indicated in Table 1. This case corresponds to $S(N, 1) = 1$. The number of task mapping combinations grows with the number of masters in the SoC, more task mapping combinations exists until $N/2$, which has the largest number of task mapping combinations ($S(N, N/2)$). Finally increasing the number of masters until $M = N$ leads again to a single task mapping as each task is mapped onto its own master, hence $S(N, N) = 1$. This configuration also typically leads to the fastest system. It should be noted that if the area of the masters is ignored, the total system area is virtually the same for all systems, as each system has the same number of slaves (although the bus complexity increases slightly with the number of masters and hence its area). In contrast, the performance will change with different mappings. The numbers of mappings in each case can be calculated as [43]:

$$S(n,k) = \frac{1}{k!}\sum_{i=0}^{k}(-1)^{k-i}\binom{k}{i}i^n \tag{1}$$

where n is the number of slaves, which is always constant $= N$ and k is the number of masters (M).

To generate valid mappings which can be simulated and synthesized, the original behavioral descriptions have to be modified to include a bus interface. For this purpose, commercial HLS tools provide a set of synthesizable APIs for different standard buses, *i.e.* AMBA AHB and AXI. The tasks merged into the same master must write to the correct memory mapped slave, by calling the API with its assigned address, while the slaves *listen* until a master initiates the communication with them. The Masters can send data in burst mode or as individual data (when possible burst mode is chosen in this work), while the slaves wait for the masters to transmit the data. Because the entire system should be synthesizable, the testbenches should also be given in synthesizable C or SystemC code. The output is hence a list of synthesizable behavioral descriptions for the masters $MList = \{M_1, M_2, \ldots, M_P\}$ and for the slaves $SList = \{S_1, S_2, \ldots, S_N\}$.

This step also generates the bus definition file, which the bus generator in the next step takes as input in order to create a complete C-based SoC. This bus definition file includes: (1) arbiter protocol (fixed or round robin), (2) memory map, (3) number of masters and slaves, (4) bus type (AHB or AXI) and (5) bus bitwidth. By default the values for (1) (4) and (5) are set to round robin, AHB and 32-bits, but can be set to any other values externally. It should be noted that our proposed method works with any arbitration policy and even other bus structures including Networks-on-Chip.

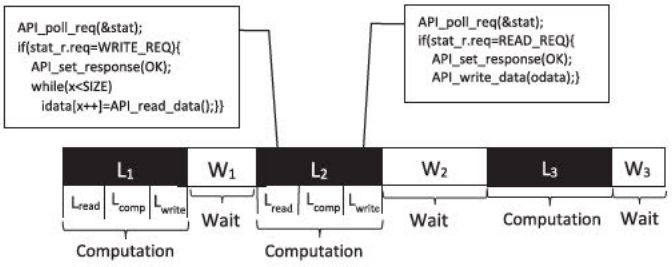

Fig. 3. Task execution schedule example including synthesizable read and write APIs.

In this case we would have to manually create the synthesizable API as no commercial tool, up to date, include synthesizable NoC interfaces.

Step 2: System generation. Once the system has been characterized in the previous step, the bus generator is called. This bus generator reads in the bus definition file created in the previous step and creates the synthesizable ANSI-C or SystemC code for the masters' ($MIFList = \{MIF_1, MIF_2, \ldots, MIF_P\}$) and slaves' interfaces ($SIFList = \{SIF_1, SIF_2, \ldots, SIF_N\}$) as well as for the bus (*bus*) itself and the top module (*top*) which instantiates all the components in the system. The commercial HLS tools used in this work (NEC's CyberWorkBench [44]) includes a bus generator that generates these automatically given the bus definition file created in step 1.

Step 3: HLS and cycle-accurate model generation. Once the system has been generated, each of the behavioral descriptions generated previously are synthesized. Because HLS is a single process synthesis method, each of them is synthesized individually with its own set of constraints. Once each of them is synthesized, a cycle-accurate model is generated in SystemC for the entire system. State of the art HLS tools also typically come with different model generators in order to verify the design at different levels of abstractions, e.g. behavioral-level (to verify the data type conversion) and cycle-accurate (to verify the timing). Once the system's cycle-accurate model is created, it is compiled using g++ and executed. All the BIPs used in this work were slightly modified to report the total time they remained in idle mode and the total time they were actively performing some computation. The results of the execution is a timing report indicating the idle and computational time of each of the slaves.

Step 4: Slave (BIP) optimizations. Based on the timing report obtained in the previous step, our method assigns to each slave a new micro-architecture from the trade-off curve generated for each BIP. Fig 3 graphically shows the report. It can be observed that at regular intervals the BIP receives the data from the master and computes it. It takes each BIP L_i cycles to finish the computation, where $L_i = \{L_{read} + L_{comp} + L_{write}\}$, with L_{read} the time required to read the data sent from the master, which is always constant once the communication has been established, L_{comp} the time taken to compute the new output and L_{write} the time taken to write the data back to the master. The only factor which changes between two executions of the same task is L_{write}, as the master has to retrieve the control over the bus, which might change between two executions. Moreover, the main difference in the execution of the task is in the waiting time between two consecutive executions. Based on the number of other tasks being executed, their bandwidth required to send and receive data and the arbiter's priority, which in the round robin case keeps changing. In this case $W_3 < W_1 < W_2$.

Our work considers these waiting cycles as positive slack, where the smallest waiting period (i.e. W_3) is the maximum slack. because the goal of our method is to find the smallest design which can sustain the same performance. This means that a micro-architecture with latency $L_{comp\_new} = floor(L_{comp} + W_{\min})$ is chosen from the pre-characterized micro-architectural exploration trade-off curve and the BIP substituted. Because the dominating curve does not contain designs of all latencies, the closest smallest value is chosen. This analysis is done for each of the slaves. Once all of the BIPs are substituted by their respective smallest designs, a new system is generated, re-synthesized and re-simulated to get accurate performance values. The same system choosing the smallest micro-architecture for each BIP is also generated as reference for each mapping in order to provide the user the range of systems that can be generated.

4.2. System exploration

Our exploration method uses the previously introduced slave optimization technique as its core to obtain dominating trade-off curve for systems with different number of masters and accelerators as well as the best task mapping (*MP*) on each of the masters. Hence, the result of our method is a trade-off curve (*TDC*) composed of unique micro-architectures ($micro_x$) of each *BIP* found in the pre-characterization stage when each BIP is explored, $TDC = \{BIP_1(micro_p), BIP_2(micro_q), \ldots, BIP_M(micro_w)\}$ and the best task mapping (*MP*). The best task mapping is the most efficient workload distribution among masters to maximize the throughput of the system. Nevertheless, our proposed method can also be set up to report a trade-off curve for the least favorable task mapping to take into account worst-case scenarios.

The complete exploration can be subdivided into 2 main steps. Algorithm 1 summarizes these steps, shown also graphically in Fig. 4 for a particular number of master. The main step can be summarized as follows:

Algorithm 1: System exploration.

input : $BIPL = \{BIP_1 = \{(A_1, L_1), A_p, L_p\}), \ldots\}, N\}$
BIPL: BIP list pre-characterized after DSE
N: Maximum number of masters
output: $TDCL = \{(TDC_{M=1}, MP_1), (TDC_{M=2}, MP_2), \ldots, (TDC_{M=N}, MP_N)\}$
TDLC: Trade of curves List for different masters
$TDC_{M=N}$: Trade of curve for system with *N* masters
MP_N: Task mapping for system with *N* masters

```
/* Step 1:;
Fastest Systems Generation */
foreach N do
    BIPL_max = select_fastest_microarch(BIPL);
    foreach MP do
        S_seed(A, P) = simulate_system(BIPL_max);
    end
end
/* Step 2: Explore each Fastest Design */
foreach N do
    foreach S_seed((A, P) do
        while (BIPs not smallest) do
            S_new(A, P, Slack) = simulate_system(S);
            optimize_micro(S_new(Slack), BIPL);
            select_new_BIP_smaller(BIPL);
        end
    end
    TDC_i = extract_tradeoff(S_all);
end
return(TDCL);
```

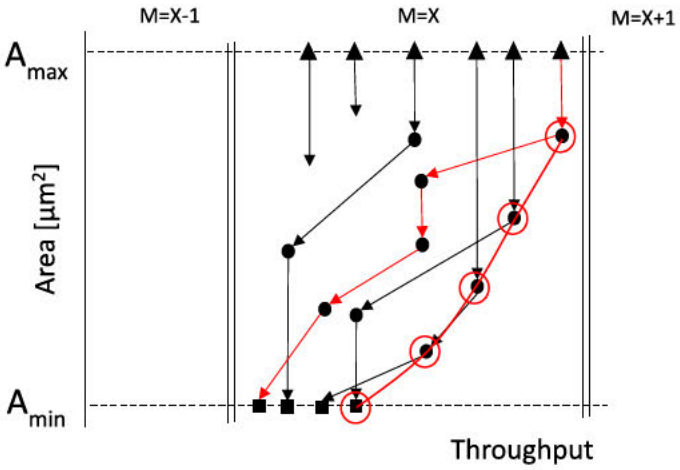

Fig. 4. System exploration overview.

Step 1: Fastest systems generations. This very first step selects the fastest micro-architectures for all of the BIPs and generates systems with all the different possible task mappings. Because these micro-architectures are the fastest, but also the largest in terms of area, they should lead to a system with the highest possible performance. These configurations are considered the *seed* configurations (S_{seed}) used to generate the final trade-off curve. They are depicted graphically in Fig. 4 by triangles.

Step 2: Single trade-off curve generation. Once the *seed* configurations have been generated, our explorer continues by following the steps described in the previous subsection for each *seed* configuration. This results in new systems S_{new} with the same performance as the S_{seed}, but smaller area. The explorer then continues choosing a new micro-architecture for the slave which has the smallest area difference among all the micro-architectures composing S_{new}. This leads, as shown in Fig. 4, to a smaller system, but slower. This step is repeated until a system containing only the smallest, but slowest micro-architectures are reached ($S_{smallest}$) (depicted in Fig. 4 as squares). Once this iteration is finished, the next S_{seed} is chosen and the same steps repeated. Once all the seed systems have been explored, the dominating configurations of all the configurations are kept and stored in the trade-off curve *TDC*.

4.3. Behavioral IP template

One of the problems of the proposed method so far, is that although a behavioral level cycle-accurate simulation is faster than an RT-level simulation, it is still often too slow, especially when dealing with very large systems. For this purpose, the concept of BIP templates was introduced in [3] to accelerate the system simulation. The idea behind these templates is to substitute each BIP with a template which mimics the BIPs' IOs behavior, but is *empty* inside. This implies that it only reads data from the master and returns data after *X* cycles similar to the original BIP, where *X* is the latency of the BIP. Although the results returned are functionally incorrect, the timing behavior is preserved. This strategy has several main advantages: Firstly, the workload pattern of the entire system is preserved (considering that the master is not using the returned data for control actions). Secondly, the compile time of the entire model is accelerated, as the complexity of these BIP templates is much lower than that of the actual BIPs (it should be noted that for larger circuits the compilation time can be significant). Thirdly, the cycle-accurate simulation is much faster as each template does not require to perform any actual computation. Lastly, it allows the exploration of configuration of any latencies

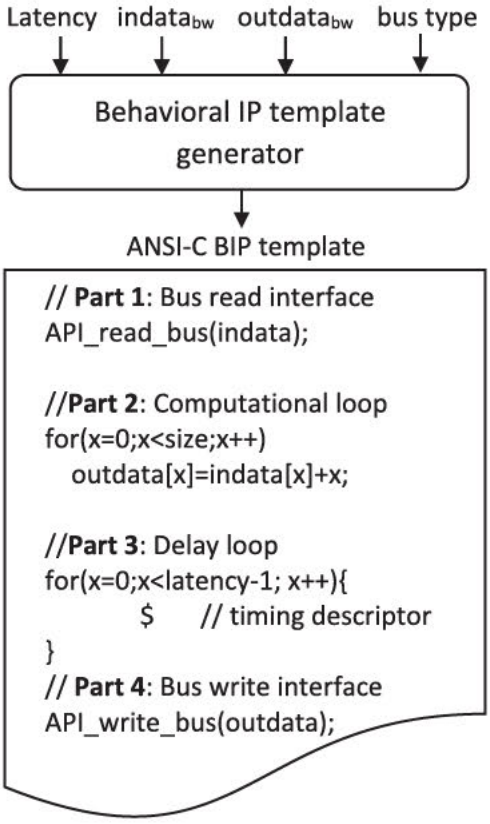

Fig. 5. BIP template generator and BIP template overview.

and not just those of the Pareto-optimal trade-off curve of each BIP. Hence it is very easy to generate different *what-if* scenarios.

Fig. 5 shows an overview of how these templates are generated and their main structure. The input to the template generator is the bitwidth of the inputs and outputs, the latency of the template and the bus type (AHB or AXI). The template generator takes these inputs and generates a synthesizable ANSI-C program. The template is composed of 4 main parts:

Part 1: Bus read interface. The first part contains the bus interface API required to read data from the bus.

Part 2: Computation loop. The second part performs some basic computation on the input data. This is important because the HLS tool would optimize the logic of the entire template away if no computation is performed. Hence this part ensures that the template structure is preserved.

Part 3: Delay loop. The third part is a for loop containing a timing description and the last part the bus interface API to write data back to the master. The timing descriptor symbolizes a clock step. Commercial HLS tools normally support different types of scheduling modes. The traditional one is the automatic scheduling, in which the HLS scheduler automatically times the behavioral description. Another way of doing this is by manually. Thus, these tools provide manual scheduling capabilities to allow designers to manually time the behavioral description. In SystemC this is done with *wait* statements, while at the ANSI-C level, this is vendor specific. In the case of the commercial HLS tool used in this work [44], a *$* sign is used to denote a clock boundary. Hence when having a loop with *N* iterations it will take *N* cycles to execute the loop.

Part 4: Bus write interface. Finally the last part returns the data to the master using the synthesizable API for the particular bus selected.

In order to create a more flexible template, the latency is passed as an input to each template, thus making it fully parameterizable at runtime.

4.4. In situ exploration

The draw-back of these cycle-accurate simulation models is that they can still be relatively slow, because the PC executes these sequentially. Another drawback is that the target platform can often not be exactly modeled, hence the results can slightly differ. For example in the case of the simulation based methods proposed previously, the master is modeled as a testbench (traffic generator). It would be prohibitory expensive to fully model the processors cycle-accurately. To address this, we also propose an alternative method based on in situ exploration, which performs the exploration on a CSoC FPGA. This FPGA contain multiple ARM cores and reconfigurable fabric onto which multiple accelerators can be mapped. In case that the actual target platform is also the CSoC then this method also guarantees achieving 100% accurate results as the CSoC being used in the exploration is also the final target architecture.

FPGA vendors provide system-level design tools for their CSoC design. *E.g.*, Xilinx's System Builder or Altera's Qsys. In this work, only Altera's CSoCs are targeted and hence only the interface to Qsys is built. Qsys is used to create the complete system using its AMBA AXI bus interface, mapping each dominating design obtained in step 1, as a slave in the system with a unique address. The entire system is synthesized, placed and routed and a configuration file generated (.sof). This configuration file (.sof) is then converted to raw binary file (.rbf), which enables the embedded system with the ability to configure the system at runtime.

This method is only constrained by the limited size of the reconfigurable fabric onto which to map the Pareto-optimal designs (dominating designs) of each of the HWAccs in the system. One possible solution would be to use a larger CSoC of the same family to perform the search and then use the system with specific micro-architectures for the production system, as this will only contain a single micro-architecture for each HWAcc. This would not jeopardize the quality of the results as the architecture of CSoCs are exactly the same.

The next section presents the experimental results showing the effectiveness of our three different proposed methods.

5. Experimental results

Different computational intensive applications, amenable to HW acceleration, were selected and grouped together into complex systems in order to test our proposed method. These designs were taken from the open source Synthesizable SystemC Benchmark suite (S2CBench) [45]. Table 2 shows how these complex benchmarks were formed. The first column indicates the name of the benchmark, the second column indicates the total number of dominating designs reported by the DSE for each benchmark. Columns S1-S8 indicate the number of instantiations of each test case used to build each complex benchmark. The last two rows report the total number of applications used in each system benchmark and total number of design candidates contained (adding up the results of the DSE of each application).

Table 2
Complex system benchmarks.

Bench	DSE	S1	S2	S3	S4	S5	S6	S7	S8
md5c	3	1	1		1		1		1
kasumi	2	1			1	1	1	1	1
interp	4			1	1		1	1	1
fir	3			1	1	1	1	1	1
adpcm	2		1	1		1	1	1	1
qsort	3	1	1			1		1	1
aes	2								
Tasks		3	3	3	4	4	5	5	6
Designs		8	8	9	12	10	14	14	17
Masters		1–3	1–3	1–3	1–4	1–4	1–4	1–4	1–4

The experiments were run on an Intel dual 2.40GHz Xeon processor machine with 16 GBytes of RAM running Linux Fedora release 19. The HLS tool used is NEC CyberWorkBench v.6.1 [44]. The target architecture, as mentioned previously, is a multi-core processor system with masters ranging from 1 to 4 depending on the benchmark. The masters and slaves are connected through a 32-bit AMBA AHB bus using a round robin arbiter. The HLS target frequency for all of the processes in the system is set to 100MHz.

The reconfigurable computing board used is a Terasic DE1-SoC Board. This board contains an Altera Cyclone V SoC 5CSEMA5F31 CSoC, which contains a dual-core ARM Cortex A-9 processor. Ubuntu 32-bit 15.1 was installed on this CSoC.

Due to the limited size of the reconfigurable fabric of the CSoC used, only 7 different BIPs are used. The size of the reconfigurable fabric also limits the maximum number of HWaccs in the experimental section to 17 (*S8*). When synthesized in Quartus II, the Logic utilization (in ALMs) for the largest system (*S8*) is 31,572/32,070 (98%). We believe that the number of HWaccs as well as dominating designs used in this section should serve as a proof of concept for our proposed method. A larger FPGA from the same family could be used in the case that trade-off curves with larger number of HWaccs need to be explored. The results should be exactly the same as the underlying architecture is the same and only the number of logic resources change, thus allowing to map the additional configurations onto the fabric.

The proposed exploration framework was executed also with different number of masters ranging from 1 to 4, but because the target CSoC only contains a dual-core ARM processor, results up to two masters are only shown for the in-situ exploration.

Tables 3 and 4 show the qualitative and quantitative results respectively of our method without behavioral templates (*FEBS*), making use of the behavioral templates to speed up the simulation ($FEBS_{template}$) and also an in situ search ($FEBS_{insitu}$). The proposed methods are compared against a brute force search method without templates which for each system tries all possible micro-architectures reported by the DSE.

The main problem when comparing different multi-objective function optimization methods is how to measure the quality of the results. Closeness to the Pareto front, wider range of diverse solutions, or other properties are some of them. Several studies can be found in the literature that address the problem of comparing approximations of the trade-off surface in a quantitative manner. Most popular are unary quality measures, *i.e.* the measure assigns each approximation set a number that reflects a certain quality aspect, and usually a combination of them is used [46,47]. A multitude of unary indicators exist *e.g.* hypervolume indicator, average best weight combination, distance from reference set and spacing. Zitzler et al. provide a good review of all existing methods in [48], indicating that there isn't any single indicator able to measure the quality of the results. Nevertheless quality measures are necessary in order to compare the outcome of the DSE. In this work we measure the quality of the different methods using the following criteria, which are also the main indicators used in this field:

Average Distance from Reference Set (ADRS): This measure (*ADRS*) indicates how close a Pareto-front is to the reference front. The smaller the value the closer the obtained approximate front is to the reference front. Given a reference Pareto front $\Gamma = \gamma_1 = (a_1, l_1), \gamma_2 = (a_2, l_2), \ldots, \gamma_n = (a_n, l_n)$ and an approximate Pareto front $\Omega = \omega_1 = (a_1, l_1), \omega_2 = (a_2, l_2), \ldots, \omega_n = (a_n, l_n)$ with a $\in A$ and l $\in L$, where A is the designs area and l its correspondent latency. It follows: $ADRS(\Gamma, \Omega) = \frac{1}{|\Gamma|} \sum_{\gamma \in \Gamma} \min_{\omega \in \Omega} f(\gamma, \omega)$ where

$$f(\gamma = (a_\gamma, l_\gamma), \omega = (a_\omega, l_\omega)) = max\left\{ \left| \frac{a_\omega - a_\gamma}{a_\gamma} \right|, \left| \frac{l_\omega - l_\gamma}{l_\gamma} \right| \right\}.$$

Table 3
Experimental results: Quality of Results (QoR) (ADRS and dominance) comparison between in situ exploration ($FEBS_{insitu}$) and proposed method without (*FEBS*) and with behavioral template ($FEBS_{template}$) taken the *BF* as reference in %.

		FEBS		$FEBS_{template}$		$FEBS_{insitu}$	
Bench	Masters	ADRS[%]	Dom[%]	ADRS[%]	Dom[%]	ADRS[%]	Dom[%]
S1	M=1	0.0	100	0	100	0.1	98
	M=2	1.1	50	1.3	75.0	0.0	100
	M=3	1.8	60	1.5	70.0	–	–
S2	M=1	2.2	65	2.2	65.0	0.7	77
	M=2	4.1	75	4.1	75.0	0.2	87
	M=3	5.3	43	5.3	42.9	–	–
S3	M=1	1.4	71	3.4	71.4	0.0	100
	M=2	3.7	50	1.2	75.0	2.0	90
	M=3	1.4	71	1.6	66.0	–	–
S4	M=1	3.2	41	3.2	41	0.1	90
	M=2	0.0	100	0.0	100.0	0.0	100
	M=3	3.2	41	3.9	29.4	–	–
	M=4	4.6	50	4.6	50.0	–	–
S5	M=1	0.0	100	0.0	100.0	0.0	100
	M=2	0.0	100	0.0	100.0	0.0	100
	M=3	0.4	75	2.3	50.0	–	–
	M=4	1.4	82	1.4	82	–	–
S6	M=1	0.0	100	0.0	100.0	0.0	100
	M=2	3.3	42	3.9	25.0	1.2	70
	M=3	4.2	55	4.5	25.0	–	–
	M=4	3.8	65	3.9	25.0	–	–
S7	M=1	0.0	100	0.0	100.0	0.0	100
	M=2	4.3	42	4.3	42	0.5	80
	M=3	3.8	58	3.8	58	–	–
	M=4	4.5	75	4.8	25.0	–	–
S8	M=1	0.0	100	0.0	100.0	0.0	100
	M=2	3.0	43	3.4	28.6	0.9	85
	M=3	2.8	50	4.6	44.0	–	–
	M=4	3.2	33	4.3	32.0	–	–
Avg.	–	2.3	65	2.6	59.7	0.35	93

Table 4
Running time results [min].

	BF	*FEBS*	$FEBS_{template}$	$FEBS_{insitu}$	Comparison		
Bench	Run[min]	Run[min]	Run[min]	Run[min]	Δ_{BF-FH}	Δ_{BF-FHT}	Δ_{BF-FHI}
S1	58	24	20	10	2.42	2.90	5.8
S2	113	23	20	8.5	4.91	5.65	13.2
S3	70	19	15	12.7	3.68	4.67	5.5
S4	827	102	50	27	8.11	16.54	30.6
S5	386	69	42	34	5.59	9.19	11.4
S6	911	534	152	122	1.71	5.99	7.5
S7	1514	576	154	113	2.63	9.83	13.4
S8	1387	2575	542	355	0.54	2.56	3.9
Avg.	–	–	–	–	3.42	6.64	11.4
Geomean	544	208	90	38	–	–	–

The lower the distance value (ADRS) is, the more similar two Pareto sets are. For example, a high *ADRS* value tells that an entire region of the reference Pareto-front is missing in the approximation set.

Pareto Dominance: This index is equal to the ratio between the total number of designs in the Pareto set being evaluated (obtained by executing one exploration method), also present in the reference Pareto set. The reference Pareto set is obtained by combining the best results of each method over 5 runs. The higher the value, the better the Pareto set is.

When presenting DSE results it is also often custom to show the trade-off curves graphically in order to provide a quick visual representation of the quality of the results. Unfortunately, in this work 32 trade-off curves would need to be shown making it impractical. Nevertheless, Fig. 6 shows the trade-off curves for the in-situ implementation for our fast search method and the brute force method which is used as the reference front to compare the quality of results for three proposed methods for the case of 1 and 2 masters.

Different conclusions can be drawn from the results shown in Tables 3 and 4.

From Table 4 it can be seen that the in-situ method is the fastest, followed by the template based method and followed by the detailed cycle-accurate one. In particular they are on average 14.3 ×, 6.0 ×, and 2.6 × faster compared to the brute forces based on the detailed cycle-accurate simulation. It should be noted that the running times include the time required to place and route the designs on the FPGA for the in-situ method and the compilation time for the simulation based methods.

In terms of the Quality of Results (QoR) our three proposed methods also show to lead to good results with average *ADRS* of 2.3% and 2.6% without and with template and 0.35% for in situ method. The template based method works specially very well, for data-intensive applications with no conditionals that affect the

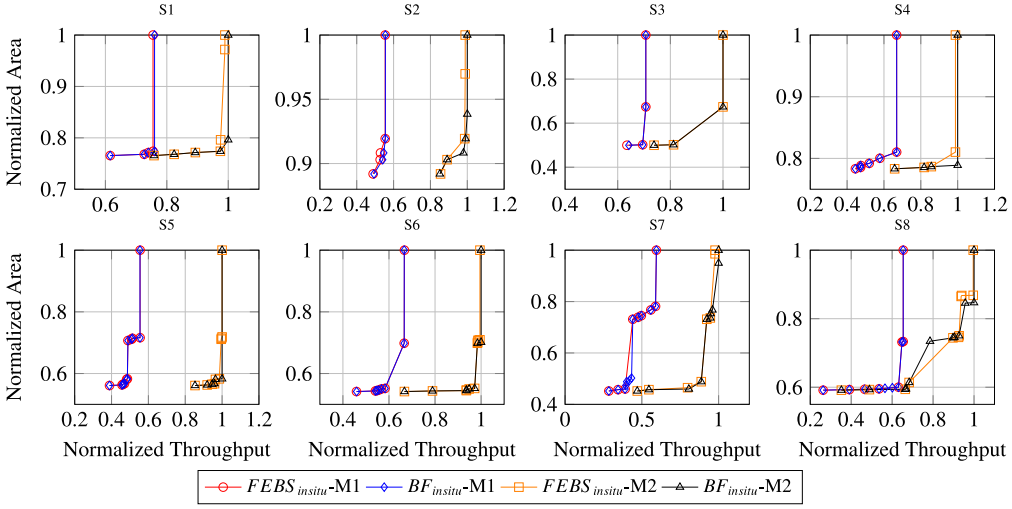

Fig. 6. System Exploration trade-off curves for each benchmark for 1 master (M=1) and 2 masters (M=2) comparing the brute force (BF_{insitu}) and our proposed fast heuristic (*FEBSinsitu*).

Table 5
Comparison between the 3 proposed exploration methods.

Method	Speed	Quality of results
FEBS	+	++
$FEBS_{template}$	++	+/++[a]
$FEBS_{insitu}$	+++	+++[b]

[a] Better results when no runtime data dependencies in benchmark that affect the latencies.

[b] Under specific circumstances: (1) Only 2 masters allowed (2) the complete trade-off curves of all accelerators fits on the reconfigurable fabric.

latency of the circuit (*.e.g* no breaks in for loops). Our previous work [3] studies this extensively.

Table 5 compares all three methods based on the running time results and quality of the exploration results. The *FEBS* is a fast method that leads to good results. The templates based simulation version $FEBS_{template}$ is faster, but leads to slightly worse results, while the in-situ based method $FEBS_{insitu}$ is the fastest and leads to the best results subject to different conditions: First, that the SoC only has 2 master and lastly that the complete trade-off curves of all the accelerators can be mapped on the reconfigurable fabric of the FPGA.

It can be therefore concluded that our three methods are very effective and that they lead to comparable results compared to an exhaustive search much faster.

6. Summary and conclusions

In this work we have presented three fully automatic methods to characterize heterogeneous systems composed of multiple masters and hardware accelerators in shared bus systems. In particular, this work makes use of advanced features available in state-of-the-art HLS tools, which allow the generation of complete SoC systems in C and also their simulation with cycle-accurate models. Because these BIPs often have to wait for data to be transferred from the masters and also wait to get permission from the bus arbiter to access the bus, it was shown that it is not necessary to implement these BIPs maximizing performance and hence using more hardware resources. For different mappings with different number of masters generating different traffic patterns, it was shown that our methods are very effective compared to an exhaustive search while being much faster. The concept of behavioral IP template was also introduced to further accelerate the running time of the proposed method. We have shown that the quality of the results only suffers minor degradations in designs with data-dependent latencies.

Also, a fast and accurate method to explore the design space of heterogeneous MPSoCs mapped on a CSoC by performing the exploration in situ on the same target platform is also presented. This leads to 100% accurate results, while at the same time accelerates the exploration time compared to a method based on using cycle-accurate models. This method is only constrained by the limited size of the reconfigurable fabric onto which to map the Pareto-optimal designs (dominating designs) of each of the HWAccs in the system. One possible solution would be to use a larger CSoC of the same family to perform the search and then use the system with specific micro-architectures for the production system, as this will only contain a single micro-architecture for each HWAcc. This would not jeopardize the quality of the results as the architecture of CSoCs are exactly the same.

References

[1] International Technology Roadmap for Semiconductors, www.public.itrs.net/Links/2013ITRS/2013Chapters, ITRS, 2013.

[2] Y. Liu, M. Villaverde, F. Moreno, B.C. Schafer, Characterization and optimization of behavioral hardware accelerators in heterogeneous mpsocs, in: 2017 12th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2017, pp. 1–8, doi:10.1109/ReCoSoC.2017.8016158.

[3] A. Mahapatra, Y. Liu, B. Carrion Schafer, Accelerating cycle-accurate system-level simulations through behavioral templates, Integration 34 (1) (2018) 1–14.

[4] S. Xu, B.C. Schafer, Y. Liu, Configurable soc in-situ hardware/software co-design design space exploration, in: 2017 IEEE International Conference on Computer Design (ICCD), 2017, pp. 509–512, doi:10.1109/ICCD.2017.88.

[5] N. Bombieri, H.-Y. Liu, F. Fummi, L. Carloni, A method to abstract rtl ip blocks into c++ code and enable high-level synthesis, in: Proceedings of the 50th Annual Design Automation Conference, in: DAC '13, 2013, pp. 156:1–156:9.

[6] A. Mahapatra, B. Carrion Schafer, Veriintel2c: abstracting rtl to c to maximize high-level synthesis design space exploration, Integration 34 (1) (2018) 1–12.

[7] J. Hauser, J. Wawrzynek, Garp: a MIPS processor with a reconfigurable coprocessor, in: FCCM, 1997, pp. 12–21.

[8] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin, R. Espasa, E. Grochowski, T. Juan, P. Hanrahan, Larrabee: a Many-core x86 architecture for visual computing, in: ACM SIGGRAPH 2008 Papers, in: SIGGRAPH '08, ACM, 2008, pp. 18:1–18:15.

[9] H. Singh, M.-H. Lee, G. Lu, F. Kurdahi, N. Bagherzadeh, E. Chaves Filho, Morphosys: an integrated reconfigurable system for data-parallel and computation-intensive applications, Comput., IEEE Trans. 49 (5) (2000) 465–481.

[10] N. Clark, A. Hormati, S. Mahlke, VEAL: virtualized execution accelerator for loops, in: ISCA, 2008, pp. 389–400.

[11] H. Tabkhi, R. Bushey, G. Schirner, Function-level processor (FLP): raising efficiency by operating at function granularity for market-oriented MPSoC, in: ASAP, 2014, pp. 121–130.
[12] P. Mantovani, G. Di Guglielmo, L. Carloni, High-level synthesis of accelerators in embedded scalable platforms, in: Asia Sout Pacific Design Automation Conference (ASP-DAC), 2016, pp. 1–6.
[13] J. Cong, M.A. Ghodrat, M. Gill, B. Grigorian, K. Gururaj, G. Reinman, On-chip interconnect network for accelerator-rich architectures, in: DAC, 2015, pp. 389–400.
[14] T. Givargis, F. Vahid, J. Henkel, System-level exploration for Pareto-optimal configurations in parameterized systems-on-a-chip, in: ICCAD, 2001, pp. 25–30.
[15] W. Fornaciari, D. Sciuto, C. Silvano, V. Zaccaria, A sensitivity-based design space exploration methodology for embedded systems, Des. Autom. Emb. Syst. 7 (1–2) (2002) 7–33.
[16] C. Erbas, S. Cerav-Erbas, A.D. Pimentel, Multiobjective optimization and evolutionary algorithms for the application mapping problem in multiprocessor system-on-chip design, Trans. Evol. Comp. 10 (3) (2006) 358–374.
[17] F. Ferrandi, P.L. Lanzi, C. Pilato, D. Sciuto, A. Tumeo, Ant colony heuristic for mapping and scheduling tasks and communications on heterogeneous embedded systems, IEEE Trans. CAD Integr. Circuits Syst. 29 (6) (2010) 911–924.
[18] M. Lukasiewycz, M. Glass, C. Haubelt, J. Teich, Efficient symbolic multi-objective design space exploration, in: ASP-DAC, 2008, pp. 691–696.
[19] G. Beltrame, L. Fossati, D. Sciuto, Decision-theoretic design space exploration of multiprocessor platforms, IEEE Trans. CAD Integr. Circuits Syst. 29 (7) (2010) 1083–1095.
[20] H. Liu, M. Petracca, L.P. Carloni, Compositional system-level design exploration with planning of high-level synthesis, in: 2012 Design, Automation Test in Europe Conference Exhibition (DATE), 2012, pp. 641–646, doi:10.1109/DATE.2012.6176550.
[21] G. Di Guglielmo, C. Pilato, L.P. Carloni, A design methodology for compositional high-level synthesis of communication-centric socs, in: Proceedings of the 51st Annual Design Automation Conference, in: DAC '14, ACM, New York, NY, USA, 2014, pp. 128:1–128:6.
[22] L. Piccolboni, P. Mantovani, G.D. Guglielmo, L.P. Carloni, Cosmos: coordination of high-level synthesis and memory optimization for hardware accelerators, ACM Trans. Embed. Comput. Syst. 16 (5s) (2017) 150:1–150:22.
[23] B. Khailany, E. Khmer, R. Venkatesan, J. Clemons, J.S. Emer, M. Fojtik, A. Klinefelter, M. Pellauer, N. Pinckney, Y.S. Shao, S. Srinath, C. Torng, S.L. Xi, Y. Zhang, B. Zimmer, A modular digital vlsi flow for high-productivity soc design, in: Proceedings of the 55th Annual Design Automation Conference, in: DAC '18, ACM, New York, NY, USA, 2018, pp. 72:1–72:6.
[24] Y. Corre, V.-T. Hoang, J.-P. Diguet, D. Heller, L. Lagadec, HLS-based fast design space exploration of ad hoc hardware accelerators: a key tool for MPSoC synthesis on FPGA, in: DASIP, 2012, pp. 1–8.
[25] Y. Corre, J.-P. Diguet, L. Lagadec, D. Heller, D. Blouin, Fast template-based heterogeneous MPSoC synthesis on FPGA, in: Proceedings of the 9th International Conference on Reconfigurable Computing: Architectures, Tools, and Applications, in: ARC'13, 2013, pp. 154–166.
[26] E. Sotiriou-Xanthopoulos, S. Xydis, K. Siozios, G. Economakos, D. Soudris, Rapid prototyping and design space exploration methodologies for many-accelerator systems, in: International Conference on Field Programmable Logic and Applications (FPL), 2015, pp. 1–2.
[27] F. Bellard, QEMU, a fast and portable dynamic translator, in: USENIX, 2005, p. 41.
[28] T. Austin, E. Larson, D. Ernst, SimpleScalar: an infrastructure for computer system modeling, Computer 35 (2) (2002) 59–67.
[29] OVP, 2015. [Online]. Available: www.ovpworld.org.
[30] M. Lis, et al., Scalable, accurate multicore simulation in the 1000-core era, in: ISPASS, 2011, pp. 175–185.
[31] Y. Corre, et al., HLS-based Fast Design Space Exploration of ad hoc hardware accelerators: a key tool for MPSoC Synthesis on FPGA, in: DASIP, IEEE, 2012, pp. 1–8.
[32] S. Gheorghita, et al., System-scenario-based design of dynamic embedded systems, ACM Trans. Des. Autom. Electron. Syst. 14 (1) (2009) 3:1–3:45.
[33] P. van Stralen, A. Pimentel, Scenario-based design space exploration of MPSoCs, in: ICCD, 2010, pp. 305–312.
[34] R. Electronics, Stream transpose processor, 2018,
[35] T. Toi, N. Nakamura, Y. Kato, T. Awashima, K. Wakabayashi, L. Jing, High-level synthesis challenges and solutions for a dynamically reconfigurable processor, in: ICCAD, ACM, 2006, pp. 702–708.
[36] T. Schwarzer, J. Falk, M. Glaß, J. Teich, C. Zebelein, C. Haubelt, Throughput-optimizing compilation of dataflow applications for multi-cores using quasi-static scheduling, in: Proceedings of the 18th International Workshop on Software and Compilers for Embedded Systems, in: SCOPES '15, 2015, pp. 68–75.
[37] K. Bertels, V.M. Sima, Y. Yankova, G. Kuzmanov, W. Luk, G. Coutinho, F. Ferrandi, C. Pilato, M. Lattuada, D. Sciuto, A. Michelotti, Hartes: hardware-software codesign for heterogeneous multicore platforms, IEEE Micro 30 (5) (2010) 88–97.
[38] J. Henkel, A low power hardware/software partitioning approach for core-based embedded systems, in: Proceedings of the 36th Annual ACM/IEEE Design Automation Conference, in: DAC '99, ACM, New York, NY, USA, 1999, pp. 122–127.
[39] G. Stitt, R. Lysecky, F. Vahid, Dynamic hardware/software partitioning: a first approach, in: Proceedings of the 40th Annual Design Automation Conference, in: DAC '03, ACM, 2003, pp. 250–255.
[40] J.W. Tang, Y.W. Hau, M. Marsono, Hardware/software partitioning of embedded system-on-chip applications, in: 2015 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC), 2015, pp. 331–336.
[41] B. Carrion Schafer, K. Wakabayashi, Machine learning predictive modelling high-level synthesis design space exploration, IET Comput. Digital Tech. 6 (3) (2012) 153–159.
[42] B. Carrion Schafer, T. Takenaka, K. Wakabayashi, Adaptive simulated annealer for high level synthesis design space exploration, in: VLSI-DAT, 2009, pp. 106–109.
[43] H. Sharp, Cardinality of finite topologies, J. Combinational Theory 5 (1) (1968) 82–86.
[44] NEC CyberWorkBench, 2017, [Online]. Available: www.cyberworkbench.com.
[45] B. Carrion Schafer, A. Mahapatra, S2CBench:Synthesizable systemC benchmark suite for high-Level synthesis, IEEE Embed Syst. Lett. 6 (3) (2014) 53–56.
[46] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: Nsga-ii, IEEE Trans. Evol. Comput. 6 (2) (2002) 182–197.
[47] D.A.V. Veldhuizen, G.B. Lamont, On measuring multiobjective evolutionary algorithm performance, in: Evolutionary Computation, 2000. Proceedings of the 2000 Congress on, 1, 2000, pp. 204–211 vol.1.
[48] E. Zitzler, L. Thiele, M. Laumanns, C.M. Fonseca, V.G. da Fonseca, Performance assessment of multiobjective optimizers: an analysis and review, IEEE Trans. Evol. Comput. 7 (2) (2003) 117–132.

Siyuan Xu received the B.E. degree in microelectronics from the Guangdong University of Technology, Guangzhou, China, in 2014, and the M.Sc. degree (Hons.) in electronic and information engineering from the Hong Kong Polytechnic University, Hong Kong, in 2016. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, The University of Texas at Dallas, Richardson,TX, USA, where he investigates the approximate computing in high-level synthesis.

Shuangnan Liu received the M.Sc. degree from the Hong Kong University of Science and Technology in 2014. He is currently pursuing the Ph.D. degree with the Department of Electronic and Information Engineering, where he investigates data-flow system optimizations in high-level synthesis.

Félix Moreno was born in Valladolid, Spain, in 1959. He received the M.Sc. and Ph.D. degrees in telecommunication engineering from Universidad Politécnica de Madrid (UPM), in 1986 and 1993, respectively. Currently, he is Associate Professor of Electronics at UPM. His research interests are focused on evolvable hardware, high-performance reconfigurable and adaptive systems, hardware embedded intelligent architectures, and digital signal processing systems.

Benjamin Carrion Schafer completed his Ph.D. at the University of Birmingham, U.K. in 2002. He then worked in the Computer Science Department at the University of California Los Angeles (UCLA) as a Postdoctoral Researcher from 2003 to 2004 and joined the School of Electronic Engineering and Computer Science at Seoul National University, Korea, as a Visiting Research Scholar from 2005 to 2007. From 2007 until September 2012, he was a researcher at the System IP Core Department, Central R&D Centre, NEC Corporation, Kawasaki, Japan. From 2012 to 2016, he worked as an assistant professor at the Department of Electronic and Information Engineering (EIE) at the Hong Kong Polytechnic University, where he established the Design Automation and Reconfigurable Computing Laboratory (DARClab). Since 2016, he works as an assistant professor at the Department of Electrical and Computer Engineering at the University of Texas at Dallas. Dr. Carrion Schaefer has been engaged in the research and development of VLSI systems, reconfigurable computing, thermal-aware VLSI design and High-Level Synthesis(HLS). He has over 30 publications as the first author in international scientific journals, conferences and books. He served on the TPC of most EDA and FPGA conferences including ASPDAC, CASES, DAC, and ESLSyn, FPL, RECONFIG. He was also a member of Accelleras SystemC synthesizable user group committee, leading the effort to standardize asynthesizable subset of SystemC. He holds an MBA from McGill University.