A Generic Infrastructure for System-level MP-SoC Design Space Exploration

Zai Jian Jia, Andy D. Pimentel, *Senior Member, IEEE,* Mark Thompson, Tomás Bautista and Antonio Núñez

***Abstract*— In this paper, we present a new and generic system-level MP-SoC DSE infrastructure, called NASA (Non Ad-hoc Search Algorithm). This highly modular framework uses well-defined interfaces to easily integrate different system-level simulation tools as well as different combinations of search strategies in a simple plug-and-play fashion. Moreover, NASA deploys in a so-called dimension-oriented DSE approach, allowing designers to configure the appropriate number of, possibly different, search algorithms to simultaneously co-explore the various design space dimensions. As a result, NASA provides a flexible and re-usable framework for the systematic exploration of the multi-dimensional MP-SoC design space, starting from a set of relatively simple user specifications. To demonstrate the capabilities of NASA framework and to illustrate its distinct aspects, we also present several DSE experiments in which we, e.g., compare NASA configurations using a single search algorithm for all design space dimensions to configurations using a separate search algorithm per dimension. These experiments indicate that the latter multi-dimensional co-exploration can find better design points and evaluates a higher diversity of design alternatives as compared to the more traditional approach of using a single search algorithm for all dimensions.**

***Index Terms*— MP-SoC design, performance analysis and design aids, system-level design space exploration.**

I. Introduction

Today's embedded systems are increasingly based on multi-processors systems-on-chip (MP-SoC). These MP-SoCs typically contain multiple storage elements, networks, I/O components, and a number of heterogeneous programmable processors for flexible application support as well as dedicated processing elements for achieving high performance and power goals [1]. In order to cope with the design complexity of such systems in a time-efficient way, the abstraction level of the design process has in recent years been raised towards the system-level. Design Space Exploration (DSE) is a key ingredient of such system-level design, during which a wide range of design choices are explored, especially during the early design stages. Such early DSE is of paramount importance as early design choices heavily influence the success or failure of the final product, and can avoid wasting time and effort in further design steps without the possibility of meeting design requirements because of an inappropriate system architecture design. The process of system-level DSE logically consists of two interdependent components [2]: (i) evaluation of a design point in the design space using e.g. analytical models or (system-level) simulation, and (ii) the search mechanism to systematically travel through the design space.

Both DSE components have received significant research attention during the last decades, e.g., [3]-[8]. For instance, system-level simulation is a popular method for evaluating single design points [2]. These simulation tools usually operate at a high level of abstraction and are often based on the Y-Chart principle [9], [10]. According to this approach, any system can be specified with the combination of three models: an application model, an architecture model and a mapping model. The latter means that the Y-Chart principle decouples application from architecture by recognizing two distinct models for them. An application model – derived from a target application domain – describes the functional behaviour of the application (using, e.g., Kahn Process Networks or tasks-graphs) in an architecture-independent manner. Simultaneously, an architecture model – defined with the application in mind – defines the architecture resources and captures their performance constraints. Finally, an explicit step (or model) maps the application model onto an architecture model for co-simulation, after which distinct system metrics can be quantitatively evaluated.

However, these simulation tools only provide a partial solution since an overall framework is needed to systematically explore the design space. Such a system-level DSE framework should allow for exploring a wide variety of system parameters and design choices, including the number and type of processing elements in the MP-SoC platform, the type of on-chip network, the memory organization, the mapping of application tasks and communications onto architecture resources, scheduling policies, and so on. Evidently, the more details (or dimensions) taken into account, the larger the design space that needs to be searched, and therefore the more costly the analysis. Although many

Manuscript received July 28, 2010. This work was supported in part by Spanish Ministry for Science and Technology.

Zai Jian Jia, Tomás Bautista and Antonio Núñez are with the Research Institute for Applied Microelectronic, University of Las Palmas de Gran Canaria, Spain; e-mail: {cjia, bautista, nunez}@ iuma.ulpgc.es).

Andy D. Pimentel and Mark Thompson are with the Computer Systems Architecture Group, Informatics Institute, University of Amsterdam, The Netherlands; (e-mail: {a.d.pimentel, mthompson}@uva.nl).

DSE approaches based on a large variety of search techniques have been proposed, three common factors can be identified in all of them:

1) DSE efforts are usually targeted to a specific system-level simulation tool (or analytical evaluation method), where each effort typically uses a different kind of simulator. Consequently, it is hard to re-use these DSE frameworks and elements in them.
2) Setting up the DSE experiments can be very labour intensive. It is often the case that for every experiment, control scripts need to be (re-)written to manipulate the simulation parameters and configuration files (specifying the design instance to evaluate) according to the algorithm that searches through the design space. These scripts are often inflexible and hard to re-use for different types of DSE experiments, i.e., assessing different parameters or parameter ranges.
3) In spite of the wide variety of eligible architectures for implementing embedded systems applications, many DSE experiments are focused on a particular class of MP-SoC architectures only. Moreover, designers have to write such models manually. This latter is an error-prone task and one of the bottlenecks in improving the designer's productivity, and severely limits the amount of the design space that can be explored in a reasonable time.

In summary, to the best of our knowledge, there does not exist a generic infrastructure to facilitate and support system-level MP-SoC DSE experiments, and to foster the re-use of software in the context of system-level MP-SoC DSE. This calls for a unified framework integrating and coupling both simulation and search mechanisms to efficiently and systematically explore design spaces, as well as a fast tool to automatically generate a wide range of architecture models, so that a large variety of architectures can be easily explored and evaluated. The resulting relationship between these three components is shown in Fig. 1.

To address the above challenges, this paper presents a new generic system-level DSE infrastructure implemented in C++, called NASA (Non Ad-hoc Search Algorithm). Its main goal is to provide a single, common, and modular framework for system-level DSE experiments. It allows for incorporating different (existing) system-level simulation tools as well as different combinations of search strategies by means of a simple plug-in mechanism. An architectural platform generator has also been integrated in NASA to free designers from the efforts to manually create architecture models. Thus, this automation improves the design productivity and enables the designer to focus on the more valuable issue of making design decisions. As a consequence, the NASA framework provides a flexible and re-usable environment to systematically explore the multidimensional MP-SoC design space, starting from a set of relatively simple user specifications. NASA's output includes information about all explored design points as well as a set of optimal design points within the explored design space, which best meet the user constraints such as real-time application constraints,

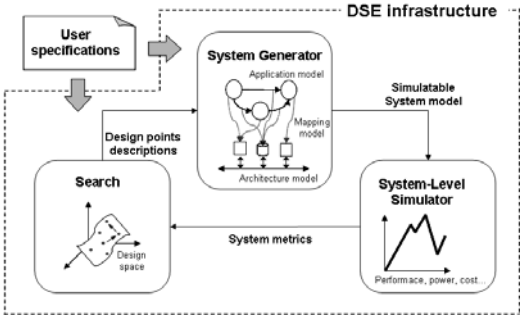


Fig. 1. Integration of an external system-level simulator with searching mechanism and a system generator in a single DSE infrastructure.

number and types of available components in the platform architecture, costs/area, etc.

The remainder of the paper is organized as follows. In the next section, related work and our contributions are presented. In Section III, we describe various implementation aspects of NASA framework. In Section IV, we present a range of experimental results, demonstrating NASA's capabilities. Finally, Section V concludes the paper.

II. RELATED WORK AND CONTRIBUTIONS

Performing DSE in a time-efficient and accurate way is not a new problem and there exists a large body of related work in this area. Most of the approaches in the embedded systems domain are targeted to the system-level exploration of heterogeneous MP-SoC [3]-[5], [8], [11]. Although these efforts are fairly efficient to explore various alternatives for mapping a specific application onto a target MP-SoC architecture, they typically still require significant effort to (re-)write scripts that control the evaluation mechanism (analytical model or simulator) during the search through the design space. In fact, this often means that there exists a repetitive effort to build customized scripts and/or architecture models for every different kind of DSE experiment. Thus, automating such a process becomes a key element in terms of reusability and flexibility for larger design space explorations in the design of a heterogeneous multiprocessor architecture.

Several proposals to integrate external design-point evaluation tools in a DSE environment can also be found in literature. In [12], a hierarchical and three-phase DSE methodology is presented. It facilitates the integration of simulators by using a set of tool-dependent interpreters or adapters. Angiolini *et al.* [13] present a framework that integrates an ASIP tool-chain within a virtual platform to explore a number of axes of the MP-SoC configuration space. However, unlike our work, this framework does not allow the integration of external search methods. Moreover, it still requires human intervention in the feedback loop of the searching and optimization process.

The MultiCube project [14] has similar objectives as the work presented in this paper, but it targets the exploration of the configuration space of homogeneous chip multiprocessors rather than system-level MP-SoC platform DSE. This implies that it has limited or no capabilities to explore different

application to architecture mappings, heterogeneous processing elements, different interconnections, and so on.

Other works have also developed a modular interface-based system-level MP-SoC DSE framework [15], [16]. In these cases, different search algorithms can be plugged in, but the resulting DSE is limited in terms of the target MP-SoC platforms that can be explored. This last aspect has been addressed in [17]. Künzli *et al.* [17] proposed a generic and modular framework based on PISA [18] for DSE of embedded system. The PISA interface separates the problem-dependent variation and estimation part from the generic search and selection. The resulting two parts are implemented as independent processes that are communicating via text files. But, unlike our work and to the best of our knowledge, they have only coupled analytical models to evaluate design points. This means that, e.g., the problem of incorporating a system model generator and external simulation tools has not addressed.

Using pre-compiled and ready-to-use search algorithms available at [19] of the PISA framework, Madsen *et al.* [20] have created a multi-objective DSE framework. Different mapping alternatives can be evaluated (by means of analytical models) for a fixed or flexible platform during the exploration process. Moreover, the chosen representation formats for internal interfaces in [20] are problem specific, which means that they should be modified for each particular problem. In our case, these are dynamically and automatically updated according to an input constraints file. Finally, the kind of platforms generated in [20] is limited to hierarchical bus topologies, while our approach is not restricted to analyse a particular architecture.

To conclude this section, we summarize our contributions as follows. First, we propose a generic infrastructure for system-level MP-SoC design space exploration, which is capable of supporting different search strategies and existing system-level simulation tools in a single environment. As a result, the potentials for reuse of the framework are significantly increased since each DSE experiment can be performed without the need of preparing experiment-customized scripts, but it only requires a simple change of the user's input constraint values. Second, we have implemented and integrated a new approach in NASA to gradually and automatically generate simulatable system models that are used for obtaining system metrics to evaluate design decisions. Thus, the entire DSE process (composed of searching, system models generation and design point evaluation) is performed in an automatic and systematic fashion, thereby improving design productivity and decreasing the designer's efforts. Third, NASA deploys a novel *dimension-oriented* DSE approach in which the design space is explicitly separated into dimensions, which could represent design decisions that are orthogonal to each other such as mapping, architectural components, and platform. Thus, the designer can choose to simultaneously explore all dimensions, or to fix one or more of these dimensions (e.g., a fixed platform) and to focus the exploration within one or two dimensions (e.g., mapping exploration only). To this end, designers are allowed to configure the appropriate number of, possibly different, search algorithms to simultaneously co-explore the various design space dimensions.

III. THE NASA FRAMEWORK

Four key properties have been taken into account in the design of NASA:

Modularity. NASA is a highly modular framework in which the interaction between its modules is established by well-defined interfaces, allowing each module to act like an independent black box inside the framework. As a result, different modules can be easily integrated in a plug-and-play fashion.

Flexibility. A key element in NASA is its *hierarchical* DSE approach in which the design space is explicitly separated into different dimensions. As will be explained in more details later, three dimensions are currently distinguished in NASA: platform, architectural component, and mapping exploration. Thus, the designer can choose to simultaneously explore at all of these levels, or to fix one or more of these levels (e.g., a fixed platform) and to focus the exploration on one or two levels (e.g., mapping exploration only).

Re-usability. For a given set of user constraints, NASA is capable of exploring the design space in a systematic way, automatically generating the system models of selected design points that need to be evaluated by the system-level simulator. Hence, there is no need to prepare experiment-customized scripts. To perform a new DSE experiment, a designer only requires changing the constraint values.

Extensibility. Due to the modularity and the well-defined interfaces, new modules or functionalities can easily be plugged into the NASA framework. These new modules could, for instance, handle additional dimensions in the design space without needing to modify other modules.

The infrastructure of NASA is shown in Fig. 2. Essentially, six main modules can be distinguished in the framework: the Search module, Feasibility Checker, Architectural Platform Generator, Translator, Simulator and Evaluator. Subsequently,

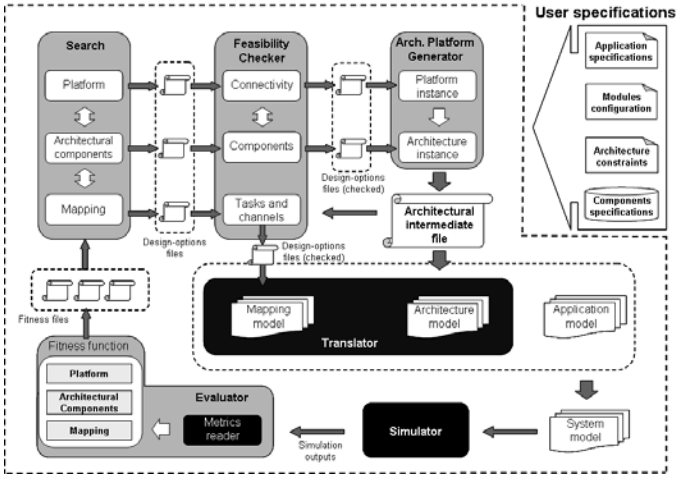


Fig. 2. The NASA infrastructure.

the different interfaces used by NASA as well as the functionality of each module are discussed, emphasizing on the implementation details of the two most important modules: Search module and Architectural Platform Generator.

A. Interfaces

Three kinds of interfaces are used in NASA: the *architectural intermediate file* is used for communication between the Architectural Platform Generator and Translator, the *fitness file* links the Evaluator with the Search module, and the *design-options file* is used in all sub-modules of both the Search module and the Feasibility Checker. Note that these files are dynamically and automatically created (and updated) according to the user input files.

In our approach, both the design-options and fitness files share the same XML-based format, in which *design decisions are encoded in strings*. Moreover, each explored dimension uses a separate design-options and fitness file. For example, in the 3-level design space exploration shown in Fig. 3, the platform dimension uses a design-options file to describe design decisions about the topology, network type(s) and the connectivity properties for the rest of architectural elements of a design point; the architectural components dimension uses its corresponding design options file to specify the type information of different components, while the decisions about the mapping of an application onto the different processing and storage elements are described in a third design-options file. If the designer decides to use less than one search algorithm per dimension, then adapter modules will automatically translate the input and output of the Search module to match the one design-option file per search algorithm interface. Note that the number of strings contained in any design-options file is equal to the number of design points explored by the Search module in each iteration, as will also be explained in Section III-B. Examples of design-decision strings are shown in Fig. 3, for three (Fig. 3(a)) and one (Fig. 3(b)) search algorithms (SA) in the Search module.

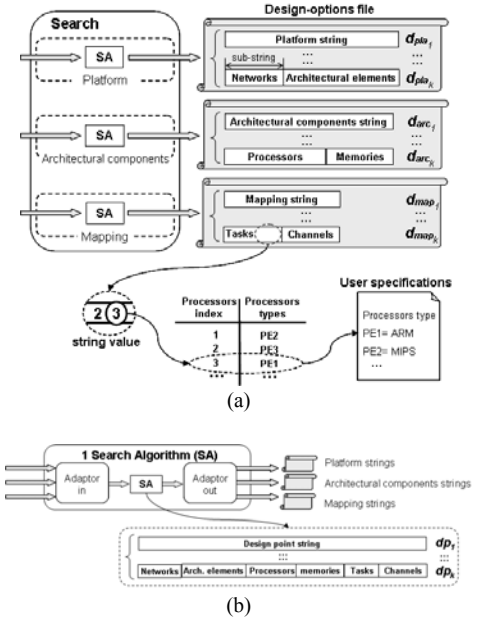


Fig. 3. Search Algorithms (SA) and search strings in NASA.

The length of a string description for each dimension may vary. Using the example shown in Fig. 3, it is evident that the length of the string describing the mapping depends on the number of tasks and communication channels in the application. Similarly, the length of the string describing the architecture instance is dependent on the number of processing elements (PEs) and storage elements (SEs) in the platform.

Finally, the values inside the design-decision strings do not hard-code absolute values but are indirections to table entries (also illustrated in Fig. 3(a)). This means that, for example, in the case of the mapping dimension, the string elements do not directly hard-code the PEs (including their exact type) onto which application tasks are mapped. Instead, the string elements point to entries in a PEs table. Hence, this allows the designer to, e.g., change the type of PE or add a new type without the need to adapt any module implementation. Clearly, this makes the approach more re-usable and extensible.

The last important interface in NASA is the architectural intermediate file. It describes the architectural platform design of each design point in a single file and, as will be explained in more detail later, it is gradually constructed using the platform and architectural components strings. The architectural intermediate file is used by the Translator to generate an architecture model of the design point in question. Moreover, it is also used to check the mapping feasibility. Note that platforms are not fixed entities in NASA but are often also part of the exploration. Therefore, the Feasibility Checker requires, e.g., connectivity information specifying which and how PEs are connected, and which SEs are shared by which PEs. This information is needed to detect and repair infeasible mappings, as will be explained in Section III-C.

B. Search module

This module performs the actual search through the design space, *iteratively* pinpointing (a set of) design points that need to be evaluated by means of system-level simulation. As mentioned before, NASA applies a dimension-oriented design space exploration approach. This way, each dimension can be co-explored simultaneously using a single search algorithm, or using multiple and possibly different search algorithms for the various dimensions. In this context, co-exploration means that, in spite of using one search algorithm per dimension, we do *not* perform the system-level design space exploration as multiple independent explorations, but instead, the results from *all* dimensions are simultaneously taken into account.

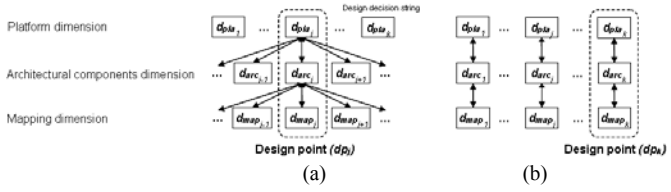


Fig. 4. Pyramidal versus one-to-one technique to link design decisions in a single design point.

This is, a design point *dp* can be expressed by linking *k* available design decision values $\{d_{pla}, d_{arc}, d_{map}\}$ corresponding to each of *k* design space dimensions, where d_{map} represents a design decision in the mapping dimension, while d_{arc} and d_{pla} express a particular design decision for the architectural component and the platform dimensions, respectively.

If multiple search algorithms are used to explore the design space, then there are many ways of linking the design decisions of each dimension to form a design point specification, as depicted in Fig. 4. For example, using a pure one-to-one linking technique, as shown in Fig. 4(b), each design decision in each dimension is linked to only one design decision in the other dimensions. Thus, the number of design points explored per iteration by the Search module is equal to the number of design decisions (or strings) contained in any design-options file, assuming that all design-options files have the same number of strings. Clearly, this significantly reduces the number of required evaluations because of the linear relationship between the number of design decisions and design points.

However, this approach may suffer from a possible convergence problem due to "under-exploration", i.e., discarding a design decision (e.g., a specific platform instance) too soon based on the results of a premature evaluation. For example, let $A=\{d^A_{pla}, d^A_{arc}, d^A_{map}\}$ and $B=\{d^B_{pla}, d^B_{arc}, d^B_{map}\}$ be two different design points. If it turns out after a single simulation that the fitness value of *A* is better than that of *B*, then this does not mean that platform d^A_{pla} or architectural components d^A_{arc} are always a better choice than d^B_{pla} and d^B_{arc}, but we do can affirm that the combination of design options $A=\{d^A_{pla}, d^A_{arc}, d^A_{map}\}$ is better than $B=\{d^B_{pla}, d^B_{arc}, d^B_{map}\}$. For instance, this latter does not guarantee that $\{d^A_{pla}, d^A_{arc}, d^A_{map}\}$ can provide a better fitness value than $\{d^B_{pla}, d^B_{arc}, d^C_{map}\}$, where d^C_{map} is another feasible mapping for *B*.

To address this under-exploration problem, we use a variant of one-to-one linking of design decisions. In this technique, unlike the pure one-to-one technique, only design decisions from the dimension of the lowest abstraction level (i.e., the mapping dimension in our case) are evaluated *and updated* during each search iteration. The search algorithms for the higher-level dimensions (i.e., the platform and architectural components dimensions) keep collecting the fitness values (for different mappings) without actually changing their design decisions during a specified number of iterations, referred to as the *collecting iterations* (δ). Only when the search has reached δ iterations, design decisions are updated, after which the process starts again. Obviously, the higher the abstraction level, the more design alternatives can be derived for a single design option (e.g., a multitude of architecture instances can be obtained from a single platform) and, consequently, the higher the value of δ should be. Note that the above mentioned feedback information, i.e., the fitness values, needed to guide this search through the design space, are iteratively provided by the Evaluator module, which will be explained in Section III-G.

C. Feasibility Checker

Because the search algorithms may try to assess infeasible design points during the DSE process, the main task of the Feasibility Checker is to detect infeasible design points and repair those design points if possible. In this context, a feasible design point is a system design that meets the user constraints both in terms of mapping and architectural implementation. If at least one of them is not satisfied, then the resulting design is classified as an infeasible one.

During this checking process, all sets-of-strings (or design-options files) are checked in a hierarchical fashion. This is, the platform string is first checked to determine whether or not the specified platform template (to be discussed in more detail in Section III-D) contains a valid topology and, e.g., whether it does not contain isolated islands of components. Next, the architectural components string is checked to determine whether or not the number and types of selected architectural components in the platform template comply with the constraints provided by the user. For example, if a design point deploys 4 ARM processors while the user has specified that only 2 ARM processors can be instantiated, then we have an infeasible design point. Finally, the mapping string is checked for infeasibility, e.g., when application tasks are mapped onto PEs that have not been allocated in the platform, or in the case there is no shared SE to map a logical communication channel between two tasks that have been assigned to different PEs. So, each design point is globally checked, i.e., taking all dimensions of the design point into account.

If an infeasible design point is detected, then different kinds of repair mechanisms can be applied, depending on the dimension where the problem occurs. Note that different repair techniques can also produce different feasible solutions from the same infeasible design decision. In our current implementation, we use heuristic minimum-distance repair techniques, which introduce a minimum number of modifications to an infeasible design string in order to obtain a feasible one. As a consequence, our repair techniques only have a minimal effect on the run-time of the framework. In the aforementioned infeasible mapping example (i.e., no reachable SE for two communicating tasks), only the communication channel of those two application tasks should be relocated into an reachable SE if a feasible mapping can be derived from such a repair. Although it is also possible to repair by mapping one of those two application tasks onto another available PE (or even both application tasks onto the same PE), this would require the resulting mapping to re-enter for a new mapping feasibility check as it may cause additional infeasibilities for other communication channels. In the worst case, this may even cause an infinite loop. The impact of these repair mechanisms on the number of explored feasible design points will be discussed in Section IV-B. Specifically, our

experimental results reveal that these repair techniques can warrant the repair of a high percentage of infeasible design points in the DSE experiments.

D. Architectural Platform Generator

The main mission of this module is to provide the architectural description for each design point by means of combining both feasible platform and architectural components information, which are contained in the strings of their respective design-option files. The resulting architectural description file is used later for (i) feasibility checking of mapping strings, and (ii) as input (to the Translator) to generate the architectural model. Thus, the Architectural Platform Generator can be considered as the first stage of the system model generation process.

An architectural description is created in two steps: platform or topological template generation and architecture instance generation. The basic building block of these descriptions is the so-called *Basic Topology Unit* (BTU). As shown in Fig. 5, the BTU is a *logical pattern* consisting of a network container (the gray component) and a variable number of element containers (the white blocks). These element containers are labelled inside each BTU and can, in a later stage, be instantiated as architectural components such as PEs and SEs. The number of element containers in a BTU depends on the user specifications, like the maximum number of PEs and SEs in a platform. Note that network containers cannot directly connect to each other, while element containers can connect to both element and network containers.

The BTU is labelled and replicated a number of times to form a meta-platform, which is used later in topological template generation. In principle, the meta-platform is used as a basis from which all feasible platform instance descriptions can be (gradually) derived and generated. The number of BTU replications in the meta-platform depends on the maximum number of network element (NE) and connections allowed among element containers, as specified by the user. The latter is referred to as connectivity, which defines for each element container both the available links and the directions (represented with numbered arrows in the top-left corner of Fig. 5). Thus, a BTU can be replicated through two or three directions and, as consequence, different kind of meta-platforms can be generated according to the user specifications. A 2D meta-platform generation process is shown in Fig. 5, although a 3D meta-platform can be also generated if the gray links of an element container (top-left corner of Fig. 5) are also used during this process. It should be noticed that the generation of the BTUs as well as the meta-platform is performed statically (but automatically) before the actual DSE process.

Driven by the exploration at platform level (in Search module), the meta-platform is used to generate topological template instances. To this end, the set of strings of feasible platforms is used to instantiate the topological templates from such a meta-platform: each string sets (for one design point)

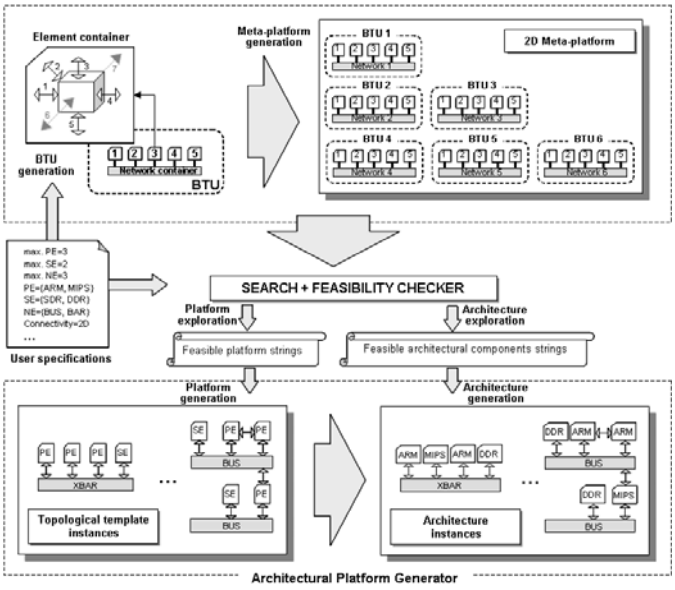


Fig. 5. Generation of topological templates and architecture instances.

the type(s) and number of networks in the platform. Moreover, the number of element containers in the platform as well as their connectivity properties are also determined. Finally, a type classification of the element containers is made. This latter means that for each allocated element container in the BTUs, it is indicated whether it contains a PE or a SE. Note that, as explained in Sections III-B and III-C, these platforms have been selected by the Search module and checked by Feasibility Checker. The latter repairs strings describing any infeasible topological templates such as, for example, isolated BTUs that do not connect to any other BTU, architectural elements with incorrect connectivity links, and other inconsistencies.

Finally, in order to obtain the complete specification of the architecture platform for each design point, the topological templates are further refined. In this process, which is driven by the exploration at architecture component level, the same topological template can be reused to derive different architecture templates. For this propose, the actual component types of the element containers in a template are added. In the example of Fig. 5, this means that, e.g., a PE allocated in an element container either becomes an ARM or MIPS processor, and the SEs either SDRAM or DDRAM. Evidently, all this information is also provided by the strings of feasible architectural components.

E. Translator

In order to integrate a system-level simulator in NASA, it is required that the simulator allows for explicitly describing the design points that need to be simulated using some kind of file format. Thus, a system model for each design alternative should be generated first. Such a system model, composed of an architecture model, an application model and a mapping model, can be provided by the Translator module in an automatic way. To this end, it uses as input the architectural intermediate file, the application specifications and the strings

that describe feasible mappings. Thus, the Translator can be considered as the second (and last) stage in the generation process of the simulatable system model.

There exist two relevant benefits in including this module in our framework. First, the Translator converts NASA's internal format of a design point to a file-based format that is specific for the target system-level simulator. Second, the integration of a new system-level simulator in NASA only requires the adaptation of the Translator module, i.e., tailoring the Translator for each different simulator, while all other modules remain unaffected. This is why two kinds of module colors can be identified in Fig. 2, simulators-dependent (black) and simulators-independent (gray) modules.

F. Simulator

At this moment, we have integrated a SystemC-based system-level simulation environment called CASSE [3] in NASA. Another system-level simulator, called Sesame [4], is in the process of being integrated. Both tools follow a Y-Chart methodology, covering application and architecture modelling, as well as mapping and analysis within a unified simulation environment.

For these simulators, the application model is described as a process network (Kahn Process Network) or as a Tasks-graph, where parallel tasks communicate with each other by means of unidirectional channels. Here, tasks (containing the application functionality) are often written in C/C++. On the other hand, the architectural model is specified as a modular composition of highly configurable predefined elements (provided by the tool libraries), including processing elements (PE), storage elements (SE) and network elements (NE). The number of elements of each type and their configuration (e.g., number and width of port, clock, memory size, network arbitration scheme, task scheduler policy, etc.) can also be properly configured in this description file. Finally, another description file is used by CASSE and Sesame to control the mapping of the application onto the architecture. Note that both tools ensure deadlock-free task mappings and scheduling for feasible design points. Obviously, all the required models and descriptions can directly be generated by a customized Translator module, as it was explained in Section III-E.

At this point, it is important to highlight a key property for the mentioned simulation tools. The system model file is read and parsed by CASSE and Sesame during elaboration time in order to properly configure the desired design point. Thus, changes in the files describing a design point do not require any recompilation effort. Evidently, this allows for evaluating design alternatives during the exploration process in a completely automatic way, without any human intervention. To give an example, the simulators are highly parametrized in terms of performance values for the different architectural processing and communication elements. These parameter values are explicitly stored in the system model file. This allows for, for example, quickly evaluating different hardware/software partitionings by simple manipulation of the performance values for selected processing elements.

Since the implementation of these tools is behind the scope of this paper, the interested reader is referred to [2] for an overview of existing system-level simulators, and to [3] [4] for more detailed information about CASSE and Sesame.

G. Evaluator

During simulations, quantitative information about the system execution (e.g., data about performance, cost/area, and power consumption) can be gathered and dumped into files for later inspection. All these metrics can be used in system-level DSE to find a set of Pareto optimal design points, which then yields a multi-objective optimization problem.

The essence of the Evaluator module is to provide this feedback about the quality of a set of evaluated design points to the Search module, influencing the search decisions taken in the exploration process.

Separating the Evaluator from the Search module again provides flexibility and enhanced reusability of the components in NASA. It allows for easily changing the optimization objectives or the function that quantifies the quality of a design point without affecting the other components. Such a function is typically referred to as the *fitness function*. The Evaluator also provides the flexibility to, e.g., use a single fitness function for all search algorithms in the Search Module, or to deploy a different, and possibly tailored, fitness function per search algorithm.

However, when multiple search algorithms and fitness functions are used together, these should be defined in a coherent way with respect to each other in order to avoid conflicting fitness functions and safeguard convergence. This is because there exists a tight connection between the different search algorithms and their respective fitness functions. This connection should be made explicit. In our current implementation, these relations can be defined by a set of hierarchical fitness functions, which can be used with a variant of the one-to-one linking technique (already explained in Section III-B) to address the under-exploration problem in hierarchical design space explorations with multiple search algorithms. Formally, these hierarchical fitness functions are formulated as follows:

$$y_{L_i} = f_L(x_1, x_2, ..., x_k); \quad \forall i = 1..I$$

$$\begin{cases} y_{j_i} = f_j(x_1, x_2, ..., x_k) = \sum_{q=1}^{\delta_j} y_{L_q}; & \forall i = 1, \delta_j, 2\delta_j..I \quad and \quad \forall j \neq L \\ \delta_z > \delta_w; & \forall z, w = 1..\beta \quad and \quad z \supset w \end{cases}$$

where y_{L_i} is the fitness value of a design point of the lowest-level dimension (the mapping dimension in our case) in the search iteration i, I is the total number of search iterations, x_k represents the value of the metric k used in the fitness function f, y_{j_i} is the fitness value of a design point in any dimension other than the lowest one, and δ_j represents the collecting iterations for the individuals of dimension j. Moreover, for a given range of dimensions β, the number of the search iterations needed for collecting fitness information for dimension z (e.g., platform) should be bigger than the number of iterations needed for dimension w (e.g., architecture) if z has a higher abstraction level than w (denoted by the $\supset$

operator).

IV. Experimental results

In this section, we present a number of DSE experiments to demonstrate the capabilities of the NASA framework and to illustrate its distinct aspects.

A. NASA configurations for experiments and parameter settings

The first set of experiments aims at comparing the more traditional approach of using a single search algorithm for all design space dimensions to our dimension-oriented approach (using a separate search algorithm per dimension, i.e., 3 search algorithms (SAs) in total). To properly evaluate and compare the quality of the DSE between different search strategies, we can define three criteria:

1) *Diversity*. A large number of different design points should be explored in each DSE experiment to cover a wide range of design decisions for each dimension.
2) *Convergence*. The strategies should provide approximations to global (or near-to) optimal solutions without being trapped at local optima.
3) *Coverage*. The explored design points should be well-distributed in the design space for a complete view of the trade-off curve or landscape of the design space as well as catching boundary values.

Assessing the quality of the exploration is not equal to assessing the "quality" of the obtained design points. However, an exploration meeting all three criteria should lead to good design points in terms of fitness values (such as good performance). Different parameter settings for the experiments, i.e., different NASA configurations, lead to different results in DSE quality and in the fitness values obtained. Next, we introduce several different NASA configurations together with the results obtained.

In Table I, the most important user specifications and parameters for the first set of experiments are listed. The studied MP-SoCs may consist of up to 6 PEs of the types ARM, PowerPC (PPC), or MIPS, up to 3 SEs of either single (SDR) or double data-rate (DDR) type, and up to 4 NEs of three types (bus, fully connected, or a customized network consisting of a bus and point-to-point links). The application that is mapped onto the MP-SoC is an optimized version of the computer vision algorithm presented in [8]. Basically, this visual tracking algorithm has a real-time requirement (25 frames/s), and applies a correlation or block matching technique to continuously track a specific target in the incoming image frames. The block or pattern size and frames size used in our experiments are 24×24 and 320×240, respectively.

TABLE I
PARAMETER SETTINGS IN OUR EXPERIMENTS

Parameter	Nr.	Types	Values
PE	≤ 6	3	ARM, PPC, MIPS
SE	≤ 3	2	DDR, SDR
NE	≤ 4	3	Bus, Fully-connected, Customized-network
App. Tasks	7	-	-
App. Channels	12	-	-
Dimensions (β)	3	-	Platform, architectural components and mapping
Search algs. (SA)	1 or 3	1	Genetic algorithms
GA Selection (S)	1	1	Proportional with elitism
GA Crossover (C)	1	2	1-point and 2-point
C probability *(pc)*	5	-	[0.1,0.3,0.5,0.8,1.0]
GA Mutation (M)	1	2	Simultaneous (M=1) and Independent (M=6)
M probability *(pm)*	5	-	[0.1,0.3,0.5,0.8,1.0]
Collecting iterations (δ_{arc})	1	-	2, architectural components dimension
Collecting iterations (δ_{pla})	1	-	4, platform dimension
Search iterations (I)	41	-	-
Population size (N)	10	-	Nr. of individuals per iteration
Simulation tool	1	-	CASSE

1) Search Algorithms settings (SA)

With respect to the search algorithm(s) we use for exploration, a multitude of them can be used (via a simple plug-in mechanism): from exhaustive search or random search, to heuristic search methods. We focus on implementations based on genetic algorithms (GAs), since GA-based DSE has been widely studied in the domain of system-level design [6], [7], [11], [17], [20], and it has been demonstrated to yield good results. In this case, we use a proprietary implementation of the GAs, but any existing GA such as SPEA2 or NSGA-II [11] could also have been used.

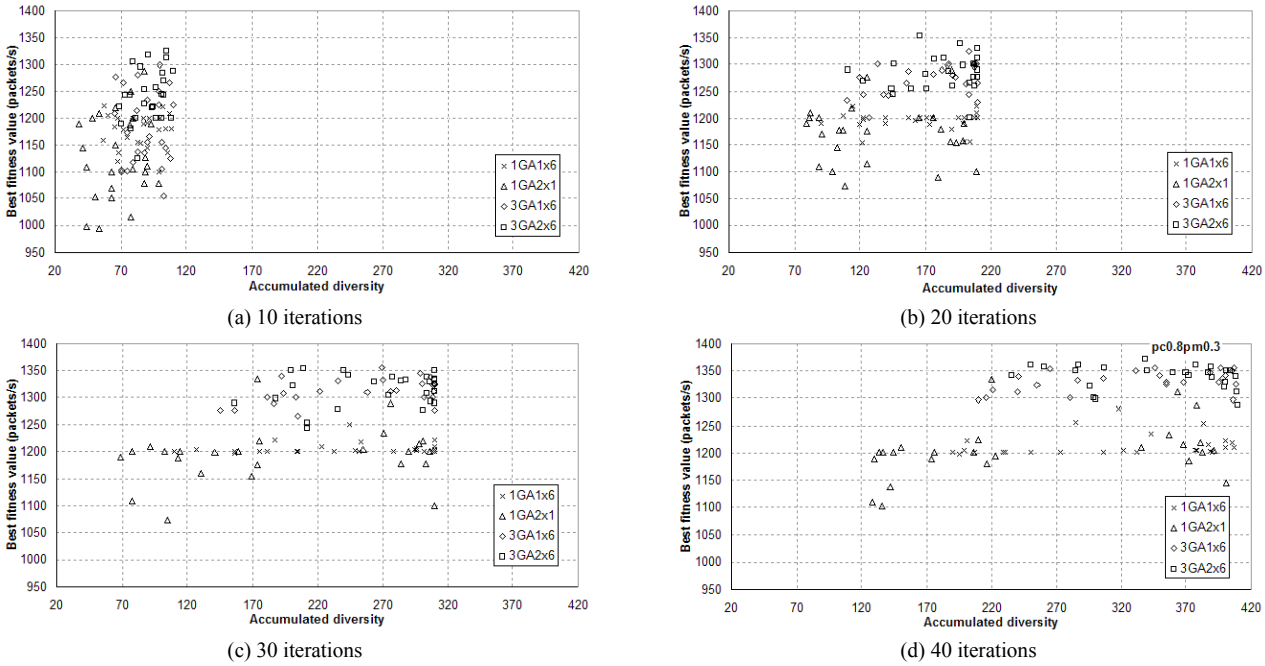
2) Crossover and mutation type settings

The crossover and mutation operators in our GAs are performed at the granularity of entire sub-strings (see Fig. 3) in a string that describes the topological platform, architectural components or mapping. These operators are applied according to their associated probabilities (*pc*: probability of crossover, and *pm*: probability of mutation). Further, the GA can perform either a 1-point or a 2-point crossover, and supports two types of mutation. In "simultaneous" mutation (M=1), a single random position is simultaneously changed in every sub-string. In "independent" mutation (M=6), the mutation probability is used for each of the six sub-strings to determine whether it is mutated or not. In the case that three GAs are used for exploration, different and customized values for the probabilities *pc* and *pm* can be used within each GA.

If all the GA parameters in Table I are taken into account, a large number of experimental combinations can be performed. From this set of experiments, we present a selection of four NASA configurations. The nomenclature used to denote these configurations is "SAgaCxM", where the meaning of each capital letter is defined in Table I. For example, "3ga1x6" refers to the configuration with 3 GAs that simultaneously explore the platform, architectural components and mapping dimensions, a 1-point crossover, and "independent" mutation (M=6).

3) Group of experiments and run-times per simulation

All possible combinations of the *pc* and *pm* values (as listed



(a) 10 iterations (b) 20 iterations

(c) 30 iterations (d) 40 iterations

Fig. 6. DSE results for four NASA configurations.

in Table I) have been evaluated. This results in 25 groups of experiments (5 *pc* probabilities × 5 *pm* probabilities) for each of the four mentioned NASA configurations. Note that, independent of the number of search algorithms used in these experiments (i.e., 1 GA or 3 GAs), a maximum of 205,000 simulations (25 groups of experiments × 20 different initial populations × 41 iterations × 10 individuals per iteration) have been performed for each NASA configuration. The CASSE tool – which dominates the run-time of our DSE experiments – requires on average 40 seconds to simulate a single design point on a PC with a Pentium IV processor at 1.6 GHz and 2 GB main memory, running Linux.

4) Fitness functions for evaluation and optimization

In order to simplify the graphic representation of the results and the explanation of the examples in this section, without loss of generality, the fitness value in our experiments only takes a single system metric into account, namely performance. We would like to stress, however, that multi-objective optimization can also be perfectly addressed with NASA.

B. DSE behaviour and sensitivity to various parameter settings

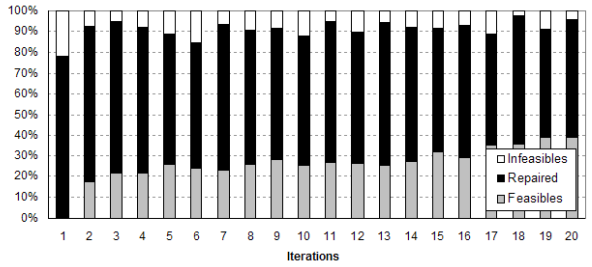
1) Impact of the number of search algorithms on DSE quality

The results of the above experiments are shown in the four scatter-plots of Fig. 6, which compare the behaviour of DSE experiments based on a single and multiple GA approach after 10, 20, 30 and 40 iterations, respectively. Each scatter-plot shows the average total of *different* explored design points (i.e., accumulated diversity) on the x-axis and the average of the best fitness values, in terms of processed data packets/s, on the y-axis for each of the experiments.

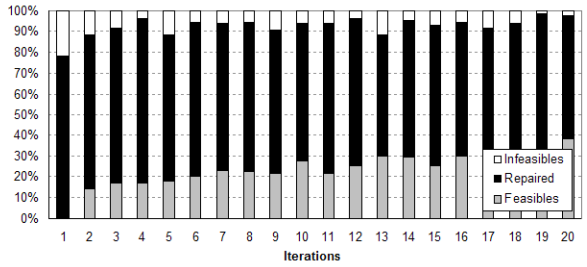
If the input arrival frame rate is 1450 packets/s and a minimum of 1250 packets/s has to be processed to satisfy the minimum real-time requirements of the studied application (which is equivalent to processing 25 frames/s), then using a 3 GA-based search approach in NASA not only provides the design alternatives with the best fitness values (in the upper right corner for each scatter-plot of Fig. 6) but the accumulated diversity of the explored design points is also largest. Notice that exploring the same design space with a traditional, single GA approach, optimal and near-to-optimal architectures are less often found. This is mainly due to a smaller accumulated diversity of explored design points. Moreover, it can also be seen that the larger the number of iterations, the larger the gap between traditional single GA-based DSE and our 3GA-based DSE in terms of accumulated diversity and best design points reached.

From this, it appears that the multiple GA search has a positive impact on the DSE quality. In other words, while all parameter settings affect the quality criteria of the exploration performed, and consequently the best design points obtained, the multiple GA-based searching seems to be an important factor for achieving quality.

For a detailed comparison between both approaches (single GA and multiple GA search), the three proposed criteria – diversity, convergence and coverage – are separately analyzed in Fig. 8, Fig. 9 and Fig. 10. To this end, we have selected one group of experiments for each of the four mentioned configurations, where all configurations use the values *pc*=0.8 and *pm*=0.3. With these probabilities, the "3ga2x6" configuration finds the design point with the best overall fitness value after 40 iterations (see upper right corner of Fig. 6(d)).



(a) 1GA: average percentage of repair = 84.21%



(b) 3GA: average percentage of repair = 86.68%

Fig. 7. Average percentage of feasible, repaired and infeasible design points per iteration in our DSE experiments.

2) Impact of repair mechanisms on the efficiency of the DSE process

At this point, it is important to highlight that, although the explored design points shown in our experimental results include both feasible and infeasible design points, only feasible design solutions are evaluated by the CASSE tool. However, due to the repair techniques applied in the Feasibility Checker module, most of the infeasible design points can be detected and converted to feasible ones. As a consequence, the feasible alternatives actually evaluated represent an important percentage of the total number of explored design points. This can be illustrated in Fig. 7(a) and Fig. 7(b), which depict the average percentage of feasible, repaired and infeasible design points per iteration for the DSE experiments based on both approaches (single and multiple GA search). Note that the gray part of each bar (in Fig. 7) represents feasible design points without any repair, the dark part refers to repaired design points (i.e., infeasible design points repaired by the Feasibility Checker module and converted to feasible ones), and the white part indicates the infeasible design points that cannot be repaired by our heuristic minimum-distance repair techniques.

From these data, it can be seen that our repair techniques can repair more than the 84 percentage of detected infeasible design points in each iteration, and as a result, more than the 91 percentage of explored design points can be actually evaluated by the CASSE tool. Thus, it seems that the repair mechanisms significantly affect and improve the efficiency of the DSE experiments.

3) Convergence rate and number of iterations

The convergence is illustrated in Fig. 8, where the horizontal axis indicates the number of explored design points (and iterations) and the vertical axis represents the fitness values in terms of processed data packets/s. Investigating these data, it can be seen that 1 GA-based experiments have a higher convergence rate (i.e., a steeper slope) than 3 GA-based experiments in the first iterations. This phenomenon is the implicit effect of using the hierarchical fitness functions (explained in Section III-G) and the variant of the one-to-one individual linking technique (presented in Section III-B) in 3 GA-based experiments.

However, when the number of iterations increases, 3 GA-based experiments do not only gradually and progressively reach higher fitness values than 1 GA-based experiments, but they can also ensure that most of the individuals in each iteration satisfy the real-time restriction (1250 packets/s). In the 1 GA-based experiments, on the other hand, mostly design solutions with fitness values lower than the real-time restriction are reached. Moreover, the 1 GA-based experiment hardly improves or provides better design alternatives with the evolution of iterations. The latter could indicate that the GA is trapped in a local optimum, which occurs when design points explored in each experiment are not sufficiently different or well-distributed (i.e., partial coverage) to properly capture the design space in its entirety (e.g., covering only partially or some regions of the design space), caused by an insufficient variety of new individuals introduced in each iteration (i.e., a low incremental diversity) that prevents the populations to escape from such local optima. These aspects can be demonstrated in both Fig. 9 and Fig. 10.

a) Diversity and the search approach

Each curve in Fig. 9 represents the percentage of new and different design points introduced in each iteration that have not been explored in any of previous iterations, i.e., the

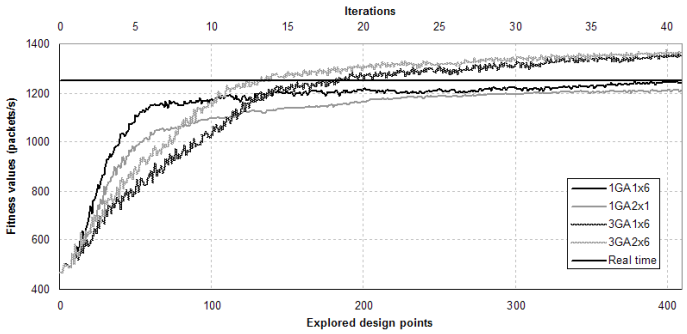


Fig. 8. Average fitness values per iterations.

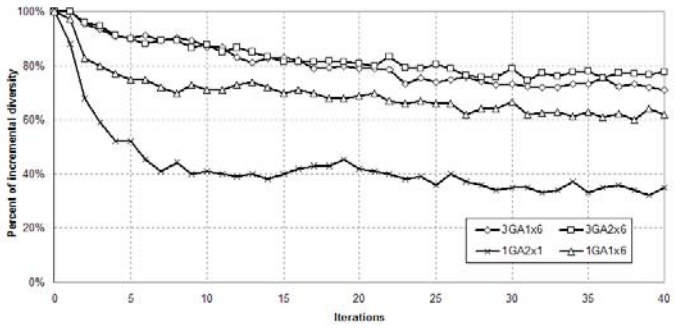
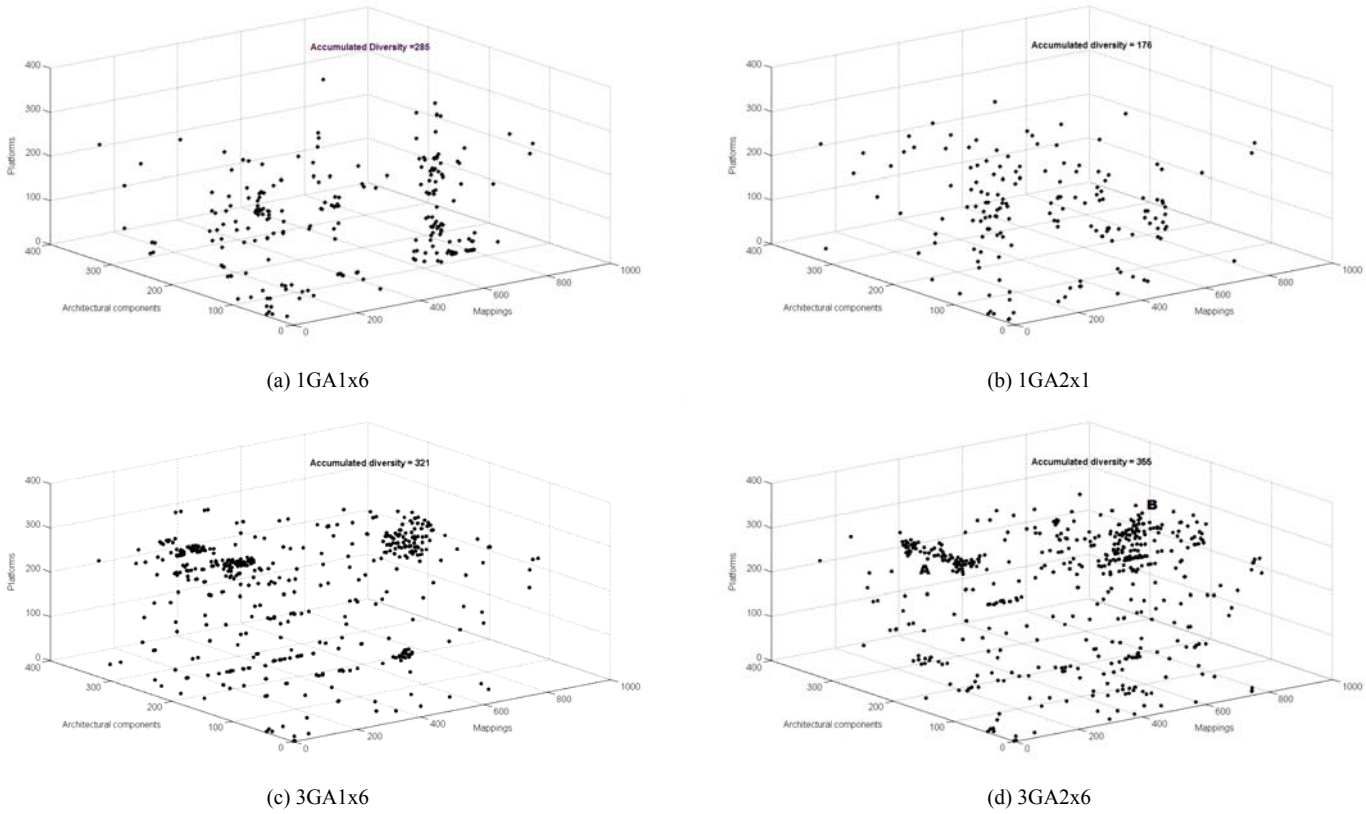


Fig. 9. Incremental diversity per iterations.



(a) 1GA1x6

(b) 1GA2x1

(c) 3GA1x6

(d) 3GA2x6

Fig. 10. Explored design points by each selected NASA configuration.

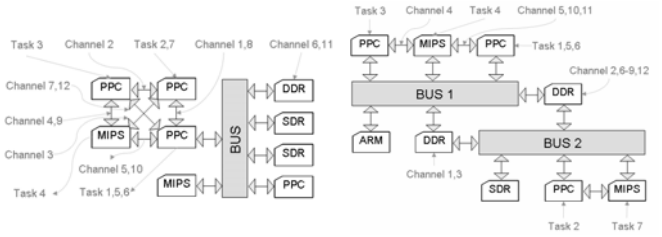
incremental diversity per iteration. These results highlight that 3 GA-based experiments clearly yield a higher incremental diversity per iteration than 1 GA-based experiments, and especially in the case of 1 GA with "simultaneous" mutation (M=1). A direct consequence of the latter result thus explains the resulting gap of the accumulated diversity between both approaches, as already shown in Fig. 6.

4) Coverage, design points concentrations and local optima

All design points explored by each of the selected group of experiments (corresponding to the four mentioned NASA configurations) are separately shown in Fig. 10, where each axis represents one design space dimension in our 3D design space. Moreover, for a fair comparison, each axis in Fig. 10 contains all *ordered* design-decision instance numbers (i.e., the canonical representations of the strings for the platform, architectural components and mappings dimensions) explored together by these four groups of experiments. It can also be seen in Fig. 10 that the design points explored in the 3 GA-based experiments are scattered over almost the whole design space (high coverage) and are characterized by a high accumulated diversity. The design solutions reached by the 1 GA-based experiments, on the other hand, have a lower accumulated diversity and are often concentrated in a single region of the explored design space (lower coverage). This indicates that the searching process is converging toward an optimum, and in this last case, toward a local optimum as already shown in Fig. 8.

5) The need for refinement

It should be noted that *design points concentration* – a visual indicator of the convergence process – can also be observed in the 3 GA-based experiments. But, unlike the 1 GA-based experiments, the convergence is toward a global optimum or toward a few optimal points. The existence of several "optimal points" can be illustrated for two NASA configurations based on multiple GAs shown in Fig. 10, where more than one design points concentration (or convergence) area can be identified. This is correct since different alternatives can often satisfy a given set of user restrictions. To illustrate the above, two design points (A and B) have been marked in Fig. 10(d), and their respective architectures and mappings are shown in Fig. 11. In this case, although both solutions (corresponding to each of the design points convergence regions) have similar performance (A achieving 1371 packets/s and B 1355 packets/s), their underlying platform architectures are however quite different. Moreover, they are also over-dimensioned in the sense that not all resources are actually used by the application. Therefore, in this case, designers can perform an additional optimization process in terms of architectural components and/or in terms of mapping. Such *refined* optimization can be performed in a next and more detailed phase of exploration experiments where, e.g., the platform is fixed and only the architectural components and mapping dimensions are



(a) Design point A: 1371 packets/s (b) Design point B: 1355 packets/s

Fig. 11. Examples of design points found by 3GA2x6 DSE with pc=0.8, pm=0.3 after 40 iterations.

explored more rigorously. Alternatively, additional objectives or fitness functions (such as the cost of designs) can also be taken into account in the optimization process. In the next section, we will further investigate refinement by fixing one dimension (platforms) and then conducting DSE in just a two dimensional space (architecture components and mapping).

C. Hierarchical refinement and analysis of 2D-DSE with NASA

Our second set of experiments presented in this paper aims at demonstrating NASA's flexibility and capacity to perform the aforementioned refinement process. To this end, this set of experiments is focused on 2D design space explorations, where design decisions about mapping and architectural components are explored for a particular platform template, i.e., the platform dimension is fixed and no search algorithm is used in this dimension. Obviously, in order to model a specific platform template, designers should properly configure the platform string values for the number and types of element containers in each instantiated BTU as well as their connections with each other.

1) Fixed target platform, variable architectural components and mapping

The selected target platform template and available type values for PE and SE are depicted in Fig. 12 and Table II. This platform template can provide architecture models based on two AMBA buses connecting up to six processing elements and three storage elements. The execution time of the visual tracking application's tasks has been estimated using an instruction set simulator [21] for the ARM processor, while we assume that the hardware dedicated block (which executes block matching operations of the target application) has a ×10 speedup factor with respect to the SW implementation. Note that in this case study, plenty of platforms could have been analyzed in our refinement experiment. However, we believe that we selected a realistic MP-SoC platform template, consisting of several homogeneous processors completed with a few coprocessors or hardware dedicated blocks in a bus-based architecture, rather than an MP-SoC based on various processing and network element types having different computational and communication characteristics.

2) 2D NASA configuration for DSE

Four NASA configurations have been selected in this second set of experiments: 1GA+1Random, 1GA+1ga, 1GA+1GA and a heuristic algorithm from [22]. The used parameter settings of the genetic algorithms are illustrated in Table II. For example, 1GA+1GA (or 1GA+1ga) refers to two identical (or different) genetic algorithms are used in the architectural components and mapping dimensions, respectively. On the other hand, in the cases of 1GA+1Random, a GA explores different architecture instances by varying the type of SEs as well as the location of the hardware dedicated block in different PE containers of the platform template (since the rest of PE share the same processor type), while a random search algorithm explores different functionality distributions onto system resources in a random fashion. It should be noted that although not included in this set of experiments, an extensive number of combinations of different search algorithms as well as GA parameters could have been used (as already shown in Fig. 6 for our first set of experiments). Therefore, the four selected configurations only represent a few samples of NASA's capacity and flexibility.

3) Heuristic-based mapping algorithm

For convenience, a brief overview of the heuristic algorithm is introduced before presenting our results and performing comparisons. This heuristic algorithm [22] uses as input a real-time application, the equivalent deadline (in number of cycles) of the real-time constraint, a MP-SoC template and a list of available PEs and SEs for such template, and estimates analytically the best MP-SoC instance (varying the location and combination of PEs and SEs) for the target real-time application, i.e., achieving real-time requirements as well as optimizing processor utilization, inter-processors traffic load, and processor load balancing. The heuristic approach consists of three phases. First, a real-time application is modelled as a tasks-graph, after which the algorithm schedules the tasks on a set of virtual processors (VPs) or logical clusters taking into account the real-time deadline and assuming that: (i) each physical PE can only hold a single VP in the further steps and, (ii) the set of PEs works in a pipeline fashion. Second, all possible MP-SoC instances are exhaustively generated, i.e., all

TABLE II
SEARCH MODULE AND TARGET ARCHITECTURE PARAMETER SETTINGS

GA	Selection (S)		Proportional with etilism
	Crossover (C)		1-point, pc = 0.5
	Mutation (M)		Independent (M=6), pm = 0.5
ga	Selection (S)		Tournament without etilism
	Crossover (C)		2-point, pc = 0.8
	Mutation (M)		Simultaneous (M=1), pm= 0.3
Collecting iterations (δ_{arc})		2	architectural components dimension
Search iterations (I)		21	-
Population size (N)		10	Nr. of individuals per iteration
PE		≤ 6	ARM and hardware dedicated block
SE		≤ 3	DDR and SDR

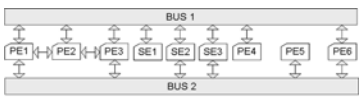


Fig. 12. Target platform template for the second set of experiments.

combinations of type and locations of PEs and SEs on the target template. In the last step, the algorithm uses a set of analytical expressions (which take into account variables such as resource connectivity, remaining processing and storage capacities, latency parameters associated to communication protocols of each component, etc.) to evaluate different alternatives. Subsequently, it outputs the best logical clusters mapping onto MP-SoC instance that satisfies the real-time constraint. Interested readers are referred to [22] for more detailed information.

It should be noted that although this heuristic algorithm can quickly and simultaneously explore both architecture candidates and feasible mappings by means of a static performance estimation technique, the output (or the selected design point) still needs to be carefully examined in a system-level simulator. This is because of *non-deterministic or non-linear system functions* (e.g., the bus arbitration delay due to simultaneous access requests by multiple PEs) are not taken into account during its estimation process, thereby making an accurate performance evaluation difficult without a simulation. To this end, the output of this algorithm is adapted to the string format required by NASA's Translator, which then produces the corresponding architectural and mapping model to be simulated in the CASSE tool.

4) Results and discussions

The results of the above four configurations are shown in Fig. 13. The dark bar represents the fitness value obtained in simulation with the design point selected by the mentioned heuristic algorithm. The curves show for the rest of the configurations (i.e., 1GAx1GA, 1GAx1ga and 1GAx1Random) the average fitness values reached by all individuals in each of the twenty iterations. From these results, it can be seen that 1GAx1Random can only sporadically reach a few design points that satisfy the real-time constraints. Moreover, it clearly cannot ensure convergence toward any global optimum. On the other hand, both the heuristic algorithm and the experiments based on two genetic algorithms can provide solutions that satisfy the input constraints. To this end, the heuristic algorithm only requires to simulate a single system model (or individual), while 1GAx1ga and 1GAx1GA need to simulate an average number of 40 and 60 individuals respectively (since 1GAx1ga has a higher convergence rate than 1GAx1GA in the first four iterations) before reaching the first individual that satisfies the real-time constraint. This might suggest that the multiple GAs strategies are not as efficient as this heuristic algorithm in terms of simulation time dedicated to DSE. However, our multiple GAs-based co-exploration approach presents three important benefits with respect to the heuristic algorithm: (i) both 1GAx1GA and 1GAx1ga can converge toward design points with higher fitness values, (ii) the studied heuristic algorithm cannot perform multi-objective optimization where the result is a Pareto front of solutions, and (iii) GAs provide not only information about the best solutions but also about all other explored design points in each experiment (rather than a single design point outputted by the heuristic algorithm). This is a key element for better understanding the studied design space, i.e., the more design points are provided to the designer, the more information can be extracted from the explored design space, and therefore, it will allow designers to more easily compare the architectural characteristics of the evaluated design points. That is, it can be very useful for a designer to distinguish the architectural similarities of the design alternatives featuring good fitness values.

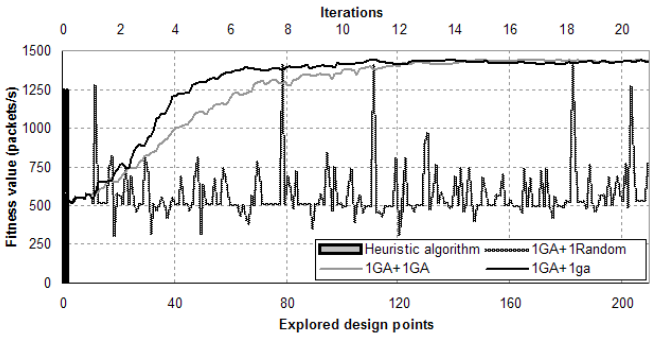


Fig. 13. Comparative results obtained in the second set of DSE experiments.

This last aspect can be illustrated in Fig. 14, which shows an example of typical NASA output after each DSE experiment. The set of simulated design points, corresponding to a 1GAx1ga experiment in this case, can form a surface that approximates the landscape of the explored design space. A 2D view of the resulting surface is shown for this experiment since it is based on 2D exploration, i.e., the x axes and y axes of the 2D view contain the explored instance numbers of the mapping and architectural components dimensions, having fixed the platform as mentioned. So, for example, 120 different mappings have been explored in this example. Note that the fitness value associated to each design point is color coded, ranging from red (high fitness value) to blue (low fitness value). Therefore, even when exploring a relatively small number of design points, the distribution of design points in the surface can clearly indicate the location of convergence region(s), while dark red areas can provide a good insight of where the sweet spots (design points with higher fitness values) in the design space are located.

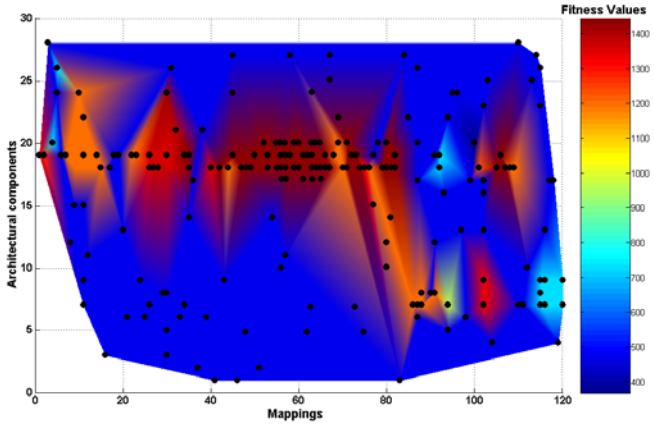


Fig. 14. Example of NASA output.

Moreover, designers can select any design point of the surface, and examine information about that design point such as the parameter values for the mapping and/or architectural components dimensions. Finally, it should be stressed that all these experiments presented in this paper have been performed in a fully automatic fashion, only providing parameter settings and constraints such as those shown in Table I and Table II.

V. CONCLUSIONS

In this paper, we addressed the lack of a generic, flexible, and re-usable infrastructure to facilitate and support system-level MP-SoC design space exploration (DSE) experiments. To this end, we have presented a system-level MP-SoC DSE support infrastructure, called NASA. This highly modular framework uses well-defined interfaces to easily integrate different system-level simulation tools as well as different combinations of search strategies in a simple plug-and-play fashion. Moreover, we described NASA's dimension-oriented DSE approach, allowing designers to configure the appropriate number of, possibly different and tailored, search algorithms to simultaneously co-explore the various design space dimensions. The result is a flexible and re-usable framework for the systematic exploration of the multi-dimensional MP-SoC design space, starting from just a set of relatively simple user specifications.

Our experimental results indicate that, compared to the more traditional approach of using a single search algorithm for all dimensions, the multi-dimensional co-exploration seems to be able to find better design points and ensure the convergence toward global optima. Furthermore, the multi-dimensional co-exploration has a higher diversity and coverage of design alternatives, producing higher quality DSE results. Finally, we have also illustrated NASA's capability and flexibility to integrate different kinds of search algorithms in DSE experiments. As future work, we plan to integrate a more extensive set of search algorithms into NASA, e.g., through the integration of the PISA optimization framework [18], [19], as well as to perform additional deployment case studies of NASA such as multi-objective optimization problems introducing other fitness and cost functions.

REFERENCES

[1] G. Martin, "Overview of the MPSoC design challenge", in *Proc. of Design Automation Conference (DAC'06)*, Jul. 2006.

[2] M. Gries, "Methods for evaluating and covering the design space during early design development", *Integration, the VLSI Journal*, vol. 38, no. 2, pp. 131-183, Dec. 2004.

[3] V. Reyes, T. Bautista, G. Marrero, P. P. Carballo and W. Kruijtzer, "CASSE: a system-Level modeling and design-space exploration tool for multiprocessor systems-on-chip", *Euromicro Symposium on Digital System Design (DSD'04)*, pp.476-483, 2004.

[4] C. Erbas, A. D. Pimentel, M. Thompson and S. Polstra, "A framework for system-level modeling and simulation of embedded systems architectures", *EURASIP Journal on Embedded Systems*, no. 1, pp. 2-2, Jan. 2007.

[5] C. Lee, S. Kim and S. Ha, "A systematic design space exploration of MPSoC based on synchronous data flow specification", *Journal of Signal Processing System*, vol. 58, no. 2, pp. 193-213, Feb. 2010.

[6] J. Teich, T. Blickle and L. Thiele, "An evolutionary approach to system-level synthesis", in *Proc. of the 5th Int. Workshop on Hardware/Software Co-Design (Codes/CASHE'97)*, pp. 167-171, 1997.

[7] M. Palesi and T. Givargis, "Multi-objective design space exploration using genetic algorithms", in *Proc. of Int. Symposium on Hardware/Software codesign (CODES'02)*, pp. 67-72, May. 2002.

[8] Z. J. Jia, T. Bautista, A. Núñez, C. Guerra and M. Hernandez, "Design space exploration and performance analysis or the modular design of CVS in a heterogeneous MPSoC", in *Proc. of the Conference on Reconfigurable Computing and FPGA (ReConFig 2008)*, pp. 193-198, Dec. 2008.

[9] K. Keutzer, S. Malik, A. Newton, J. Rabaey and A. Sangiovanni-Vincentelli, "System level design: orthogonalization of concerns and platform-based design", *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 12, pp. 1523-1543, Dec. 2000.

[10] B. Kienhuis, E. Deprettere, K. Vissers and P. van der Wolf, "An approach for quantitative analysis of application-specific dataflow architectures", in *Proc. of the IEEE Int. Conference on Application-Specific Systems, Architectures and Processors*, pp. 338, Jul. 1997.

[11] C. Erbas, S. Cerav-Erbas and A. D. Pimentel, "Multiobjective optimization and evolutionary algorithms for the application mapping problem in multiprocessor system-on-chip design", *IEEE Trans. Evolutionary Computation*, vol. 10, no. 3, pp. 358-374, 2006.

[12] S. Mohanty, V. K. Prasanna, S. Neema and J. Davis, "Rapid design space exploration of heterogeneous embedded systems using symbolic search and multi-granular simulation", in *Proc. of Languages, compilers and tools for embedded systems: software and compilers for embedded systems (LCTES'02-SCOPES'02)*, Jun. 2002.

[13] F. Angiolini, J. Ceng, R. Leupers, F. Ferrari, C. Ferri and L. Benini, "An integrated open framework for heterogeneous MPSoC design space exploration", in *Proc. of the Design, Automation and Test in Europe (DATE'06)*, pp. 1145-1150, Mar. 2006.

[14] www.multicube.eu

[15] L. Thiele, I. Bacivarov, W. Haid and K. Huang, "Mapping applications to tiled multiprocessor embedded systems", in *Proc. 7th Int. Conference on Application of Concurrency to System Design (ACSD 2007)*, pp. 29-40, Jul. 2007.

[16] G. Palermo, C. Silvano and V. Zaccaria, "A flexible framework for fast multi-objective design space exploration of embedded systems", *PATMOS 2003*, vol. 2799, pp. 249-258, Sep. 2003.

[17] S. Künzli, L. Thiele and E. Zitzler, "A modular design space exploration framework for embedded systems", *IEE Proc. Computer & Digital Techniques*, pp. 183-192, 2005.

[18] S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler, "PISA – a platform and programming language independent interface for search algorithms", *in C. M. Fonseca, P. J. Fleming, E. Zitzler, K. Deb and L. Thiele, editors, Evolutionary Multi-Criterion Optimization (EMO 2003)*, vol. 2632/2003 of LNCS, pp. 494-508. Springer-Verlag Heidelberg, 2003.

[19] http://www.tik.ee.ethz.ch/sop/pisa/

[20] J. Madsen, T. K. Stidsen, P. Kjarulf and S. Mahadevan, "Multi-objective design space exploration of embedded system platforms", *IFIP*, vol. 225, pp. 185-194, 2006.

[21] ARM Developer Suite, Version 1.2, www.arm.com

[22] Z. J. Jia, T. Bautista and A. Núñez, "Real-time application to multiprocessor-system-on-chip mapping strategy for a system-level design tool", *IEE Electronic Letters*, vol. 45, no. 12, pp. 613-615, 2009.