A System-Level Infrastructure for Multidimensional MP-SoC Design Space Co-Exploration

ZAI JIAN JIA, TOMÁS BAUTISTA, and ANTONIO NÚÑEZ, University of Las Palmas de Gran Canaria
ANDY D. PIMENTEL and MARK THOMPSON, University of Amsterdam

In this article, we present a flexible and extensible system-level MP-SoC design space exploration (DSE) 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, NASA deploys a so-called dimension-oriented DSE approach, allowing designers to configure the appropriate number of, well-tuned and possibly different, search algorithms to simultaneously co-explore the various design space dimensions. As a result, NASA provides a flexible and reusable framework for the systematic exploration of the multidimensional MP-SoC design space, starting from a set of relatively simple user specifications. To demonstrate the capabilities of the NASA framework and to illustrate its distinct aspects, we also present several DSE experiments in which, for example, we compare NASA configurations using a single search algorithm for all design space dimensions to configurations using a separate search algorithm per dimension. These proof-of-concept experiments indicate that the latter multidimensional 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.

Categories and Subject Descriptors: B.8.2 [**Performance and Reliability**]: Performance Analysis and Design Aids; J.6 [**Computer-Aided Engineering**]: *Computer-aided design (CAD)*

General Terms: Design, Experimentation, Performance

Additional Key Words and Phrases: System-level design space exploration, MP-SoC design

ACM Reference Format:
Jia, Z. J., Bautista, T., Núñez, A., Pimentel, A. D., and Thompson, M. 2013. A system-level infrastructure for multidimensional MP-SoC design space co-exploration. ACM Trans. Embedd. Comput. Syst. 13, 1s, Article 27 (November 2013), 26 pages.
DOI: http://dx.doi.org/10.1145/2536747.2536749

1. INTRODUCTION

Today's embedded systems are increasingly based on multiprocessors systems-on-chip (MP-SoC). These MP-SoCs typically contain multiple storage elements (SEs), networks (NEs), I/O components, and a number of heterogeneous programmable processors for flexible application support as well as dedicated processing elements (PEs) for achieving high performance and power goals [Martin 2006]. In order to cope with the design complexity of such systems in a time-efficient way, the abstraction level of the design

This research was supported by Spanish Science Ministry and Canary Agency for research and innovation.
Authors' addresses: Z. J. Jia, T. Bautista, and A. Núñez, Institute for Applied Microelectronics, University of Las Palmas de Gran Canaria, Spain; A. D. Pimentel and M. Thompson, Computer Systems Architecture Group, Informatics Institute, University of Amsterdam, Amsterdam, The Netherlands. Correspondence email: cjia@iuma.ulpgc.es.

DOI: http://dx.doi.org/10.1145/2536747.2536749

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. Therefore, such 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 [Gries 2004]: (i) evaluation of a design point in the design space using, for example, 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 [Reyes et al. 2004; Erbas 2007; Lee et al. 2010; Teich et al. 1997; Palesi and Givargis 2002; Jia et al. 2008]. For instance, system-level simulation is a popular method for evaluating single design points [Gries 2004]. These simulation tools usually operate at a high level of abstraction and are often based on the Y-Chart principle [Keutzer et al. 2000; Kienhuis et al. 1997]. According to this principle, any system can be specified by the combination of three models: an application model, an architecture model and a mapping model. 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, the 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 communication channels onto architecture resources, scheduling policies, and so on. Evidently, the more details (or dimensions) are taken into account, the larger the design space that needs to be searched, and therefore the more costly the analysis. Although many DSE approaches based on a large variety of search techniques have been proposed, the following 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 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, that is, 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 implement their 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

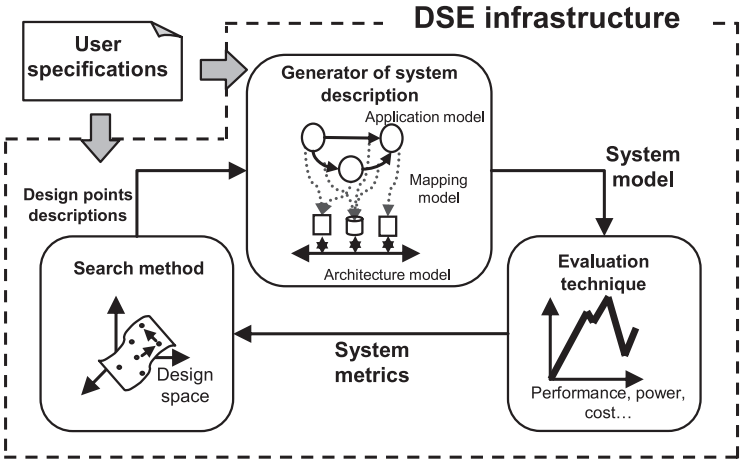


Fig. 1. Integration of an external system-level simulator with searching mechanism and a system generator in a single DSE infrastructure.

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 Figure 1.

To address these challenges, this article presents a 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 at system level, as well as allows designers to evaluate the DSE results in a time-efficient way. 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, number and types of available components in the platform architecture, costs/area, etc.

The remainder of the article is organized as follows. In the next section, related work and our contributions are presented. In Section 3, we describe various implementation aspects of the NASA framework. In Section 4, we present a range of experimental results, demonstrating NASA's capabilities. Finally, Section 5 concludes the article.

2. 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-SoCs [Thiele et al. 2007; Erbas 2007; Jia et al. 2008; Madsen et al. 2006]. 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 evaluation tools in a DSE environment can also be found in literature. In Mohanty et al. [2002], 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. [2006] 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. 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.

Lee et al. [2010] present a framework that determines the MP-SoCs for the optimal mapping of a real-time application. This two-phase approach selects first an optimal set of PEs for the mapping of the target application. Then, by means of a static estimation method based on the queuing model, they explore and prune the design space of communication architectures. Their framework also provides a set of interfaces that facilitate the integration of different simulation tools. Unlike our approach, they explore different design space axes in a sequential way, while our proposed approach co-explores simultaneously multiple design space dimensions. Moreover, their queuing model seems to be limited to bus-based topology, while our framework is flexible enough to analyze different kind of communication architectures.

The MultiCube project [MultiCube] has similar objectives as the work presented in this paper, but it mostly targets micro-architectural exploration of multiprocessors rather than system-level architectural exploration. 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 [Thiele et al. 2007; Palermo et al. 2003]. 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 Künzli et al. [2005], proposing a generic and modular framework based on PISA [Bleuler et al. 2003] for DSE of embedded systems. 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, for example, the problem of incorporating a system model generator and external simulation tools has not been addressed.

Using precompiled and ready-to-use search algorithms available at the PISA framework [PISA], Madsen et al. [2006] 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. However, the chosen representation formats for internal interfaces in their work 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.

To conclude this section, we summarize our contributions as follows. First, we propose a flexible and modular 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

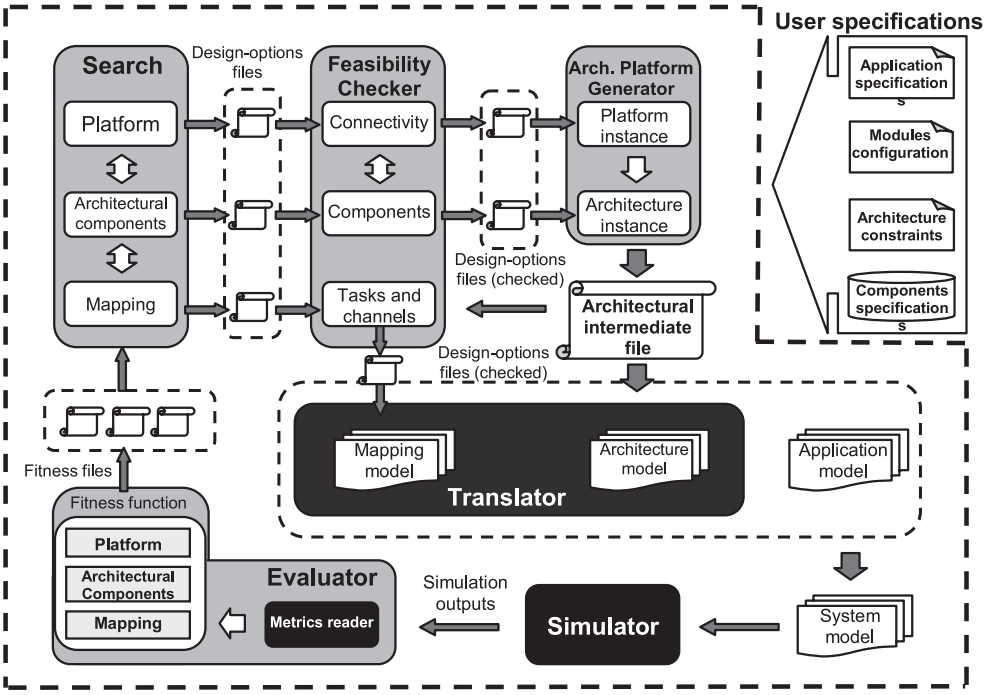


Fig. 2. The NASA framework.

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, well-tuned and possibly different, search algorithms to simultaneously co-explore the various design space dimensions.

A high-level overview of the NASA framework and the concept of dimension-oriented DSE have been introduced in Jia et al. [2010]. In this article, we provide more detailed explanations about the internal implementation of the NASA framework and, in particular, of its system-model generator. Moreover, we also present a significant number of proof-of-concept DSE experiments to demonstrate the different capabilities and benefits of the NASA framework, as will be illustrated in the next sections.

3. THE NASA FRAMEWORK

The infrastructure of NASA is shown in Figure 2. Essentially, six main modules can be distinguished in the framework: the Search module, Feasibility Checker, Architectural Platform Generator, Translator, Simulator, and Evaluator. Subsequently, the different interfaces used by NASA as well as the functionality and implementation of each of these modules are discussed.

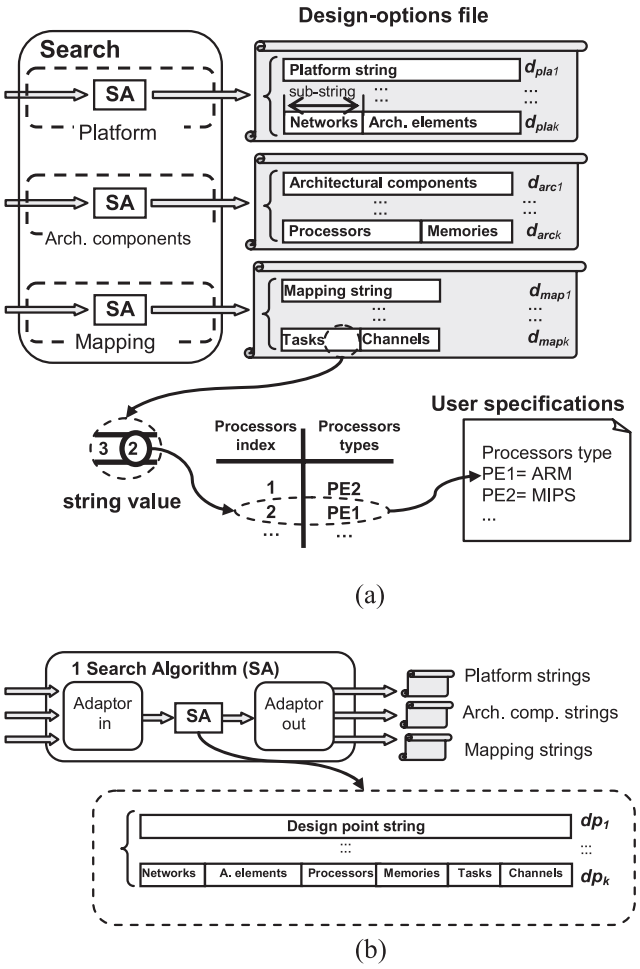


Fig. 3. Interfaces and strings for Search module with (a) three SAs and (b) single SA.

3.1. Interfaces

The NASA interfaces allow each module to act like an independent black box inside the framework. As a result, different modules handling new functionalities or additional dimensions can be easily integrated in a plug-and-play fashion without needing to modify other modules. 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 submodules of both the Search module and the Feasibility Checker. Note that all these files are dynamically and automatically created (and updated) by NASA, designers therefore do not need to implement any script.

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, three dimensions are currently distinguished in NASA (platform, architectural component, and mapping exploration), and each explored dimension uses a separate design-options and fitness file. For example, in the 3-level DSE shown in Figure 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 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 PEs and SEs 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 dimension 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 3.2. Examples of design-decision strings are shown in Figure 3, for three (Figure 3(a)) and one (Figure 3(b)) search algorithms (SA) in the Search module.

The length of a string for each dimension may also vary. Using the example shown in Figure 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 PEs and 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 Figure 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, for example, change the type of PE or add a new type without the need to adapt any module implementation. Note that the choice of the representation scheme has a strong impact on the flexibility and scalability of the framework. And in this case, our representation scheme enables to symbolically represent a large design space, as well as guarantees that each potential solution of the design space receives a unique encoding value.

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 module 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 module requires, for example, 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 3.3.

3.2. 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 introduced in our previous work [Jia et al. 2010], NASA applies a dimension-oriented design space co-exploration approach. That is, all dimensions can be explored simultaneously using a single search algorithm, or co-explored 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 design space exploration as multiple independent explorations, but instead, there exists a tight connection or communication between the search algorithms as well as the results from all dimensions are simultaneously taken into account. That is, a design point dp can be expressed by linking k available design decision values $\{d_1, d_2, \ldots, d_k\}$ corresponding to each of k design space dimensions.

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. Two examples of the linking technique are depicted in Figure 4. For example, using a pyramidal technique (as shown in Figure 4(a)), all design decisions in the

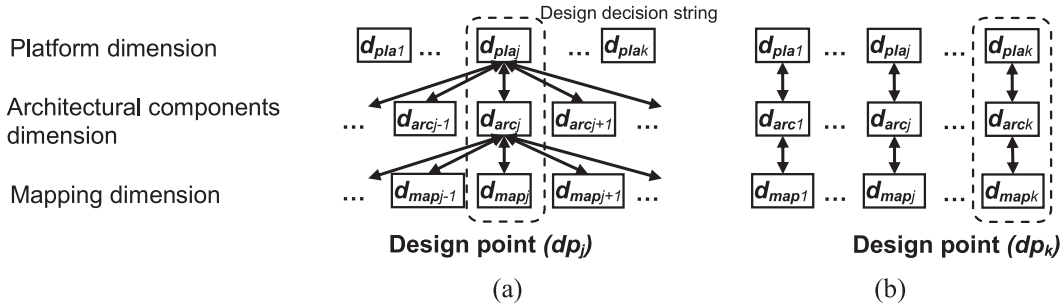


Fig. 4. Techniques of linking design decision strings: (a) Pyramidal and (b) one-to-one technique.

mapping dimension are linked with each of the decisions in the architectural components dimension, while the latter are again all linked with each of the design decisions in the platform dimension. However, this means that the number of design points to be evaluated in each search iteration grows exponentially with the number of design decisions of each dimension. On the other hand, using a pure one-to-one linking technique, as shown in Figure 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 of the design-options files, assuming that all design-options files have the same number of strings. This clearly reduces the number of 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", that is, 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} or d^B_{arc}, but we 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 there is not a single set of the collecting iteration values in the practice, but these values are set by the designer in each experiment taking into account the number of explored dimensions, the size of the design space, and the search algorithms used in the DSE experiment.

3.3. 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 our current implementation, the checking process consists of two stages: the architecture checking and the mapping checking.

During the architecture checking process, the platform string is first checked to determine whether or not the specified platform template (to be discussed in more detail in Section 3.4) contains a valid topology. 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. But before the architecture checking is done, a database containing all feasible architectures of the explored design space is generated first. The description of the feasible architecture (in the database) is a concatenation of the platform string and architectural component string, where both strings use the same string format as explained in Section 3.1. Note that the database is ordered for the purpose of speeding up the checking process. This way, a design point with an infeasible architecture can be identified by simply comparing its platform and architectural components string with the descriptive strings stored in the database. Finally, if an infeasible architecture is detected, a descriptive string in the database with the highest number of string values in common is chosen to replace the infeasible design point.

The mapping checking is carried out on design points with a feasible architecture. In our current implementation, we use a heuristic minimum-distance repair algorithm, which introduces a minimum number of modifications to an infeasible mapping in order to obtain a feasible one [Erbas 2007]. This heuristic algorithm first considers whether each application task is mapped onto a PE that has been allocated in the platform (i.e., a feasible set of PEs), and if not, it repairs by randomly mapping the task to a feasible PE. Subsequently, the repair algorithm checks for each communication channel connecting two tasks whether these tasks are mapped onto different PEs (e.g., PE1 and PE2). If a channel is assigned to a SE that is not reachable from both PE1 and PE2, then the heuristic algorithm selects randomly a SE from the set of reachable SEs and relocates the communication channel into that SE. Note that if there is not such a set of reachable SEs, no more repair is applied and the design point is considered as infeasible. 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 set of repair algorithms presented in this section has been widely tested and has demonstrated good efficiency and effectiveness in the practice [Erbas 2007]. More specifically, these repair algorithms not only have a minimal effect on the run-time of the framework, but also can warrant the repair of a high percentage of infeasible design points in our DSE experiments (as will be illustrated in Section 4.2). Finally, we would like to stress here that other repair mechanisms can be also used in NASA framework, if they satisfy the interface requirement of our framework.

3.4. Architectural Platform Generator

The main mission of this module is to generate the architectural intermediate file, which combines both feasible platform and architectural components information containing in the strings of their respective design-option files. The resulting architectural intermediate file is used later for (i) feasibility checking of mapping strings, and (ii) as

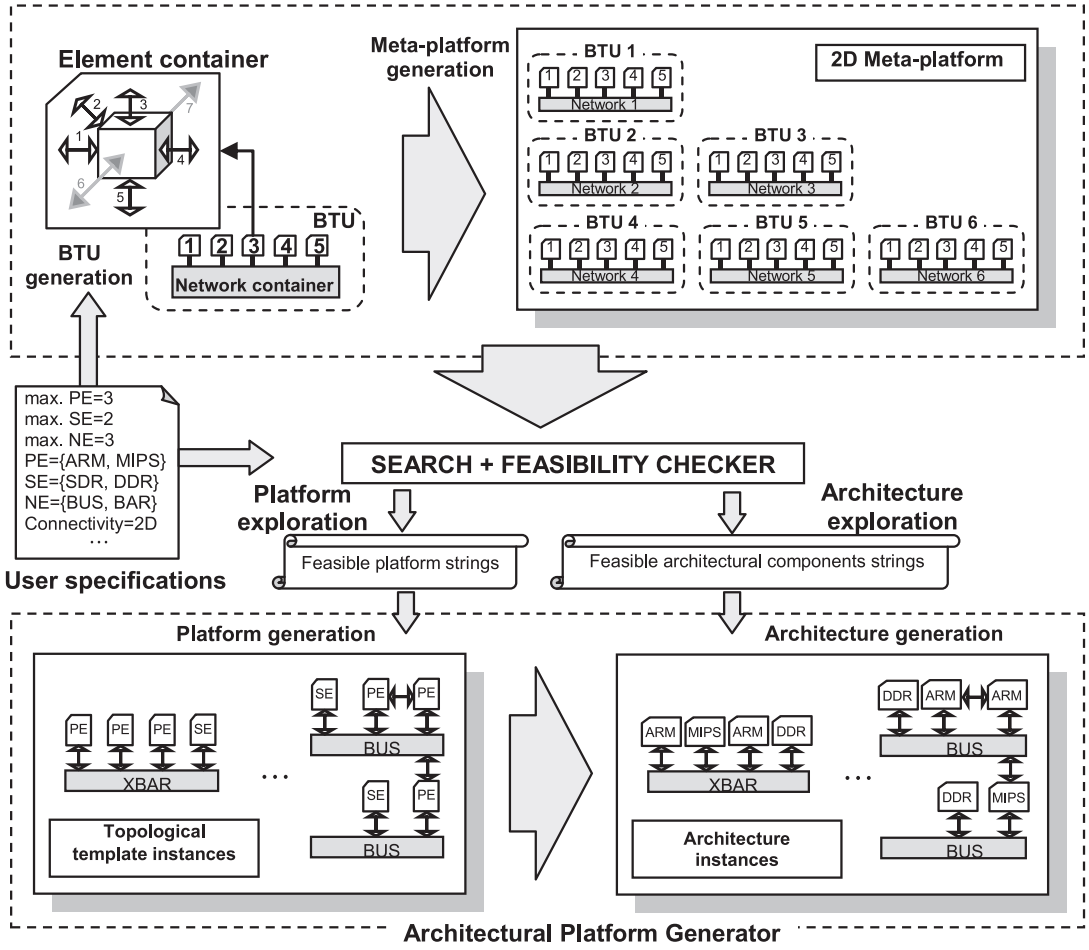


Fig. 5. Generation of topological templates and architecture instances.

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 in the architectural intermediate file 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 Figure 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, which specify the maximum number of PEs, SEs, and NEs in a platform, the available types for each kind of architectural components, the connectivity property between different BTUs, and so on. 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 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 Figure 5). Thus, a BTU can be replicated through two or three directions and, as consequence, different kinds of meta-platforms can be generated according to the user specifications (e.g., architectures based on buses, crossbar, grid architectures composed of several dozens of PEs, etc.). A 2D meta-platform generation process is shown in Figure 5, although a 3D meta-platform can also be generated if the gray links of an element container (top-left corner of Figure 5) are also used during this process. It should be noted 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 (by the 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) 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 3.2 and 3.3, 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 purpose, the actual component types of the element containers in a template are added. In the example of Figure 5, this means that, for example, 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.

3.5. 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 point 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 specification(s) 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.

The implementation of a Translator for a target simulator is a process that requires some effort from the designer, since a design point is usually described in each simulator using a particular language. Evidently, the designer must (know and) adhere to the syntax and semantics of the target simulator descriptions to avoid inconsistent translations. That is, the Translator acts as a synthesis step, that is, converts NASA's internal format of a design point (the architectural intermediate file) to a file-based format that is specific for the target simulator plugged into NASA in each DSE experiment. Moreover, the complexity of this conversion process also varies from simulator to simulator, that is, it depends on the quantity of the implementation details that

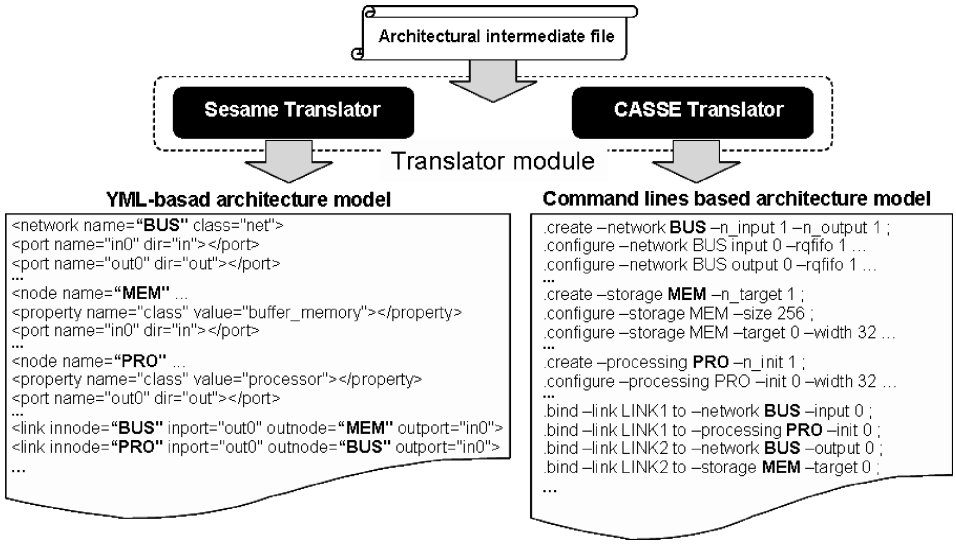


Fig. 6. Plug-in examples for generating architectural model in Sesame and CASSE.

should be included in the resulting system model. For example, Figure 6 illustrates an example of the conversion to different architectural models (from a single design point specification) for two system-level simulators, Sesame [Erbas 2007] and CASSE [Reyes et al. 2004]. While the CASSE translator, containing 1100 lines of codes, generates system models based on command-line expressions, the Sesame translator requires the implementation of 880 lines of code and produces descriptive models based on the Y-Chart Modelling Language (YML). Experience shows that these translation efforts typically take one or a few weeks.

However, we would like to point out that the implementation of a tailored Translator is a *one-time effort*. This means that such a Translator can be reused in the future for different kinds of DSE experiments, allowing designers to save a lot of time and effort in manually implementing different architectural models and/or experiment-customized scripts. As a consequence, the integration of a new system-level simulator in NASA only requires the adaptation of the Translator module, while all other modules remain unaffected. This is why two kinds of module colors can be identified in Figure 2, simulators-dependent (black) and simulators-independent (grey) modules.

3.6. Simulator

At this moment, we have successfully integrated in NASA two system-level simulation tools: CASSE and Sesame. 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, storage elements and network elements. The number of elements of each type and their configuration (e.g., number and width of ports, clock, memory size, network arbitration scheme, task scheduling policy, etc.) can also be properly configured in this architectural model. Finally, another description file is used by CASSE and Sesame to control the mapping of the application

onto the architecture. Obviously, all the required models and descriptions can directly be generated by a customized Translator module, as it was explained in Section 3.5.

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 automated way, without any human intervention. To give an example, the simulators are highly parameterized 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 the simulation tools is beyond the scope of this article, the interested reader is referred to Gries [2004] for an overview of existing system-level simulators, and to Reyes et al. [2004] and Erbas [2007] for more information about the implementation details, speed-accuracy tradeoff, the specification method of the application and architecture models in CASSE and Sesame.

3.7. Evaluator

The simulation of design points can typically generate a variety of quantitative information about the evaluated systems, such as data about performance, cost/area, and power consumption. 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 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, for example, to 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, as explained in Section 3.2. 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 3.2) to address the under-exploration problem in hierarchical design space explorations with multiple search algorithms. Formally, these hierarchical fitness functions are formulated as follows:

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

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

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

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	-	No. of individuals per iteration
Simulation tool	1	-	CASSE

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).

4. EXPERIMENTAL RESULTS

4.1. NASA Configurations for Experiments and Parameter Settings

The first set of experiments aims at comparing the more traditional approach (using a single search algorithm for all design space dimensions) to our dimension-oriented approach (using a separate search algorithm per dimension, that is, three search algorithms (SAs) in total). To properly evaluate and compare the quality of the DSE results of the different approaches, many indicators can be used [Basseur et al. 2002; Erbas 2007; Gries 2004]. In this article, we use the following 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 in local optima.
(3) *Coverage*. The explored points should be well distributed for achieving a complete view of the landscape of the design space as well as for 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, that is, 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. Note that after specifying such user specifications, the NASA

framework automates the further DSE process. The studied MP-SoCs 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). Although not studied in this set of experiments, NASA and the incorporated simulators (i.e., CASSE and Sesame) also allow designers to explore different configurations in the architectural components, such as the bandwidth of a NE, the size of a SE, the scheduling policy of a PE, etc. The latter can be done by introducing and configuring an additional dimension in NASA, as already explained in Section 3.

A real-life multimedia application is used to be mapped onto the target MP-SoC platforms in our DSE experiments. This application is an optimized version of the computer vision algorithm presented in Jia et al. [2008]. 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.

4.1.1. 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. In this article, we focus on implementations based on genetic algorithms (GAs) since GA-based DSE has been widely studied in the domain of system-level design [Teich et al. 1997; Palesi and Givargis 2002; Erbas et al. 2006; Künzli et al. 2005; Madsen et al. 2006], 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 [Erbas et al. 2006] could also have been used. However, it should be mentioned that our interest is not focused on the type of GA used in each experiment. Instead, we aim to analyze the behaviour of the design space co-exploration process when multiple search algorithms are used.

4.1.2. Crossover and Mutation Type Settings. The crossover and mutation operators in our GAs are performed at the granularity of entire sub-strings (see Figure 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 substring. In "independent" mutation (M = 6), the mutation probability is used for each of the six substrings 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, "3ga1 $\times$ 6" 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).

4.1.3. Group of Experiments and Run-Time per Simulation. Each experiment consists of a maximum of 410 simulations (41 iterations $\times$ 10 individuals per iteration). Note that the GA is extremely sensitive to its parameters such as the initial population, the probability associated to crossover (*pc*) and mutation (*pm*) operators. That is, the variation of the results achieved in DSE experiments with different GA parameter settings can

be significant. Therefore, in order to achieve a fair comparison between the traditional approach and our dimension-oriented approach, many experiments with different combinations of GA parameters settings must be done to find out the well-tuned GA (with the best parameters settings) for each studied case. To this end, all possible combinations of the *pc* and *pm* values (as listed in Table I) have been evaluated. This results in 20 groups (i.e., 20 different initial populations) of 25 experiments (5 *pc* probabilities $\times$ 5 *pm* probabilities) for each of the four mentioned NASA configurations.

We have plugged the CASSE simulator into NASA for the evaluation of the design points in our DSE experiments. The total run time of the NASA framework is defined as the sum of the times spent in the searching, feasibility checking, system model generation and simulation. The simulation—which dominates the run-time in our DSE experiments—represents 99 percent of the total run time. More specifically, CASSE 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. It should be noted that although the evaluation of the design points in our experiments can be carried out by means of simulators and/or analytical methods, the analysis of speed-accuracy tradeoff between different evaluation techniques is not discussed in this article.

4.1.4. 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.

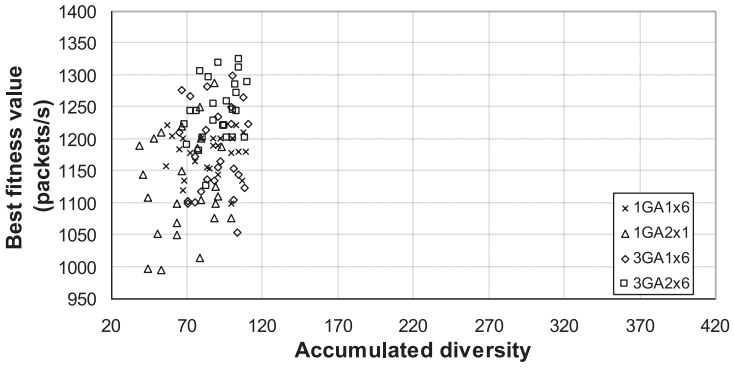
4.2. DSE Behaviour and Sensitivity to Various Parameter Settings

4.2.1. Impact of Number of Search Algorithms on DSE Quality. The results of those experiments are shown in the four scatter-plots of Figure 7, 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.

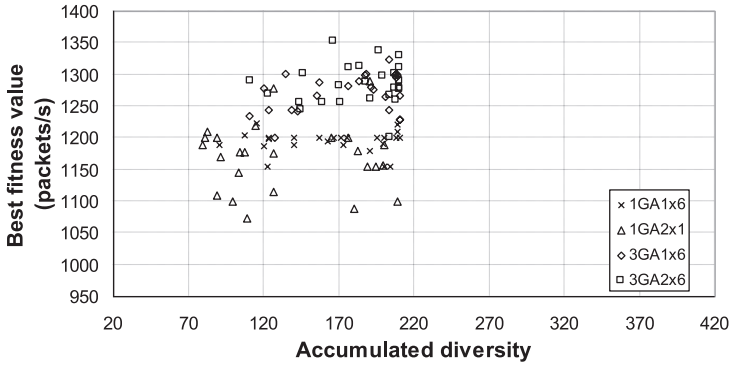
If the input arrival frame rate is 1450 packets/s and a minimum of 1250 packets/s has to be processed to satisfy the minimum real-time requirements of the studied application (which is equivalent to processing 25 frames/s), then using a 3 GA-based search approach in NASA not only provides the design alternatives with the best fitness values (in the upper right corner for each scatter-plot of Figure 7) 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.

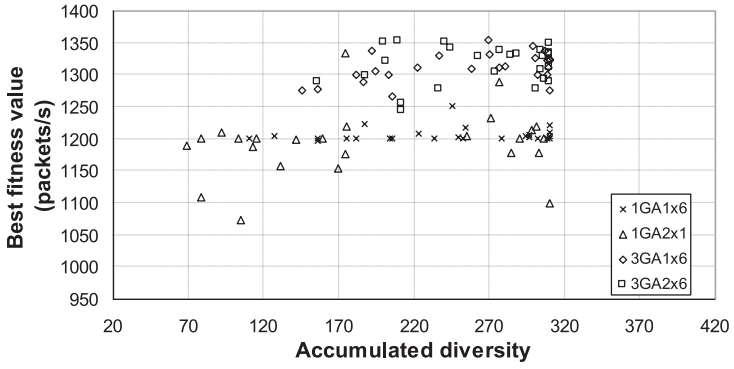
For a detailed comparison between both approaches (single GA and multiple GA search), the three proposed criteria—diversity, convergence and coverage—are separately analyzed in Figure 9, Figure 10, and Figure 11. To this end, we have selected one group of experiments for each of the four mentioned NASA configurations, where each



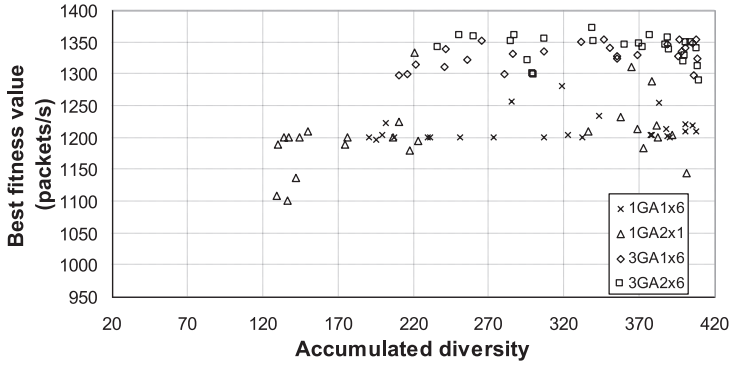
(a) 10 iterations.



(b) 20 iterations.

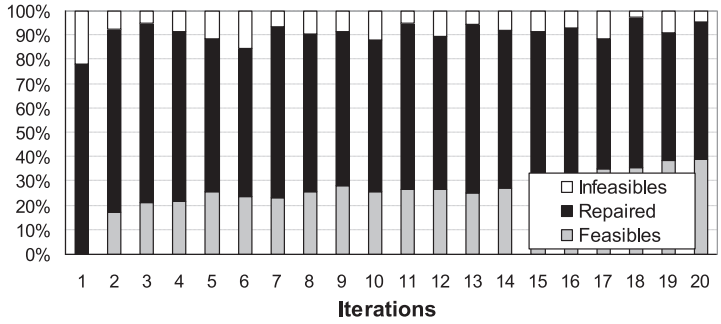


(c) 30 iterations.

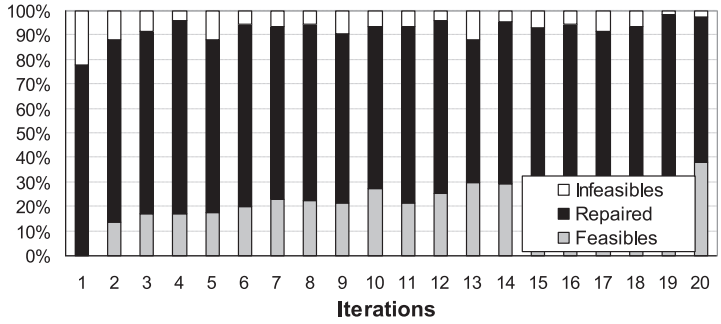


(d) 40 iterations.

Fig. 7. DSE results for four NASA configurations.



(a) 1 SA: average percentage of repair = 84,21%



(b) 3 SAs: average percentage of repair = 86,68%.

Fig. 8. Average percentage of feasible, repaired and infeasible design points per iteration in our DSE experiments.

configuration uses its respective best GA parameters setting, that is, the parameters setting with which each configuration achieved its best results of the DSE.

4.2.2. Impact of Repair Mechanisms on the Efficiency of DSE Process. In our experiments, due to the repair techniques applied in the Feasibility Checker module, the feasible alternatives actually evaluated represent an important percentage of the total number of explored design points. This can be illustrated in Figure 8(a) and Figure 8(b), which depict the average percentage of feasible, repaired and infeasible design points per iteration for the DSE experiments based on both approaches. Note that the grey part of each bar (in Figure 8) 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 repair techniques. In this latter case, no system models are generated for these infeasible design points, and therefore, they are not evaluated by the simulation tool.

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

4.2.3. Convergence Rate and Number of Iterations. The convergence is shown in Figure 9, 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

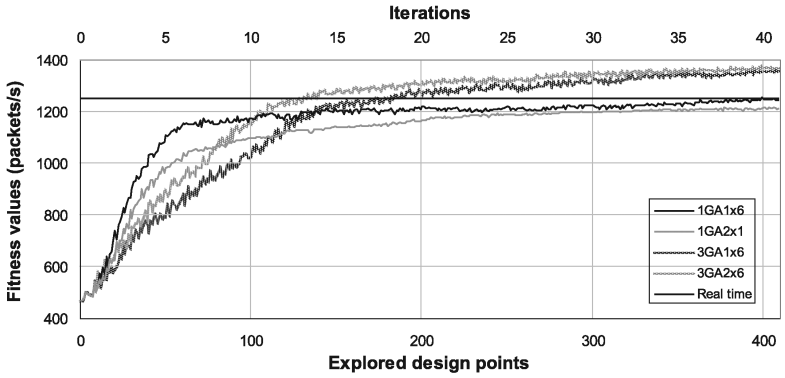


Fig. 9. Average fitness values per iterations.

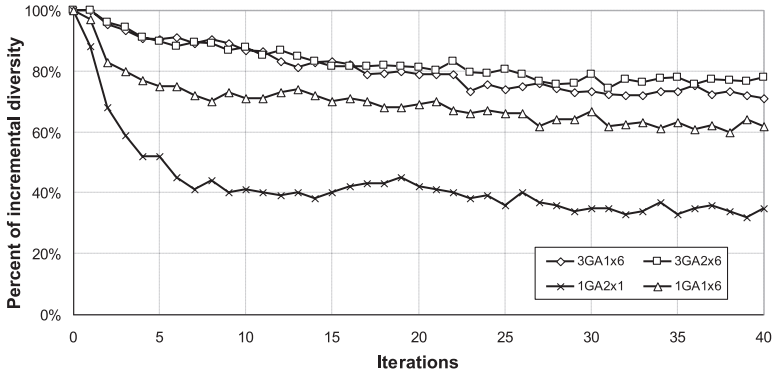


Fig. 10. Incremental diversity per iterations.

functions and the variant of the one-to-one individual linking technique (presented in Section 3.2) in 3 GA-based experiments.

However, when the number of iterations increases, 3 GA-based experiments do not only 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 may 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 Figure 10 and Figure 11.

4.2.4. Diversity and the Search Approach. Each curve in Figure 10 represents the percentage of new and different design points introduced in each iteration that have not been explored in any of the previous iterations, that is, the 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 Figure 7.

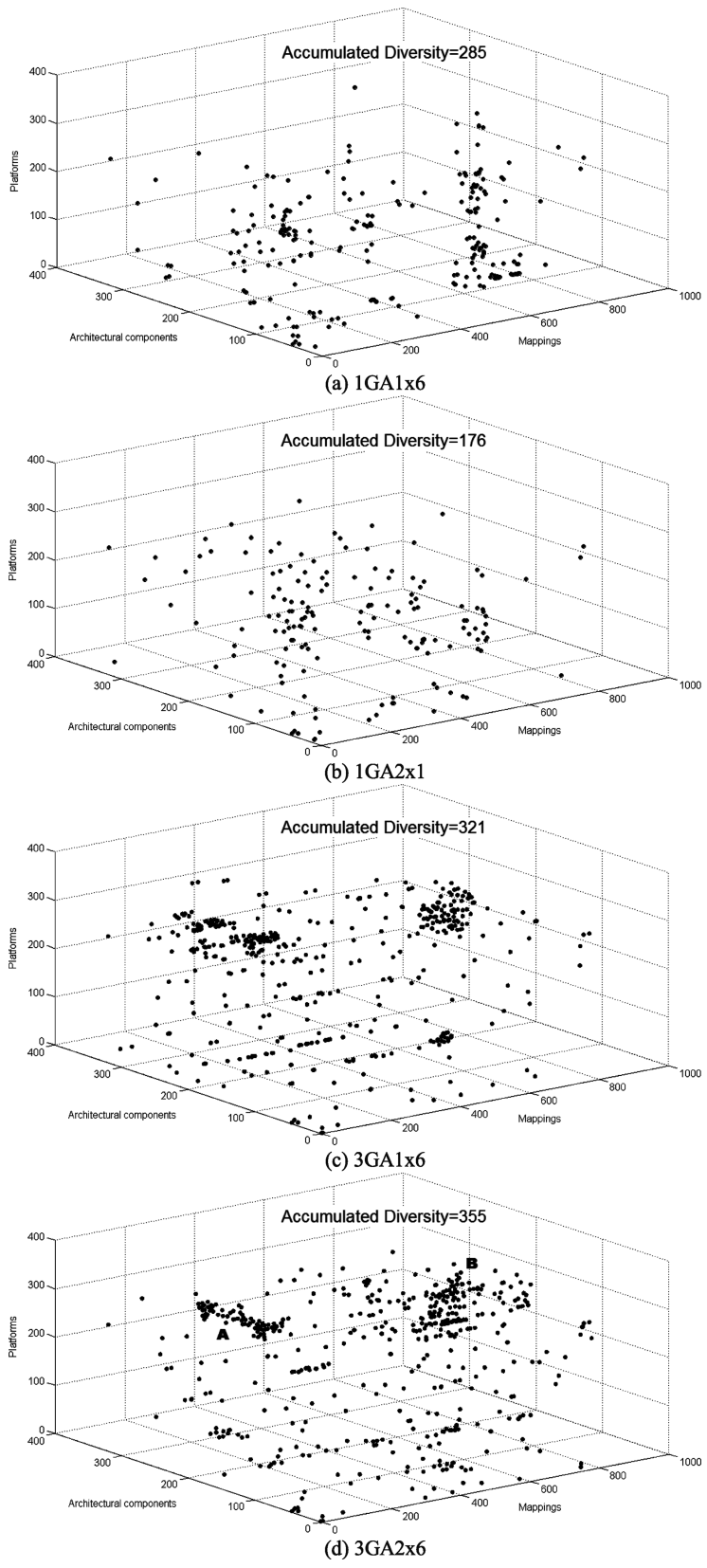
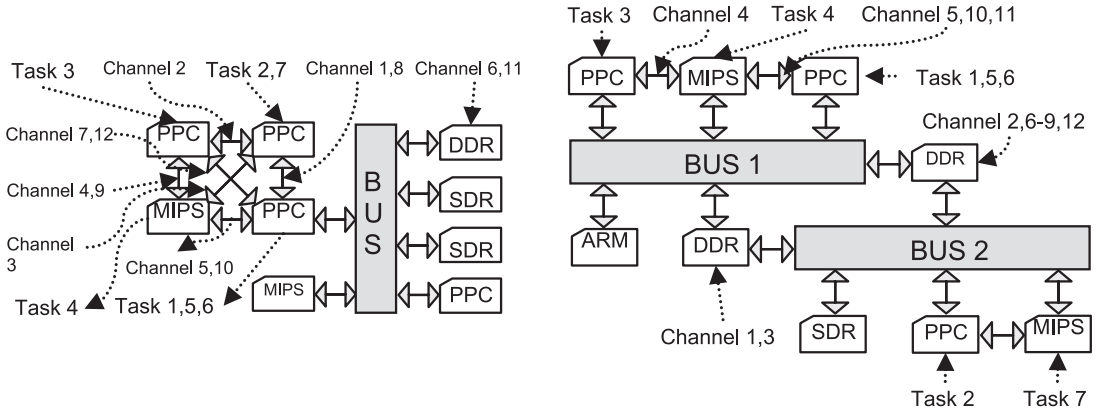


Fig. 11. Explored design points by each selected NASA configuration.



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

Fig. 12. Examples of design points found by 3GA2×6 DSE after 40 iterations.

4.2.5. Convergence, 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 Figure 11, where each axis represents one design space dimension in our 3D design space. Moreover, for a fair comparison, each axis in Figure 11 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 Figure 11 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 Figure 9.

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 Figure 11, where multiple areas of design points concentration (or convergence) 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 Figure 11(d), and their respective architectures and mappings are shown in Figure 12. Note that designers can select from NASA's output any design point explored in the DSE experiment, and examine information about that design point such as the architectural characteristics, the description of the mapping, etc.

In this case, although both design points (corresponding to each of the convergence regions) have similar performance (A achieves 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 new DSE experiments with a larger number of iterations (e.g., 60 iterations) and/or introduce additional objectives in the fitness functions (such as the cost of designs) for achieving a better convergence toward the optimal points. Alternatively, since NASA's output (Figure 11) can provide a good insight of where the sweet spots in the design space are located, designers can also perform more detailed explorations focusing on a particular convergence area, for

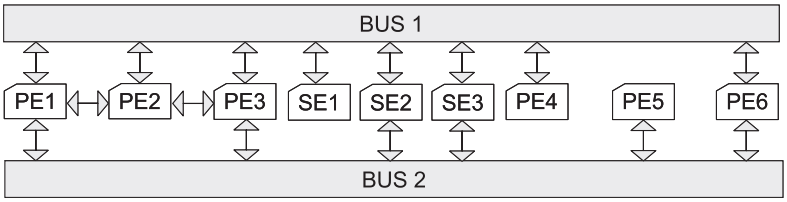


Fig. 13. Target platform template for the second set of experiments.

Table II. Search Module and Target Architecture Parameter Settings

GA	Selection (S)		Proportional with elitism
	Crossover (C)		1-point, pc = 0,5
	Mutation (M)		Independent (M = 6), pm = 0,5
ga	Selection (S)		Tournament without elitism
	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	Number of individuals per iteration
PE		≤ 6	ARM and hardware dedicated block
SE		≤ 3	DDR and SDR

example, the platform is fixed and only the architectural components and mapping dimensions are explored more rigorously. In the next section, we will present another set of experiments to demonstrate NASA's flexibility and capacity to carry out this kind of DSE process.

4.3. Hierarchical Refinement and Analysis of 2D-DSE with NASA

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, that is, 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.

4.3.1. Fixed Platform, Variable Architectural Components and Mappings. The selected target platform template and available type values for PE and SE are depicted in Figure 13 and Table II. This platform template can provide architecture models based on two AMBA buses connecting up to six PEs and three SEs. The execution time of the visual tracking application's tasks has been estimated using an instruction set simulator [ARM] for the ARM processor, while we assume that the hardware dedicated block (which executes block matching operations of the target application) has a $\times 10$ 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, for the sake of illustration we have 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.

4.3.2. NASA Configuration for 2D-DSE. Four NASA configurations have been selected in this second set of experiments: 1GA+1Random, 1GA+1ga, 1GA+1GA and a heuristic

algorithm from Jia et al. [2009]. 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. 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 Figure 7 for our first set of experiments). Therefore, the four selected configurations only represent a few samples of NASA's capacity and flexibility.

4.3.3. 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 [Jia et al. 2009] 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, that is, 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 task-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, that is, all combinations of type and locations of PEs and SEs in 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 the MP-SoC instance that satisfies the real-time constraint. Interested readers are referred to Jia et al. [2009] 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 *nondeterministic or nonlinear 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.

4.3.4. Results and Discussions. The results of these four configurations are shown in Figure 14. The dark bar represents the fitness value obtained in simulation with the design point selected by the aforementioned heuristic algorithm. The curves show for the rest of the configurations (i.e., 1GA+1GA, 1GA+1ga, and 1GA+1Random) the average fitness values reached by all individuals in each of the twenty iterations. From these results, it can be seen that 1GA+1Random can only sporadically reach a few design points that satisfy the real-time constraints. Moreover, it clearly cannot ensure convergence toward a global optimum. On the other hand, both the heuristic algorithm

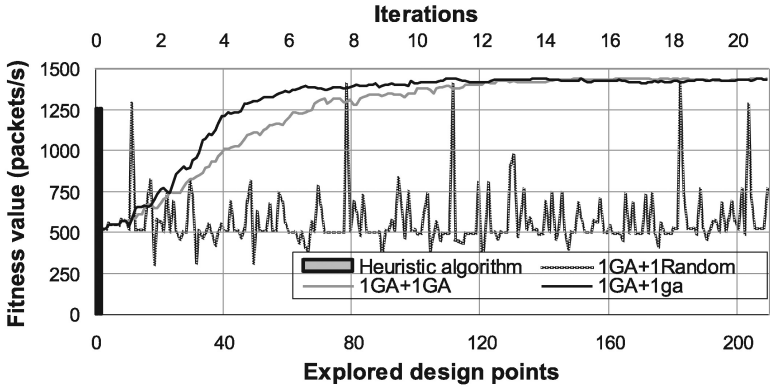


Fig. 14. Comparative results obtained in the second set of DSE experiments.

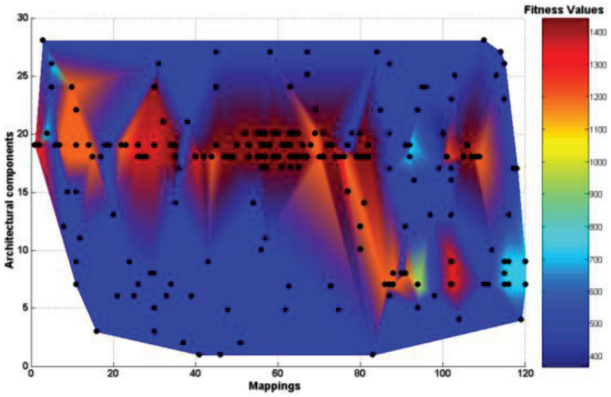


Fig. 15. Example of NASA output.

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 1GA+1ga and 1GA+1GA need to simulate an average number of 40 and 60 individuals respectively (since 1GA+1ga has a higher convergence rate than 1GA+1GA 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 two important benefits with respect to the heuristic algorithm: (i) both 1GA+1GA and 1GA+1ga can converge toward design points with higher fitness values, and (ii) our co-exploration approach provides 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, that is, 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.

This last aspect can be illustrated in Figure 15, which shows an example of typical NASA output after each DSE experiment. The set of simulated design points, corresponding to a 1GA+1ga 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, that is, the x axes and y axes of the 2D view contain the explored instance numbers of the mapping and architectural components dimensions. So, for example, 120 different mappings have been explored in this example. Note that the fitness value associated to each design point is colour coded, ranging from red (high fitness value) to blue (low fitness value). Therefore, the distribution of design points in the surface can clearly indicate the location of convergence region(s). Finally, it should be stressed that all these experiments presented in this article have been performed in a fully automatic fashion, only providing parameter settings and constraints such as those shown in Table I and Table II.

5. CONCLUSIONS

In this article, we addressed the lack of a generic, flexible, and re-usable infrastructure to facilitate and support system-level MP-SoC 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 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 well-tuned, 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 multidimensional MP-SoC design space.

Our experimental results indicate that, compared to the more traditional approach of using a single search algorithm for all dimensions, the multidimensional co-exploration seems to be able to find better design points and ensure the convergence toward global optima. Furthermore, the multidimensional co-exploration has a higher diversity and coverage of design alternatives, producing higher quality DSE results. More experiments are needed, however, to demonstrate that multidimensional DSE consistently outperforms traditional DSE. 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, for example, through the integration of the PISA optimization framework [Bleuler et al. 2003; PISA], as well as to perform additional deployment case studies of NASA such as multi-objective optimization problems introducing other fitness and cost functions. Moreover, we intend to integrate NASA with the Daedalus system-level MP-SoC synthesis framework [Thompson et al. 2007] to validate NASA's DSE results against actual FPGA-based prototype MP-SoC implementations.

REFERENCES

ANGIOLINI, F., CENG, J., LEUPERS, R., FERRARI, F., FERRI, C., AND BENINI, L. 2006. An integrated open framework for heterogeneous MPSoC design space exploration. In *Proceedings of the Design, Automation and Test in Europe (DATE'06)*, 1145–1150.

ARM DEVELOPER SUITE, Version 1.2, www.arm.com.

BASSEUR, M., SEYNHAEVE, F., AND TALBI, E. 2002. Design of multi-objective evolutionary algorithm: application to the flow-shop scheduling problem. In *Proceedings of Evolutionary Computation*. Vol. 2, 1151–1156.

BLEULER, S., LAUMANNS, M., THIELE, L., AND ZITZLER, E. 2003. PISA—a platform and programming language independent interface for search algorithms. In *Proceedings of the Symposium on Evolutionary Multi-Criterion Optimization (EMO'03)*. C. M. Fonseca, P. J. Fleming, E. Zitzler, K. Deb, and L. Thiele, Eds., Lecture Notes in Computer Science, vol. 2632/2003, Springer-Verlag, Heidelberg, 494–508.

ERBAS, C., CERAV-ERBAS, S., AND PIMENTEL, A. 2006. Multiobjective optimization and evolutionary algorithms for the application mapping problem in multiprocessor system-on-chip design. *IEEE Trans. Evolut. Computat. 10*, 3, 358–374.

ERBAS, C. 2007. System-level modeling and design space exploration for multiprocessor embedded system-on-chip architectures. PhD thesis, Amsterdam University Press, Amsterdam, the Netherlands.

GRIES, M. 2004. Methods for evaluating and covering the design space during early design development. *Integration, VLSI J. 38*, 2, 131–183.

JIA, Z. J., PIMENTEL, A. D., THOMPSON, M., BAUTISTA, T., AND NÚÑEZ, A. 2010. NASA: A generic infrastructure for system-level MPSoC design space exploration. In *Proceedings of the IEEE 8th Workshop on Embedded Systems for Real-time Multimedia*, 41–50.

JIA, Z. J., BAUTISTA, T., AND NÚÑEZ, A. 2009. Real-time application to multiprocessor-system-on-chip mapping strategy for a system-level design tool. *IEEE Electron. Lett. 45*, 12, 613–615.

JIA, Z. J., BAUTISTA, T., NÚÑEZ, A., GUERRA, C., AND HERNANDEZ, M. 2008. Design space exploration and performance analysis or the modular design of CVS in a heterogeneous MPSoC. In *Proceedings of the Conference on Reconfigurable Computing and FPGA (ReConFig'08)*. 193–198.

KEUTZER, K., MALIK, S., NEWTON, A., RABAEY, J., AND SANGIOVANNI-VINCENTELLI, A. 2000. System level design: Orthogonalization of concerns and platform-based design. *IEEE Trans. Comput.-Aided Des. Integ. Circ. Syst. 19*, 12, 1523–1543.

KIENHUIS, B., DEPRETTERE, E., VISSERS, K., AND VAN DER WOLF, P. 1997. An approach for quantitative analysis of application-specific dataflow architectures. In *Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures and Processors*. 338–349.

KÜNZLI, S., THIELE, L., AND ZITZLER, E. 2005. A modular design space exploration framework for embedded systems. *IEEE Proc. Comput. Digit. Tech.* 183–192.

LEE, C., KIM, S., AND HA, S. 2010. A systematic design space exploration of MPSoC based on synchronous data flow specification. *J. Sig. Proc. Syst. 58*, 2, 193–213.

MADSEN, J., STIDSEN, T., KJARULF, P., AND MAHADEVAN, S. 2006. Multi-objective design space exploration of embedded system platforms. *IFIP*. Vol. 225, 185–194.

MARTIN, G. 2006. Overview of the MPSoC design challenge. In *Proceedings of Design Automation Conference (DAC'06)*.

MOHANTY, S., PRASANNA, V., NEEMA, S., AND DAVIS, J. 2002. Rapid design space exploration of heterogeneous embedded systems using symbolic search and multi-granular simulation. In *Proceedings of Languages, Compilers and Tools for Embedded Systems: Software and Compilers for Embedded Systems (LCTES'02-SCOPES'02)*.

MULTICUBE, www.multicube.eu.

PALERMO, G., SILVANO, C., AND ZACCARIA, V. 2003. A flexible framework for fast multi-objective design space exploration of embedded systems. In *Proceedings of PATMOS'03*. Lecture Notes in Computer Science, vol. 2799, Springer-Verlag, Berlin, 249–258.

PALESI, M. AND GIVARGIS, T. 2002. Multi-objective design space exploration using genetic algorithms. In *Proceedings of International Symposium on Hardware/Software Codesign (CODES'02)*. 67–72.

PISA, www.tik.ee.ethz.ch/sop/pisa/.

REYES, V., BAUTISTA, T., MARRERO, G., CARBALLO, P., AND KRUIJTZER, W. 2004. CASSE: A system-Level modeling and design-space exploration tool for multiprocessor systems-on-chip. In *Proceedings of the Euromicro Symposium on Digital System Design (DSD'04)*. 476–483.

TEICH, J., BLICKLE, T., AND THIELE, L. 1997. An evolutionary approach to system-level synthesis. In *Proceedings of the 5th International Workshop on Hardware/Software Co-Design (Codes/CASHE'97)*. 167–171.

THIELE, L., BACIVAROV, I., HAID, W., AND HUANG, K. 2007. Mapping applications to tiled multiprocessor embedded systems. In *Proceedings of the 7th International Conference on Application of Concurrency to System Design (ACSD'07)*. 29–40.

THOMPSON, M., STEFANOV, T., NIKOLOV, H., PIMENTEL, A. D., ERBAS, C., POLSTRA, S., AND DEPRETTERE, E. F. 2007. A framework for rapid system-level exploration, synthesis, and programming of multimedia MP-SoCs. In *Proceedings of the International Conference on Hardware-Software Codesign and System Synthesis (CODES+ISSS'07)*. 9–14.

Received August 2011; revised February 2012; accepted April 2012