System-level MP-SoC Design Space Exploration Using Tree Visualization

Toktam Taghavi, Andy D. Pimentel, Mark Thompson

Computer Systems Architecture group
Informatics Institute, University of Amsterdam
Amsterdam, The Netherlands
{T.TaghaviRazaviZadeh, A.D.Pimentel, M.Thompson}@uva.nl

Abstract— **The complexity of today's embedded systems forces designers to model and simulate systems and their components to explore the wide range of design choices. Such design space exploration is especially needed during the early design stages, where the design space is at its largest.**

Due to the exponential design space in real problems, evaluating and comparing every single point in the design space is infeasible. Therefore, heuristic search techniques, such as Evolutionary Algorithms (EA), are often used to search the design space for optimum design points using only a finite number of design-point evaluations. Understanding how the design space was searched by such searching algorithms and providing insight into the "landscape" of the design space, may be of invaluable importance to the designer, To this end, this paper presents a novel interactive visualization application, based on tree visualization, to understand the search dynamics of an evolutionary algorithm and to visualize where the optimum design points are located in the design space.

Keywords— **Design space exploration, multimedia MP-SoC design, visualization, evolutionary algorithms.**

I. Introduction

The complexity of today's embedded systems forces designers to start with modeling and simulating system components and their interactions in the very early design stages. It is therefore crucial to have good tools for exploring a wide range of design choices, especially during the early design stages, where the design space is at its largest. In the Sesame framework [1,2], a modeling and simulation environment is developed for the efficient design space exploration of multimedia embedded systems that are based on heterogeneous Multi-Processor System-on-Chip (MP-SoC) architectures. Models in Sesame are defined at a high level of abstraction and capture only the most important characteristics of the components in the system.

Because Sesame maintains independent application and architecture models and relies on the co-simulation of these two models in the performance evaluation of the composed embedded system, it is in need of an explicit mapping step which relates application tasks (i.e., processes) onto architecture components (typically processors). Each mapping decision taken in this step corresponds to a single point in the design space. In order to achieve an optimal design, the designer should ideally evaluate and compare every single point in this space. However, such exhaustive search quickly becomes infeasible, as the design space grows exponentially with the sizes of both application and architecture model components.

In general, to trim down an exponential design space into a finite set of points, which are more interesting (or superior) with respect to some chosen design criteria, design space pruning is used. In [3], the mapping decision problem is formulated as a multi-objective combinatorial optimization problem in which three criteria are considered: the processing time, power consumption and cost of the architecture. To solve this problem, an Evolutionary Algorithm (EA) has been used to achieve a set of best alternative mapping decisions under the aforementioned multiple criteria. To this end, it searches the design space over several iterations, called *generations*, during which the EA converges to an optimum.

As the searched design space still is vast, interpreting all evaluation data and understanding how the EA searched through or pruned the design space is also cumbersome. Such analysis is, however, essential to the designer as it provides insight into the "landscape" of the design space (e.g., indicating which design parameters are more important than others). Moreover, it allows tool designers to optimize their design space exploration algorithms to reduce search times.

To address these problems, this paper presents a novel interactive visualization application to understand how an evolutionary algorithm, such as presented in [3], searches the design space and where the optimum design points are located. In that respect, we visualize the design space as a tree and show how the EA searches through the design space over different generations. To give a rough feeling of how such a visualization looks like, Fig. 1 shows a screenshot of our visualization application.

Data visualization in the context of embedded systems design, and especially for understanding the process of design space exploration, is novel and hardly addressed in this domain. This paper, therefore, focuses on introducing and presenting the concepts for visualization of EA-based searches through vast design spaces. A follow-up paper will show how these visualization techniques can be actually applied to real design case studies. The remainder of this paper is organized as follows. Section II describes related work. In Section III, we explain how a design space can be modeled as a tree. Section IV introduces some techniques we provide in our visualization to be able to handle large trees. In Section V, we describe specific

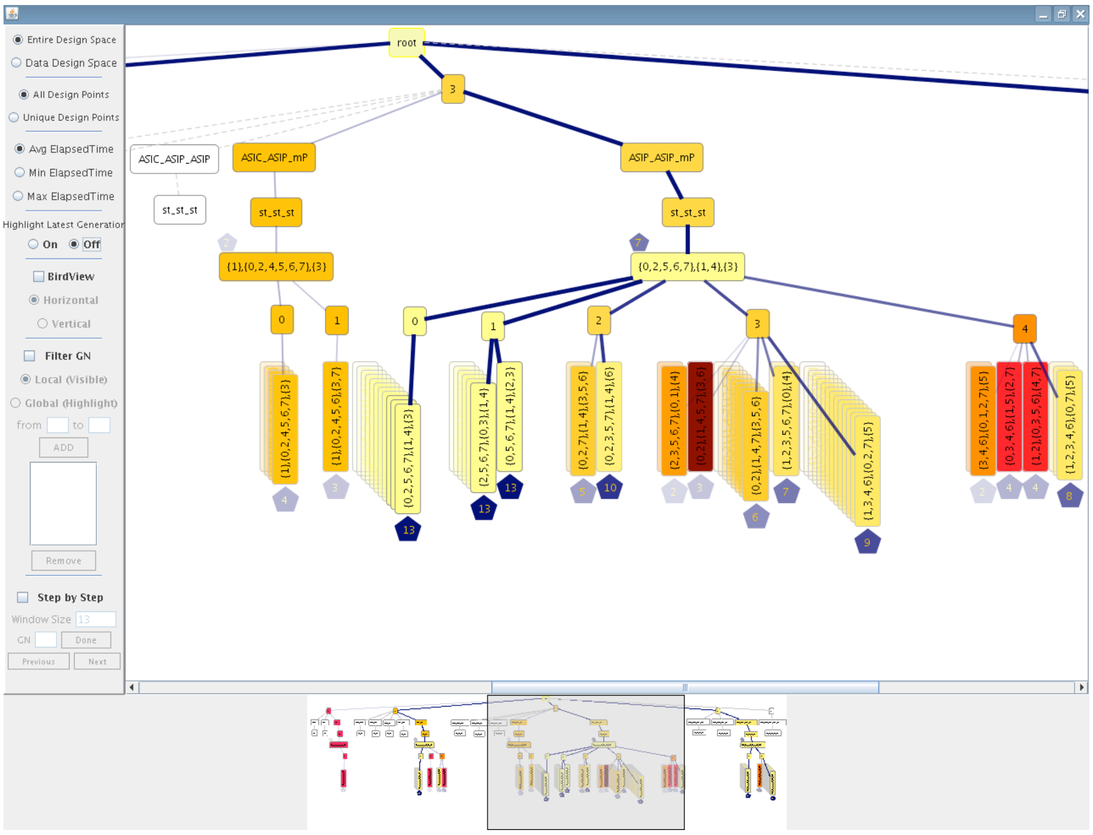

Fig. 1 Screenshot of the tree visualization

capabilities that have been added to the tree to show the search process of the EA in the design space. Section VI presents a small case study with a Motion-JPEG encoder application to reiterate the benefits of using visualization in the design exploration process. Finally, Section VII concludes the paper.

II. Related Work

In the field of computer architecture simulation, and especially in the area of system-level design space exploration, little research has been undertaken on visualization of simulation data in exploring alternative architectural solutions. Most of the visualization work in this area focuses on educational purposes (e.g., [4,5]), or only provides some basic support for the visualization of simulation results in the form of 2D (and sometimes 3D) graphs.

The work of [6,7] provides advanced and generic visualization support, but tries to do so for a wide range of computer system related information which may not necessarily be applicable to computer architecture simulations and in particular to design space exploration, with its own domain-specific requirements.

In [8], an interactive visual tool is presented to visualize the results from system-level design space exploration experiments. The simulation results are visualized using a coordinated, multiple-view approach which enables users to understand the information through different perspectives. It is possible to compare different design points with respect to various characteristics and gain more insight in the performance landscape of the design space. But this tool does not provide any insight in the searching process as performed by e.g. an EA. For example, there is no way to distinguish between the design points searched and evaluated in the different generations.

There are only a few research efforts addressing the visualization of EAs. Most visualization approaches are simple line plots. Commonly, a diagram of the population's fitness versus generation number is used to study the quality of the solutions over the generations [9]. Although such a diagram shows the improvements in the quality of the solutions considered during EA search process, it does not show anything about the properties of the solutions being searched, or the regions of the search space being explored.

More complex techniques have focused on how to display the progress of the EA in variables (parameters) space or objectives space [10,11]. Usually they use 2D or 3D plots in which one axis represents the generation numbers and the others show variables or objectives. Therefore, two separate views are needed to show the distribution of the population in both variables and objectives spaces. Furthermore, due to the large number of dimensions in practical problems, techniques such as Sammon Mapping [12] should be used to transform higher dimensional search spaces into smaller ones. Tree visualization, as presented in this paper, enables us to easily visualize more than three dimensions as well as to show variables and objectives in one view.

III. Tree Visualization

A. *Modeling the design space as a tree*

We visualize the design space as a tree in which each level shows one aspect of the design space such as the number of processors, type of processors, etc. Because of the human visual limitation to three dimensions, most of the commonly used techniques for visualization can visualize data sets dependent on two or three variables. Modeling the design space as a tree enables us to easily visualize multivariate data without the limitation on the number of variables as each level of the tree shows one dimension of design space. Furthermore, both categorical and numerical data types can be shown in one view. Variables which are more distinctive and are more important should be located at the higher levels of the tree.

A simple scenario is depicted in Fig. 2, illustrating a tree model of a small design space. In this case, the design space has three parameters: number of processors (maximum 2 processors), type of processors (one general purpose microprocessor (mP), two ASIPs and two Application Specific Integrated Circuits (ASICs)) and scheduling policy (static). These parameters are shown in the first three levels of the tree. Leaves show the mapping decision corresponding to a single point in the design space, i.e., which application process is mapped onto what resource. We should note that Fig. 2 is just for illustration purposes; in reality, we can deal with larger systems having more processors, processor types, etc.

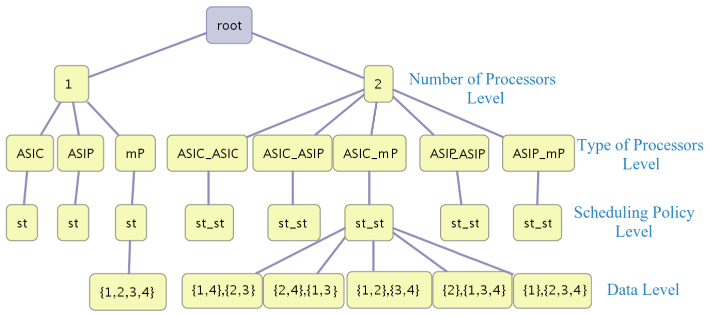

Fig. 2 Modeling design space as a tree

We refer to leaf nodes as *data nodes* and all other nodes as *parameter nodes*. Colors of the nodes can be used to show properties of the nodes. For example, in Section V.C we use the colors to visualize the performance. The order of the sibling nodes (nodes with the same parents) can also specify something. Here, they are just alphabetically ordered.

One of the drawbacks of the tree presented in Fig. 2 is that all the data nodes are at the same level and, as the number of design points increases, the tree becomes wider and more difficult to explore. Furthermore, it is hard to distinguish between sibling leaves. In the next section, we introduce a clustering mechanism to group the design points based on particular features.

B. *Clustering*

Clustering helps a user to easier interpret the design points and prevent the tree of becoming deformed and too wide. By clustering the design points, we distribute them at some levels instead of putting all of them at one level. Depending on the final aim of the clustering, the user can define a similarity criterion.

In this paper, the similarity criterion is *mapping distance*. We define the *mapping distance* between two mappings as the minimum number of application-task reallocations needed to transform the one mapping into the other. To give an example, let's assume two mappings {{1}, {2, 3, 4}} and {{1, 4}, {2, 3}}. Mapping {{1}, {2, 3, 4}} means that application task 1 is mapped onto the first processor and tasks 2, 3, 4 are mapped onto the second processor. The distance between these two mappings is one because one reallocation is needed to transform the first to the second mapping (task 4 should be executed on processor one instead of processor two).

We use the following algorithm to cluster the design points:

1. Put all design points (leaf nodes in Fig. 2) with the same parent in the same cluster.
2. Choose the design point with the smallest processing time as a cluster seed.
3. For each point in the cluster (except the cluster seed), calculate the mapping distance between the point and the cluster seed.
4. Put all points with the same mapping distance in the same cluster.

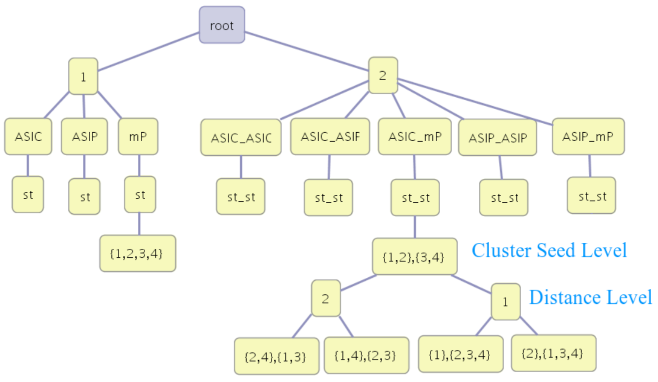

Fig. 3 Clustering design points

Fig. 3 shows the same tree as in Fig. 2 after applying the clustering algorithm, which added two extra levels to the tree: the cluster seed level which holds the seed and the distance level which contains the distance value. The rest of the design points are then children of the corresponding distance node. Therefore, our data nodes are now distributed over two levels: the cluster seed level and the leaves.

The design points with the same parent at the cluster seed level, have the same architecture components but the way that application tasks are mapped onto those components is different. This difference in mapping causes different processing times. Clustering these design points according to the mapping distance enables us to investigate the effects of mapping on the processing time. For example, design points might have a completely different processing time, but a very small mapping distance. This shows that the application tasks which are differently mapped are critical and mapping those tasks on the same processors may yield good performance. On the other hand, design points might have almost the same processing time, but with a large mapping distance. This may show that for the tasks mapped on different processors in these mappings, the type of the processors is not important.

IV. Handling Large Trees

Although clustering the design points may reduce the complexity of interpreting the tree data, it is still possible that the user encounters a very big tree. To tackle this problem, we have added several other capabilities to our visualization application, which are explained in the following subsections.

A. *Zoom*

In our visualization tool, among normal zoom features, we provide two extra zooming features for improved exploring: bird view and satellite view.

Bird view, depicted in Fig. 4, is a window moving with the mouse-pointer and shows a scene with a specified zoom factor and works like a magnifier. So, by simply hovering over the tree with the mouse-pointer, it is possible to zoom in on an area of interest to show its details. Bird view is helpful when the tree is big and the user wants to see the whole picture in one scene and still wants to view the details such as the labels of nodes.

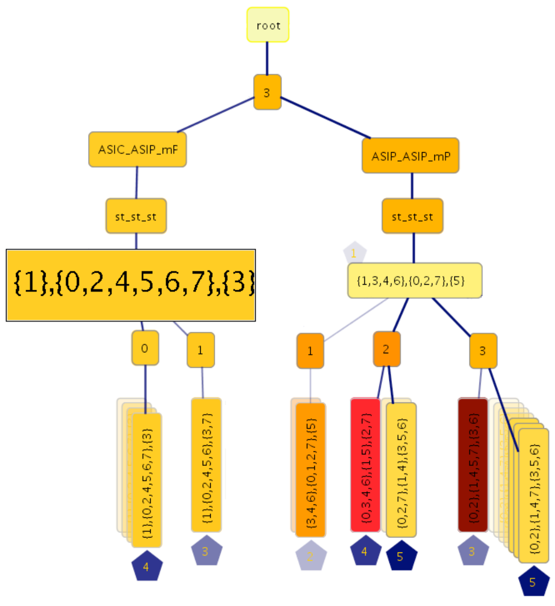

Fig. 4 Bird View

Satellite view, shown at the bottom of Fig. 1, gives an overall, smaller scale view of the entire scene, which allows the user to navigate quickly across the view. It also enables the user to zoom in on certain parts of the scene to focus on certain nodes in the tree without losing track of the position in the entire scene.

B. *Hiding nodes*

To reduce the number of visible nodes in the tree to make it smaller for better viewing, three options are provided in our tool:

1) *Hiding sub trees without data nodes*

Since some areas of the design space are not visited by the searching algorithm (e.g., they are not interesting enough so we do not have evaluation data for those parts), it should be possible to hide the sub trees of the nodes that have no data. This way, the user can focus on the sub trees which are more important and can easily see which parts of the tree are searched by EA. This is depicted in Fig. 5 and Fig. 6. Here, Fig. 6 has omitted the “empty” tree nodes, which are shown in Fig. 5 as the uncolored nodes.

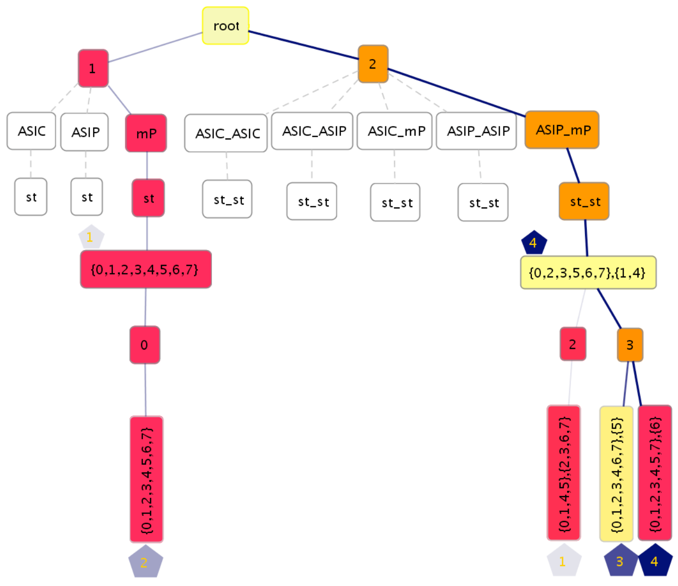

Fig. 5 The entire design space

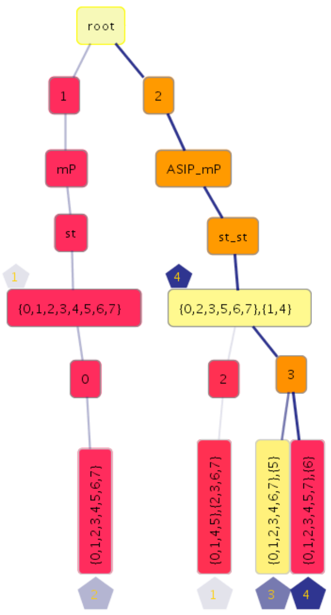

Fig. 6 The data design space only

2) *Hiding duplicate design points*

During the process of design space exploration using an EA, some design points that have a good performance may be regenerated in different generations. Therefore, there might be some duplicate design points in different generations. The user can select to see only the unique design points in the tree or he can select to see all design points generated by the EA. To distinguish copy design points from the main design points (generated in the most recent generation), the copies are stacked behind the corresponding main design points and are sorted by generation number.

Copies near the main design points are generated later than copies far from the main design points. For the main design points, the generation number is written inside a pentagon at the bottom of the node and for the copies the generation number is shown in their tooltip. A tooltip is a small pop-up window that appears when a user hovers the mouse pointer over an element. The pentagon will be described in more detail in Section V.B.

For better vision, the background color of the duplicate nodes becomes gradually lighter from front to the rear. To prevent the scene from becoming cluttered, the edges between duplicate nodes and their parents are invisible. Hiding (and showing) duplicate design points is illustrated in Fig. 7 and 8.

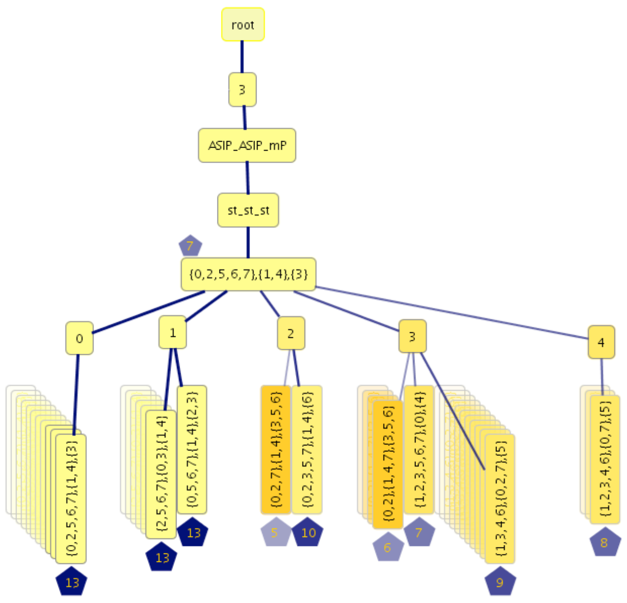

Fig. 7 All design points (including duplicate nodes)

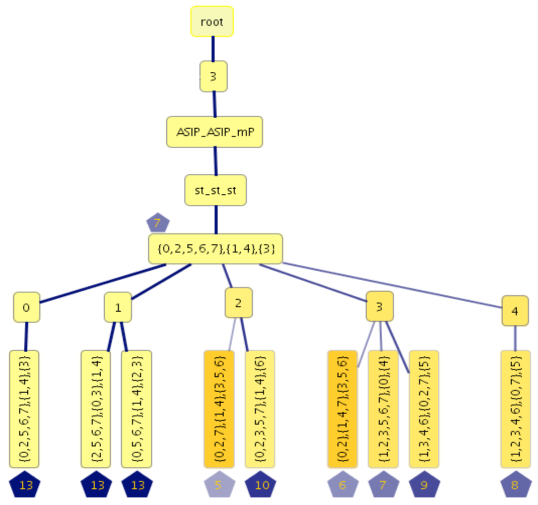

Fig. 8 Unique design points only

3) *Hiding sub trees*

If the user is not interested in some parts of the tree, then he is able to hide them in order to make the tree smaller and pay more attention to other nodes. By double clicking on a node, its sub tree becomes invisible and a blue triangle appears at the bottom of the node specifying that the children of the node are hidden. The size of the triangle represents the size of the sub tree. The bigger the triangle, the more nodes in the sub tree. By double clicking again, the sub tree becomes visible and the blue triangle is removed. Fig. 9 is the same as Fig. 7 in which the children of some nodes are hidden. As can be seen in Fig. 7, distance node ‘3’ has more children than distance node ‘2’. This is shown by a bigger blue triangle in Fig. 9.

It should be mentioned that by hiding a node, the entire tree will be redrawn, meaning that the empty space from that node will be used by the other nodes. We recalculate the location of visible nodes to optimize their fit to the screen.

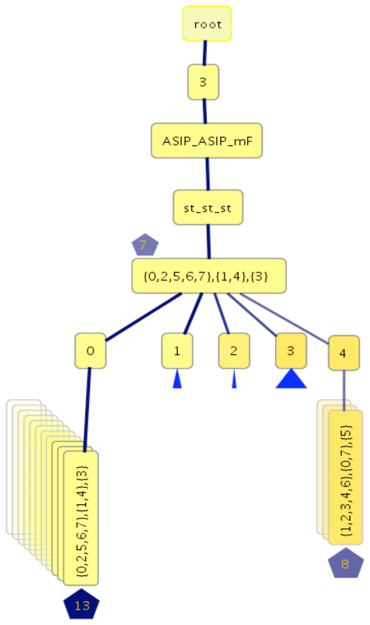

Fig. 9 Hidden sub trees

C. *Filtering*

Sometimes, the user wants to consider only the design points generated in some specific generation(s). For example, showing only design points generated in the three last generations or comparing design points in the three first generations with the three last generations, and so on. Therefore, we provide a filter option. The user can simply add (or remove) generation numbers to the list of generation numbers that need to be visualized. Two kinds of filtering are available: local and global. In local filtering, only design points with their generation numbers in the list are visible and the others become invisible. The parameter nodes with at least one child in the generation list are still visible. In global filtering, all nodes are visible but the nodes with generation numbers inside the list become highlighted. So the user can understand the position of the selected nodes in the entire tree.

In Fig. 10, design points for the three last generations are shown.

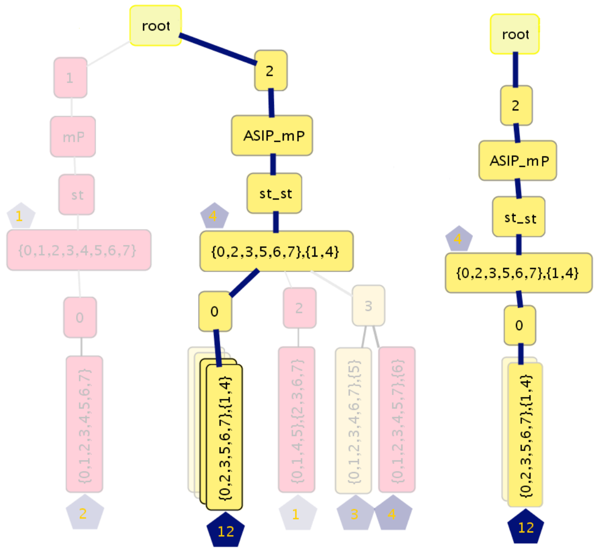

Fig. 10 (left) global filtering and (right) local filtering

V. Evolutionary Algorithm Specific Capabilities

Apart from all the aforementioned capabilities, we also added some more features specifically for studying the search as performed by the Evolutionary Algorithm (EA). These features visualize the EA generations step by step, showing which data has been generated when and where in the tree. Furthermore, they highlight the progress of the algorithm in the tree. The color of the nodes and edges, the texts inside the nodes, the thickness of the edges, etc. each show a different property of the design points.

Note that in our visualization we have the capability of showing multiple evaluation metrics, but in this paper we consider only one evaluation criterion, which is processing time. In the future, we will also consider multiple objectives.

A. *Showing Mapping Decisions*

As we mentioned before, in Sesame, each mapping decision that is evaluated, corresponds to a single point in the design space. In the tree, each data node's label shows the mapping decision. For example, if a data node is labeled with "{1,3}, {2,4}", it shows that application tasks one and three are executed on one processor and tasks two and four are executed on another one. The order of the task sets are in the same order as their parents at the type level. For example, if the label of the parent at the type level is "ASIP_mP", this means that tasks one and three are executed on an ASIP and tasks two and four are executed on a microprocessor (mP).

B. *Showing Generation Numbers*

It is important to know which design point is generated at what EA generation. To visualize this, we use a pentagon labeled with the generation number. For data nodes, at the cluster seed level, this pentagon is drawn on the upper left of the node and for the other data nodes (data nodes in leaves) it is at the bottom. To prevent the scene from being cluttered, the generation number of duplicate nodes is shown as a tooltip. If the mouse pointer goes over these nodes, the generation number is shown in a balloon. Sibling nodes at the data level are sorted by their generation numbers.

As the EA gradually converges to a (set of) optimum design point(s), we expect better design points in the later generations. To show this in the visualization, the background color of the pentagon gradually becomes darker from the first to the last generation. Furthermore, the color and thickness of the edges show the progress. The color of each edge is the same as the color of the pentagon with the highest generation number in its sub tree. Also, the edges with a higher generation number are thicker. As a result, the path from the root to the last generated data nodes is the darkest and thickest path. Edges that have no data node in their sub tree are shown in gray using dashed lines. Using this technique, it is very easy to identify which parts of the tree are searched in the later stages and which parts of the tree are not searched anymore. Fig. 11 illustrates how we show the generation numbers in our visualization application.

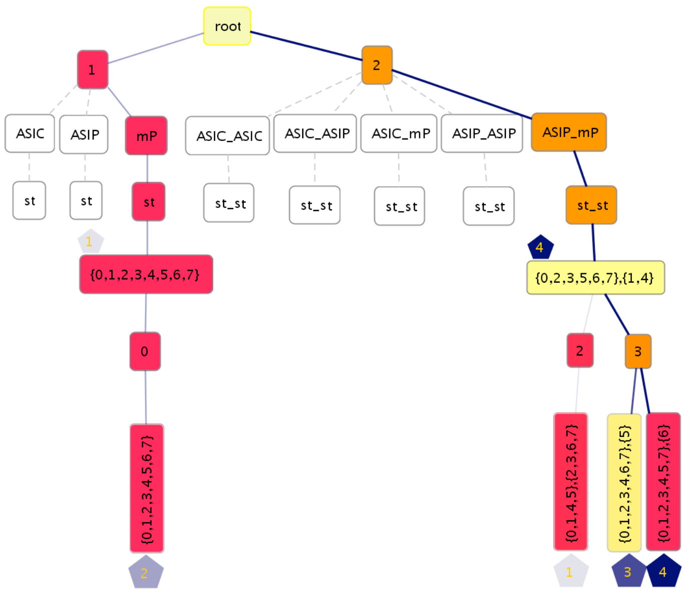

Fig. 11 EA Generations

C. *Showing the performance (processing time)*

In this paper, we consider only processing time as an objective and we use color coding to show it. Colors are varied from yellow to red with all color grades in between. Nodes with the lowest processing time are yellow and nodes with the highest processing time are red. In Fig. 11, the design point in the cluster seed level with two processors has a lower processing time than the design point in the cluster seed level with one processor.

Parameter nodes, however, do not represent single design points and therefore do not have the direct notion of processing time; only their child data nodes have. For this reason, there are three options to color the parameter nodes: based on the average, minimum, or maximum processing time of the child data nodes. In Fig. 11, the average processing time is chosen. The color of parameter nodes that have no data node in their sub trees is white.

D. *Step by step animation of the EA*

Our visualization application allows the user to trace the EA search process. The user can define a desired generation number and a window size. This means that the user wants to view the generated data ending at the desired generation number and within the window. For example, if the desired generation number is 10 and the window size is 4, the data generated in generations 7,8,9 and 10 are shown. Doing so, the tree is redrawn and contains only the design points generated up to the desired generation number and within the window. The user can also move forward and backward using next and previous buttons. For the aforementioned example, moving next means that the generations 8,9,10, and 11 are shown. Fig. 12 shows a snapshot of the first 4 EA generations using a step-by-step animation.

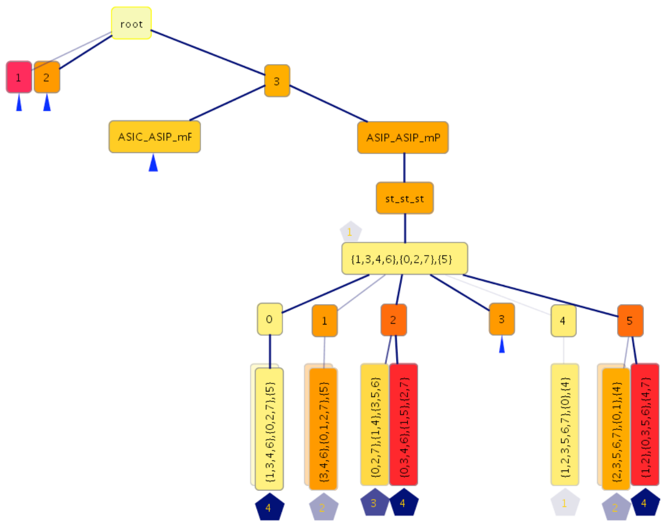

Fig. 12 Step by step animation (desired generation is 4)

When moving through the generations (i.e., replaying parts of, or even the entire, search process), it is important to know which data nodes are added in each generation (i.e., new design points, added to the population of the EA). To show this, the design points generated in the current generation are blinking. If a parameter node which had no data node in the previous step, receives its first data node in the current generation, it starts blinking as well.

For the hidden sub trees, in case of any data node addition, the blue triangle starts blinking and if the user is interested in viewing that sub tree, he can unhide it.

In Fig. 13, nodes with a green border are added in the fifth generation. A green triangle indicates that its parent has at least one new child in the current generation.

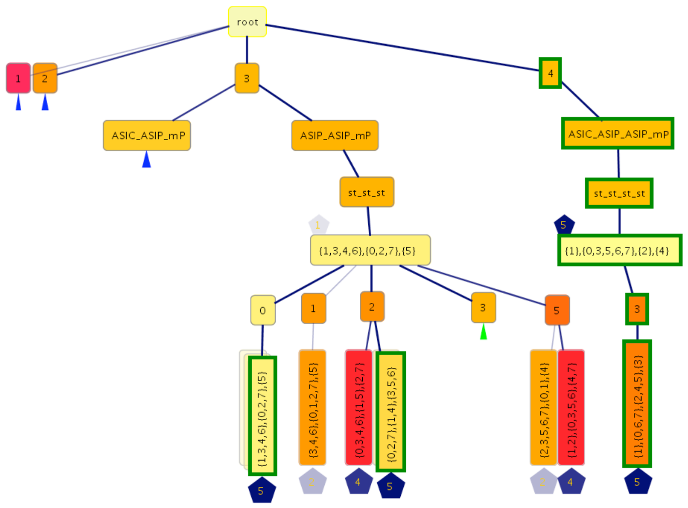

Fig. 13 Step by step animation (desired generation is 5)

VI. Case Study: Motion-JPEG Encoder

In this section, we present a small case study with a real application to reiterate the benefits of using visualization in the design space exploration process. This case study is by no means meant as an effort towards a detailed study of the design space exploration data for a particular design. As was mentioned earlier, a follow-up paper will provide a detailed and quantitative design space exploration study in which the presented visualization techniques are actually deployed.

In this case study, we map a Motion-JPEG (M-JPEG) encoder to an MP-SoC platform architecture that consists of five processors: a general-purpose microprocessor (mP), two ASIPs and two Application Specific Integrated Circuits (ASICs). Using a multi-objective evolutionary optimizer [3], we intend to find promising instances of this platform architecture that lead to a good mapping (in terms of processing time) of the M-JPEG encoder application.

It should be mentioned that the multi-objective evolutionary optimizer in [3] considers three criteria: processing time, power, and cost but in this case study we focus only on processing time, while we have fixed the values for the other criteria. In the future, we will extend our visualization to consider the other criteria as well.

Fig. 14 shows a snapshot of the visualization of the M-JPEG case study. Just by looking at the picture, one can easily draw conclusions with respect to the following issues:

- Which are the parts of the design space that are not searched at all (no design point is generated there). As we have mentioned before, nodes with a white color have no data. As can be seen in Fig. 14, there are e.g. no design points (i.e., MP-SoC platforms) with five processors or two ASICs.
- Which are the parts of the design space that are searched more often by the EA. In these areas, the tree provides more design points and the sub trees of the corresponding nodes are bigger.
- Which are the design points that are frequently re-generated. Design points with a bigger stack behind them are re-generated more often during the EA's search. As can be seen in Fig. 14, the better performing design points (node color is yellow) are repeated more than design points with a poor performance (node color is red).
- The number of unique design points generated by the EA. Duplicate design points are stacked behind their main design point.
- Which are the parts of the design space that are searched in later generations of the EA. The edges in these paths are thicker and darker. Subsequently, the designer can gain insight into the characteristics of design points in these later generations that are close to the optimum (i.e., what characteristics make a design point to be a good one?).
- Which are the parts of the tree that contain the better performing design points as indicated by the color coding.

Clearly, such visual analysis of the design space exploration process, as described above, allows the designer to better understand the design space he is dealing with. Moreover, it may help to improve the EA search algorithm (e.g., using domain specific knowledge) to more quickly converge to a global optimum.

VII. Conclusion

In this paper, we presented a visualization application that helps designers to understand the search behaviour in EA-based design space exploration as well as to gain insight into the landscape of the design space. That is, understanding the characteristics of design points with good performance and the relationships between design parameters and their effects on performance. In our application, we provide several capabilities to be able to handle large design spaces and to represent the progress of the EA during the process of design space exploration. We have also briefly illustrated the benefits of such visualization using a Motion-JPEG encoder case study.

References

[1] 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, 2007, available online: DOI 10.1155/2007/82123.

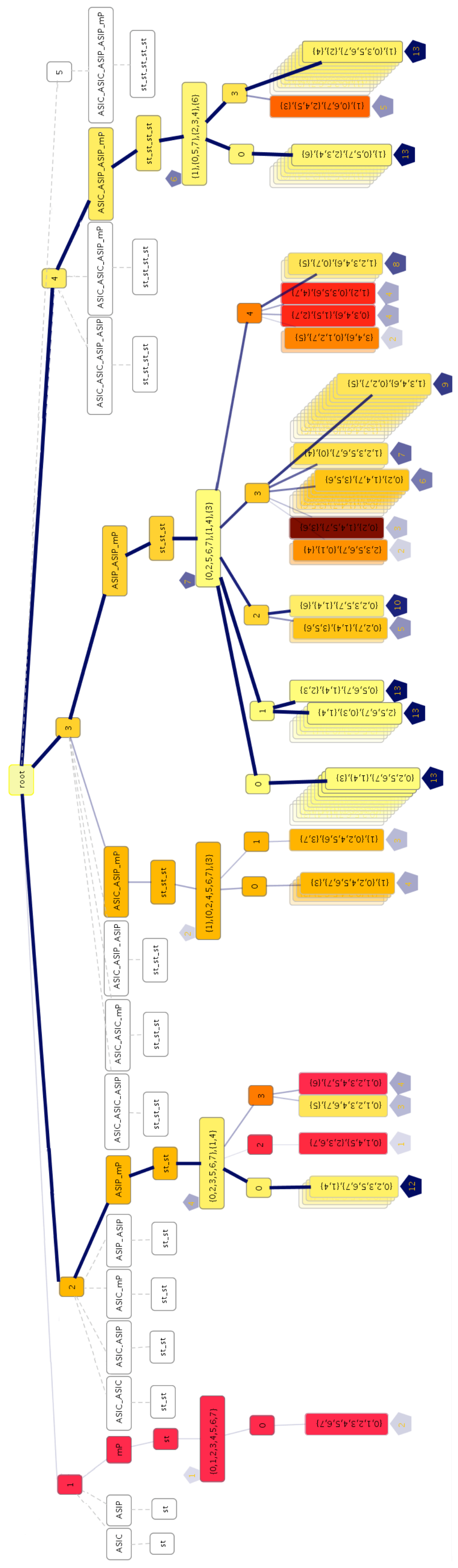

Fig. 14 Visualization of the M-JPEG case study

[2] A.D. Pimentel, C. Erbas, and S. Polstra. A Systematic Approach to Exploring Embedded System Architectures at Multiple Abstraction Levels, IEEE Transactions on Computers, vol. 55, no. 2, pp. 99-112,(2006).

[3] 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 Transactions on Evolutionary Computation, pp. 358-374, Vol. 10 (No. 3), June 2006.

[4] P. Marwedel, B. Sirocic. Multimedia components for the visualization of dynamic behavior in computer architectures, in the Proc. of the Workshop of Computer Architecture Education, 2003.

[5] C. Yehezkel, W. Yurcik, M. Pearson, D. Armstrong. Three simulator tools for teaching computer architecture: Easycpu, little man computer, and rtlsim, Journal on Educational Resources in Computing (JERIC), vol. 1, no. 4, pp. 60-80, 2001.

[6] R. Bosch, et al, Rivet: A flexible environment for computer systems visualization, SIGGRAPH Computer Graphics, vol. 34, no. 1, pp. 68-73, 2000.

[7] R.P. Bosch. Using Visualization to Understand the Behavior of Computer Systems, PhD thesis, Stanford University, 2001.

[8] T. Taghavi, A. D. Pimentel, and M. Thompson. Visualization of Computer Architecture Simulation Data for System-level Design Space Exploration, in the Proc. of Int. Symposium on Systems, Architectures, MOdeling and Simulation (SAMOS '09), July 2009.

[9] E. Hart and P. Ross. GAVEL - A New Tool for Genetic Algorithm Visualization, IEEE Trans on Evolutionary Computation, Vol. 5, No. 4, pp. 335-348, August 2001.

[10] H. Pohlheim. Visualization of evolutionary algorithms — set of standard techniques and multidimensional visualization, in the Proceedings of the 1999 Genetic and Evolutionary Computation Conference GECCO'99, Morgan Kaufmann, Los Altos, CA, 1999, pp. 533–540.

[11] T.D. Collins. Applying software visualization technology to support the use of evolutionary algorithms, Journal of Visual Language and Computing 14 (2003), 123-150.

[12] Sammon, J. W. jr. A Nonlinear Mapping for Data Structure Analysis. IEEE Transactions on Computers, vol. C-18, no. 5, pp. 401-409, 1969.