Design space pruning through hybrid analysis in system-level design space exploration

Piscitelli, R.; Pimentel, A.D.

DOI
10.1109/DATE.2012.6176600

Publication date
2012

Document Version
Final published version

Published in
Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012

Link to publication

Citation for published version (APA):
Piscitelli, R., & Pimentel, A. D. (2012). Design space pruning through hybrid analysis in system-level design space exploration. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012* (pp. 781-786). IEEE. https://doi.org/10.1109/DATE.2012.6176600

Design Space Pruning through Hybrid Analysis in System-level Design Space Exploration

Roberta Piscitelli and Andy D. Pimentel
Computer Systems Architecture group, Informatics Institute, University of Amsterdam, The Netherlands
Email: {r.piscitelli,a.d.pimentel}@uva.nl

***Abstract*—System-level design space exploration (DSE), which is performed early in the design process, is of eminent importance to the design of complex multi-processor embedded system architectures. During system-level DSE, system parameters like, e.g., the number and type of processors, the type and size of memories, or the mapping of application tasks to architectural resources, are considered. Simulation-based DSE, in which different design instances are evaluated using system-level simulations, typically are computationally costly. Even using high-level simulations and efficient exploration algorithms, the simulation time to evaluate design points forms a real bottleneck in such DSE. Therefore, the vast design space that needs to be searched requires effective design space pruning techniques. This paper presents a technique to reduce the number of simulations needed during system-level DSE. More specifically, we propose an iterative design space pruning methodology based on static throughput analysis of different application mappings. By interleaving these analytical throughput estimations with simulations, our hybrid approach can significantly reduce the number of simulations that are needed during the process of DSE.** [1]

I. Introduction

In recent years, platform based design has become the predominant design paradigm in the area of heterogeneous multiprocessor system-on-chip (MPSoC) design [20]. In contrast to more traditional design paradigms, platform based design shortens design time by eliminating the effort of the low-level design and implementation of system components. A platform based design environment typically consists of a fixed, parameterizable platform or a set of (parameterizable) components that can be combined in specific ways to compose a platform. The parameters make it possible to adjust platforms and individual components to the required application domain and platform design requirements. However, as the number of possible system candidates increases exponentially with the number of parameters, traditional design space exploration methods fall short. This has prompted increasing research effort focusing on efficient exploration techniques to identify those parameters that result in an optimal system.

System parameters are often considered by type: e.g., the number and type of processors, the size of the main memory or the mapping of application tasks to architectural resources. A multi-dimensional design space can be constructed by using each type as an axis of the design space (the so-called parameter space). The design criteria for a system can usually be translated to one or multiple objectives, e.g., power consumption, system performance or cost. The parameter space maps onto the objective space by associating objective values to each point in the parameter space. If the complete objective space were given, then a designer could easily select those system candidates that meet the design requirements or that are optimal in some pre-defined way. In practice, however, it is infeasible to obtain a representation of the objective space that is both accurate and complete.

Methods for evaluating a single design point in the design space roughly fall into one of three categories: 1) measurements on a (prototype) implementation, 2) simulation based measurements and 3) estimations based on some kind of analytical model. Each of these methods has different properties with regard to evaluation time and accuracy. Evaluation of prototype implementations provides the highest accuracy, but long development times prohibit evaluation of many design options. Analytical estimations are considered the fastest, but accuracy is limited since they are typically unable to sufficiently capture particular intricate system behavior. Simulation-based evaluation fills up the range in between these two extremes: both highly accurate (but slower) and fast (but less accurate) simulation techniques are available. This trade-off between accuracy and speed is very important, since successful design space exploration (DSE) depends both on the ability to evaluate a single design point as well as being able to efficiently search the entire design space. Current DSE efforts typically use simulation or analytical models to evaluate single design points together with a heuristic search method [6] or statistical techniques [8], [21], [23] to search the design space. These DSE methods search the design space using only a finite number of design-point evaluations, not guaranteeing to find the absolute optimum in the design space, but they reduce the design space to a set of design candidates that meet certain requirements or are close to the optimum with respect to certain objectives.

In this paper, we focus on mapping DSE, where mapping involves two aspects: 1) allocation and 2) binding. Allocation deals with selecting the architectural components in the MPSoC platform architecture that will be involved in the execution of the application workload (i.e., not all platform components need to be used). Subsequently, the binding specifies which application task or application communication is performed by which MPSoC component. As mentioned above, current state-of-the art in this particular domain is mostly based on the use of either simulations or analytical models

[1] 978-3-9810801-8-6/DATE12/©2012 EDAA

to evaluate mappings, where simulative approaches typically prohibit the evaluation of many design options due to the higher evaluation performance costs and analytical approaches suffer from accuracy issues. In this paper, we propose a hybrid approach which combines simulation with an analytical model to prune the design space in terms of application mappings that need to be evaluated using simulation. To this end, we study the analytical estimation of the expected throughput of an application given a certain architectural configuration and application-to-architecture mapping. In the majority of the search iterations of the DSE process, the throughput estimation avoids the use of simulations to evaluate the design points. However, since the analytical estimations may in some case be less accurate, we once in while interleave the analytical estimations with simulative evaluations in order to ensure that the DSE process is steered into the right direction. As we will demonstrate, the resulting *hybrid* technique yields significant efficiency improvements while still producing similar solutions in terms of quality as compared to simulation-based DSE.

The remainder of the paper is organized as follows. The next section briefly describes how the throughput analysis is combined with simulation in the DSE process. Section 3 gives an overview of the throughput analysis methodology. Section 4 presents a number of experiments in which we compare our hybrid approach against DSE using only system-level simulations or analytical estimations. In Section 5, we describe related work, after which Section 6 concludes the paper.

II. Combining throughput analysis and simulation

To evaluate design points during system-level DSE by means of simulation, we deploy the Sesame simulation framework [19]. Sesame allows for rapid performance evaluation of different MPSoC architecture designs, application to architecture mappings, and hardware/software partitionings with a typical accuracy of 5% compared to the real implementation [19], [16]. It applies separate application and architecture models, together with an explicit mapping step to map application models onto an architecture model. This mapping step has been implemented using trace-driven co-simulation of the application and architecture models. In this approach, traces with computational and communication events generated by an application model and consumed by an architecture model are an abstract representation of the workload imposed on the architecture.

In Figure 1, the entire DSE framework is shown. We adopt a hybrid approach where Sesame simulations are interleaved with analytical throughput analysis. The throughput analysis is based on the application graph, the individual task workloads and the mapping. It is used to quickly predict the performance consequences of different design points as represented by the application mapping on the underlying architecture. As these fast analytical evaluations are interleaved with the slower simulative evaluations in a way such that most evaluations are performed analytically, this approach significantly improves the efficiency of the DSE process. Consequently, this would allow for searching a much larger design space.

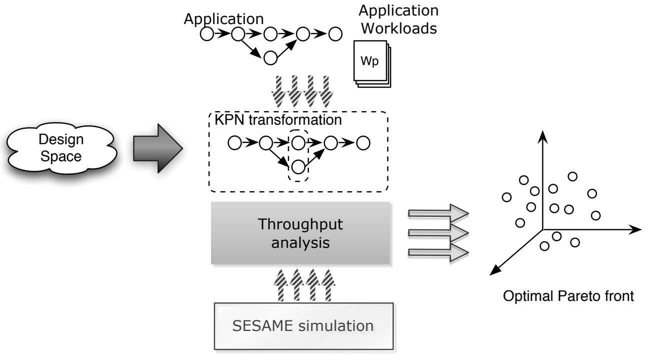


Fig. 1. Driving experiments with the expected throughput.

The application is represented as a Kahn Process Network (KPN) [9]. As will described in the next section, before performing the throughput analysis, we need to perform some transformations to the application graph of the KPN in order to take into account mapping decisions. The subsequent throughput analysis – performed on the transformed KPN – should be fast and capable of correctly capturing the throughput trend for different mappings. The analysis requires the process workloads W_{P_i} as a parameter for the throughput modeling. The workload W_{P_i} of an application process P_i denotes the number of time units that are required to execute a single invocation of the process, i.e., the pure computational workload, excluding the communication. It should be provided by the designer who can obtain it, for example, by executing the process once on the target platform, or by using an instruction set simulator.

As will be shown later on, the analytical throughput model may encounter accuracy problems when the application graph is cyclic. To correct such errors during DSE, we interleave the throughput estimation with real simulations, according to the value of a function ϕ, which can be set to 1 if a cycle occurs or every k generations in the DSE process.

In our DSE framework, we use the widely-used NSGAII genetic algorithm [3] to actually search through the mapping design space. This results in a hybrid DSE method with the following steps, as shown in Figure 1:

1) Perform an initial model calibration and generate the application workloads W_{P_i}, as explained in [16]. This is a one-time effort, and the same for both the simulation model and analytical throughput model.
2) Generate an initial population of unique mappings.
3) Transform the application KPN according to the mappings in the population and build the corresponding merged KPNs (as will be explained in the next section).
4) Perform the static throughput analysis for the merged KPN graphs and identify the best mappings based on the highest estimated throughput.
5) In case of $\phi = 1$, we interleave the throughput analysis with real simulation, in order to correct the ranking in the NSGAII evolutionary algorithm.
6) Verify the stopping criterion. If the mapping population

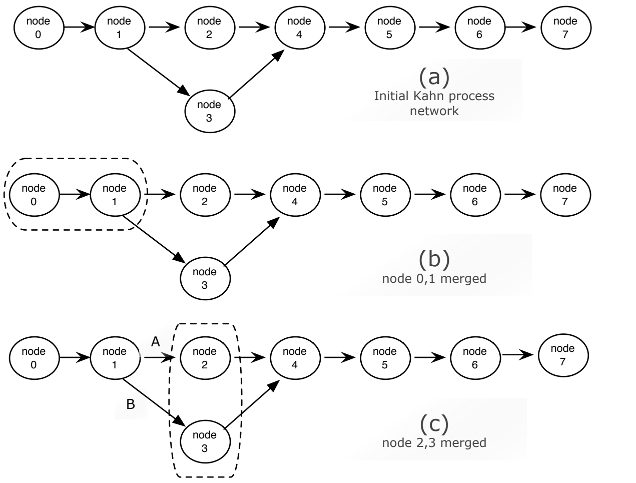
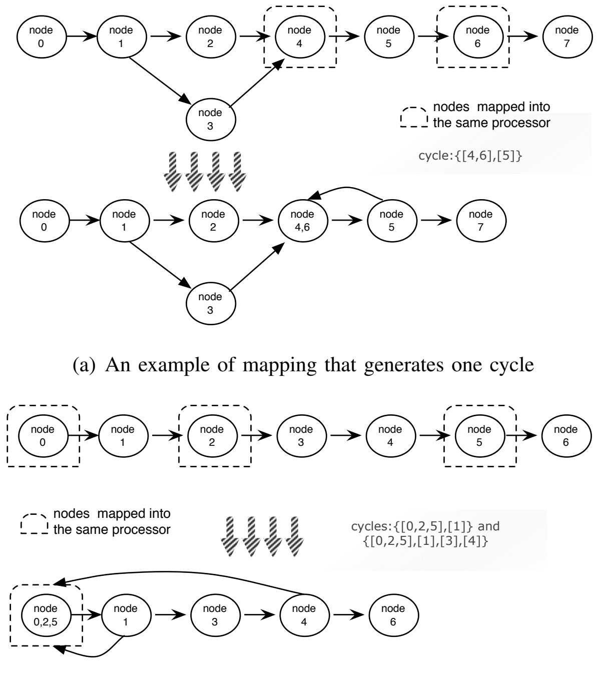


Fig. 2. Process merging in an example Kahn Process network.

within the NSGAII algorithm remains unchanged or a maximum number of iterations has been performed, the algorithm stops. Otherwise, change the mapping population using NSGAII's genetic operators, and restart from the third step.

III. Modeling application mappings as merged Kahn Process Networks

Applications in our DSE framework are modeled using KPNs [9], in which parallel processes communicate with each other via unbounded FIFO channels. In the Kahn paradigm, reading from channels is done in a blocking manner, while writing is non-blocking. Starting from a KPN, to perform throughput analysis one needs to take into account the mapping since the performance is mapping dependent. As we want to perform the throughput analysis only at the KPN level, we have to represent the mapping inside the KPN itself. To this end, we use *merging transformations* on the KPN to reflect the mapping of the different processes. Consequently, if two processes are mapped onto the same architectural component, they are merged into a single process in the KPN, as is illustrated in Figure 2. Figure 2(a) shows the initial example KPN consisting of eight processes. Performing throughput analysis on this KPN assumes that each process is mapped onto a different processor and each KPN channel is mapped onto a unique communication memory in the MPSoC (i.e., all the connections are point-to-point connections). The KPNs in Figures 2(b) and 2(c) subsequently reflect the decisions that, respectively, KPN processes 0,1 and 2,3 are mapped onto a single processor. Mapping multiple KPN tasks onto one processor allows for MPSoC implementations with less processing and communication components, i.e. with reduced implementation cost, but at the cost of potentially additional execution overhead. For example, in case of a homogeneous MPSoC and a KPN model in which processes exchange data tokens of uniform size, the performance of such mapping *in terms of throughput* can only be the same or lower (so never higher) than the performance of a mapping in which each task is mapped onto a different processor [13]. Subsequently, to assess the performance of a mapping decision, we perform throughput analysis on the transformed KPN.



(a) An example of mapping that generates one cycle

(b) An example of mapping that generates two cycles

Fig. 3. Transformation into a cyclic KPN.

A. Process Throughput and Throughput Propagation

Our throughput analysis is based on and extends the work presented in [13], in which the solution approach for the overall KPN throughput modeling relies on calculating the throughput τ_{P_i} of a process (i.e., node) P_i for all KPN processes and propagation of the lowest process throughput to the sink process. Here, we use a depth first search to determine the order of the processes for propagating throughputs. For a process P_i, the propagation consists of selecting either the aggregated incoming FIFO throughput $\tau_{F_{aggr,P_i}}$ or the isolated process throughput $\tau_{P_i}^{iso}$.

The isolated throughput $\tau_{P_i}^{iso}$ is the throughput of a process P_i when it is considered to be completely isolated from its environment. This means that the isolated process throughput is determined only by the workload W_{P_i} of a process and the number of FIFO reads/writes per process execution provided that no blocking occurs:

$$\tau_{P_i}^{iso} = \frac{1}{W_{P_i} + x \cdot C^{Rd} + y \cdot C^{Wr}} \tag{1}$$

where x and y denote how many FIFOs are read and written per process execution and C^{Rd} and C^{Wr} the performance costs for reading/writing a token from/to a FIFO channel. The throughput of a FIFO-channel f is determined by the throughput of the processes accessing it:

$$\tau_f = min(\tau_f^{Wr}, \tau_f^{Rd}) \tag{2}$$

Subsequently, the throughput τ_{P_i} of a process P_i is determined by either the throughput of the FIFOs from which process P_i receives its data or by the computational workload of the process itself, i.e., $\tau^{iso}_{P_i}$. For merged KPN processes, the incoming FIFO throughput is the aggregated throughput of the merged channels and the isolated throughput is calculated using the aggregated computational workloads. Consequently, the throughput associated to each process in an acyclic KPN graph is computed as:

$$\tau_{P_i} = min(\tau_{F_{aggr,P_i}}, \tau^{iso}_{P_i}) \tag{3}$$

For example, for the merged processes 2,3 in Figure 2(c), $\tau_{F_{aggr,P_{2,3}}} = \tau_{f_a} + \tau_{f_b}$ and $\tau^{iso}_{P_{2,3}} = \frac{1}{W_{P_2}+W_{P_3}+2\cdot C^{Rd}+2\cdot C^{Wr}}$.

B. Handling cycles

It is possible that the aforementioned merging transformation to account for mapping decisions might introduce cycles in the transformed KPN. As shown in Figure 3(a), if processes 4,6 are mapped onto the same processor, this results in a cycle containing process 5 and the merged process $4, 6$. In Figure 3(b), processes 0, 2 and 5 are mapped to the same processor, resulting in a KPN with two cycles. Cycles in a KPN are responsible for sequential execution of some of the processes involved in the cycle. The sequential execution can vary from a single initial delay to a delay at each execution of some of the processes. For accurate throughput modeling, these cycles must be taken into account. To the best of our knowledge, current research on throughput analysis of KPNs has not addressed the handling of cycles. In [13], the authors only consider acyclic KPN graphs. A preliminary process throughput analysis in case of dataflow loops is suggested in [15] in terms of *mapping rules*, but the proposed rules have never been elaborated nor verified. Based on this approach, we conservatively approximate the isolated throughput of a process P_i that is member of a cycle by:

$$\tau^{iso}_{cyc_{P_i}} = \frac{1}{\sum_{P_j \in Cycle} \tau^{iso}_{P_j}} \tag{4}$$

From equation 4, it is clear that the isolated throughput of a cycle is lower than the regular isolated throughput ($\tau^{iso}_{P_i}$) of any of the processes involved in the cycle. It also implies that the isolated throughput of a cycle can be lower than the isolated throughput of the bottleneck process. This is an important observation because, in such a case, the throughput of the cycle will determine the overall KPN performance. To conclude, the throughput associated to each process P_i will be computed as:

$$\tau_{P_i} = min(\tau^{iso}_{cyc_{P_i}}, \tau_{F_{aggr,P_i}}, \tau^{iso}_{P_i}) \tag{5}$$

For example, in Figure 3(b) two cycles are generated due to the KPN transformation. In this case, we assume that the resulting $\tau^{iso}_{cyc_{P_i}}$ for a process P_i would be

$$\tau^{iso}_{cyc_{P_i}} = min(\tau^{iso}_{cyc_{P_i}}(1), \ldots, \tau^{iso}_{cyc_{P_i}}(n)) \tag{6}$$

where $\tau^{iso}_{cyc_{P_i}}(1), \ldots, \tau^{iso}_{cyc_{P_i}}(n)$ are all the throughputs of the cycles involving process P_i.

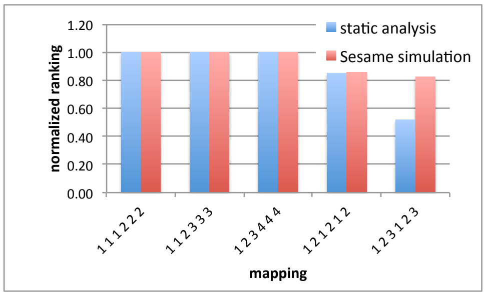


Fig. 4. Normalized mapping ranking for the MJPEG application using static analysis and simulation.

C. A hybrid DSE approach

The analytical throughput analysis may present some inaccuracies when there are cycles introduced in the transformed KPN, especially when there are many and/or complex cycles. To demonstrate this, please consider Figure 4. Assuming a Motion-JPEG (MJPEG) encoder application and a 4-processor target MPSoC platform, this figure shows the normalized performance ranking of five mappings when evaluating the mappings using Sesame simulations (in red) or using our throughput model (in blue). Here, we apply the following notation for the mappings as specified on the horizontal axis: we have six application tasks and for each task we assign the identifier of the processor to which the task is mapped. For instance, the mapping $\{1, 1, 1, 2, 2, 2\}$ indicates that tasks 0,1 and 2 are mapped to processor 1, while tasks 3,4 and 5 are mapped to processor 2. Evidently, the normalized ranking of the mappings for the MJPEG application is correct most of the times. However, in correspondence to the transformations to the KPNs to represent a certain mapping, which generates cycles (i.e., mappings $\{1, 2, 1, 2, 1, 2\}$ and $\{1, 2, 3, 1, 2, 3\}$ in Figure 4), the estimation is sometimes too pessimistic. This could imply that these design points would be excluded during the DSE phase, possibly leading to a sub-optimal solution. For this reason, we introduce a hybrid simulation and analytical approach to compensate for these estimation errors. However, as the probability of getting a transformed KPN with a cycle increases exponentially with the complexity of the topology of the KPN and the number of processors in the target platform, for efficiency reasons our DSE framework also allows for interleaving simulations with a pre-defined frequency instead of correcting the DSE every time a cycle occurs.

IV. Experimental results

Using analytical throughput estimation as fitness function during DSE can yield significant efficiency improvements. Figure 5 shows the wall-clock times for a DSE experiment, using a NSGAII genetic algorithm, for four multimedia applications: an Mp3 decoder, a H264 decoder, a Motion JPEG (MJPEG) encoder, and a Sobel filter for edge detection in images. The underlying platform for the analysis is a heterogeneous 8-processor MPSoC, which can contain different flavors of MIPS, ARM and StrongARM processors. The curves labeled with an "S_" prefix show the DSE times when only using

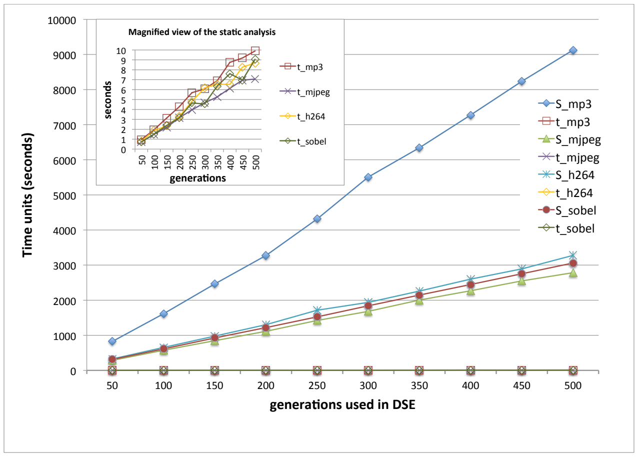


Fig. 5. DSE times using different methods.

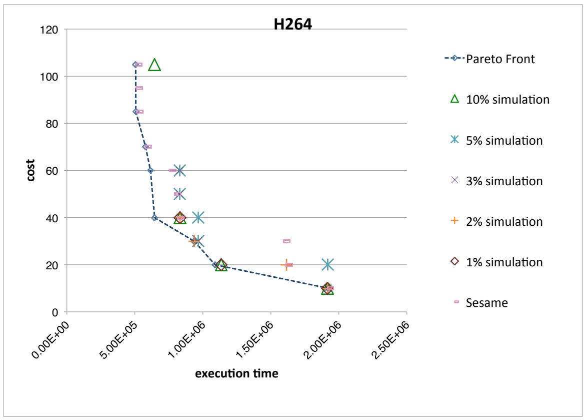


Fig. 6. H264 design space exploration.

TABLE I
THE AVERAGE RESULTS OF HYBRID METHOD FOR H264 DECODER, MJPEG ENCODER, MP3 DECODER AND THE SOBEL FILTER.

H264	10% sim.	5% sim.	3% sim.	2% sim.	1% sim.	Sesame
HV	0.914	0.777	0.817	0.767	0.906	0.903
∇	0.692	0.312	0.338	0.283	0.242	0.956
σ_{mst}	0.180	0.227	0.337	0.116	0.049	0.136
Mp3	10% sim.	5% sim.	3% sim.	2% sim.	1% sim.	Sesame
HV	0.643	0.552	0.643	0.694	0.430	0.672
∇	0.349	0.245	0.139	0.088	0.037	0.554
σ_{mst}	0.344	0.554	0.425	0.469	0.475	0.276
MJPEG	10% sim.	5% sim.	3% sim.	2% sim.	1% sim.	Sesame
HV	0.760	0.6103	0.6838	0.8744	0.6839	0.970
∇	0.600	0.517	0.137	0.205	0.133	0.915
σ_{mst}	0.153	0.112	0.131	0.239	0.141	0.200
Sobel	10% sim.	5% sim.	3% sim.	2% sim.	1% sim.	Sesame
HV	0.99	0.848	0.875	0.99	0.99	0.729
∇	0.87	0.86	0.32	0.324	0.324	0.990
σ_{mst}	0.01	0.370	0.245	0.008	0.008	0.426

Sesame simulations, as a function of the number of generations used in NSGAII. The curves with a "t_" prefix show the results of exclusively using static throughput estimation during DSE. Clearly, the DSE based on analytic throughput analysis can be three orders of magnitude faster than simulation-based DSE.

In Figure 6, we present the resulting Pareto points from a DSE experiment with the H264 application and 100 search iterations by NSGAII, when using architecture cost and execution time as optimization objectives. The graph shows the reference Pareto front, which is obtained by unifying all Pareto-optimal solutions from 10 runs of Sesame-only DSE. Moreover, the graph depicts the Pareto optimal solutions found by a single run of Sesame-only DSE (indicated as Sesame) and our hybrid method using different frequencies of simulations. The curves denoted by $k\%$ $simulation$ show the Pareto points produced by our hybrid method using simulation for $k\%$ of the total generations (e.g., 2% $simulation$ implies that Sesame is used in 2 search generations out of the 100). In this hybrid method, the final solution set is simulated with Sesame to allow for comparison with the reference Pareto front. In case of a low frequency of simulations, the solution space is clustered in the lower side of the Pareto curve, corresponding to those solutions in which the application is mapped to a few processors. This is due to the fact that the probability of obtaining a transformed KPN with a cycle increases exponentially with the complexity of the topology of the KPN and the number of processors in the target platform. As explained before, the static analysis may present some inaccuracies when there are cycles in the transformed KPN, possibly excluding them from the final solution set. However, if we increase the simulation frequency in the DSE, the extent (i.e., spread) of the solution space significantly improves.

To quantify the quality of the obtained Pareto fronts, Table I shows, for each of the studied applications, how close the found solutions are to the reference Pareto front, the spread of the solutions along the Pareto front, and the distribution of the solutions. To this end, we use three different metrics: the hypervolume, ∇ metric and σ_{MST} metric. The hypervolume (HV) [24] indicates the closeness of the solution set to the reference Pareto front. The normalized ∇ metric [5] measures the spread of solutions. It refers to the area of a rectangle formed by the two extreme solutions in the objective space, thus a bigger value spans a larger portion and therefore is better. For measuring the distribution of solutions in a Pareto optimal set, we use the σ_{MST} metric [22]. A smaller value indicates that the distribution of the solutions is closer to the uniform distribution and thus is better. The results in Table I are averages, where every DSE experiment has been performed five times. From Table I, it appears that the hypervolume is not dependent on the frequency of simulations. That is, compared to simulation-only DSE, our hybrid approach can yield similar Pareto fronts in terms of closeness to the reference front. The extent of solutions (∇) clearly can be improved by increasing the simulation frequency, as explained above. The distribution of solutions (σ_{MST}) often is close to uniform for low simulation frequencies but this is due to the small number of solutions in the corresponding Pareto fronts. For higher simulation frequencies, the distribution of solutions does not appear to be dependent on the frequency. The results from Table I indicate that our hybrid DSE is a promising technique, yielding solutions similar in terms of quality as compared to simulation-based DSE but at a fraction of the execution time.

V. Related Work

Current state-of-the-art in system-level DSE often deploys population-based Monte Carlo-like optimization algorithms like hill climbing, simulated annealing, ant colony optimization, or genetic algorithms. By adjusting the parameters, or by modifying the algorithm to include domain-specific knowledge, these algorithms can be customized for different DSE problems to increase the effectivity of the search [17], [2]. Another promising approach is based on meta-model assisted optimizations, which combines simple and approximate models with more expensive simulation techniques [12], [18], [4], [1], [11]. In [4], the authors use meta-models as a pre-selection criterion to exclude the less promising configurations from the exploration. In [11], meta-models are used to identify the best set of experiments to be performed to improve the accuracy of the model itself. In [12], an iterative DSE methodology is proposed exploiting the statistical properties of the design space to infer, by means of a correlation-based analytic model, the design points to be analyzed with low-level simulations. The knowledge of a few design points is used to predict the expected improvement of unknown configurations. However, these meta-models usually have design space parameters relative to the micro-architecture of design instances, while they do not address the problem of e.g. topological mapping of an application on the underlying MPSoC architecture. While micro-architecture parameters like cache size typically affect the system performance in a predictable, often linear, fashion, the resource binding of the application graph to the architectural platform presents a much less predictable performance.

A second class of design space pruning is based on hierarchical DSE (e.g., [7], [14], [10], [5]). In this approach, DSE is first performed using analytical or symbolic models to quickly find the interesting parts in the design space, after which simulation-based DSE is performed to more accurately search for the optimal design points. The main drawback of this method is that if the *first step* is not accurate enough, it may not produce the best set of design points to simulate. In our approach, the pruning and simulation phases are integrated to avoid this problem.

VI. Conclusion

We proposed a technique to reduce the simulation overhead in system-level design space exploration (DSE). To this end, we have presented an iterative design space pruning methodology based on static throughput analysis of different application mappings. By interleaving these analytical throughput estimations with simulations, our hybrid approach can significantly reduce the number of simulations that are needed during the process of DSE. Experimental results have demonstrated that our hybrid DSE is a promising technique, yielding solutions similar in terms of quality as compared to simulation-based DSE but at a fraction of the execution time.

References

[1] G. Ascia, V. Catania, A. G. Di Nuovo, M. Palesi, and D. Patti. Efficient design space exploration for application specific systems-on-a-chip. *J. Syst. Archit.*, 53:733–750, October 2007.

[2] V. Catania and M. Palesi. A multi-objective genetic approach to mapping problem on network-on-chip. *JUCS*, 22:2006.

[3] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast elitist multi-objective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6:182–197, 2000.

[4] M. T. M. Emmerich, K. Giannakoglou, and B. Naujoks. Single- and multiobjective evolutionary optimization assisted by gaussian random field metamodels. *IEEE Transactions on Evolutionary Computation*, 10:421–439, 2006.

[5] 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,vol.10,no.3*, 10:358–374, 2006.

[6] M. Gries. Methods for evaluating and covering the design space during early design development. *Integr. VLSI J.*, 38:131–183, December 2004.

[7] Z. J. Jia, A.D. Pimentel, M. Thompson, T. Bautista, and A. Núñez. Nasa: A generic infrastructure for system-level mp-soc design space exploration, Proceedings of the IEEE Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia) 2010.

[8] P. J. Joseph, K. Vaswani, and M. J. Thazhuthaveetil. Construction and use of linear regression models for processor performance analysis. In *In Proc. 12th IEEE Symposium on High Performance Computer Architecture*, pages 99–108, 2006.

[9] G. Kahn. The semantics of a simple language for parallel programming. In *Proc. of the IFIP Congress 74*, 1974.

[10] J. Kim and M. Orshansky. Towards formal probabilistic power-performance design space exploration. In *Proceedings of the 16th ACM Great Lakes symposium on VLSI*. ACM, 2006.

[11] J. Knowles. Parego: A hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, 10(1), 2006.

[12] G. Mariani, A. Brankovic, G. Palermo, J. Jovic, V. Zaccaria, and C. Silvano. A correlation-based design space exploration methodology for multi-processor systems-on-chip. In *Proceedings of the 47th Design Automation Conference (DAC)*. ACM, 2010.

[13] S. Meijer, H. Nikolov, and T. Stefanov. Throughput modeling to evaluate process merging transformations in polyhedral process networks. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '10, pages 747–752, 3001 Leuven, Belgium, Belgium, 2010. European Design and Automation Association.

[14] S. Mohanty, V. K. Prasanna, S. Neema, and J. Davis. Rapid design space exploration of heterogeneous embedded systems using symbolic search and multi-granular simulation. *SIGPLAN Not.*, 2002.

[15] H. Nikolov. System-level design methodology for streaming multi-processor embedded systems. In *Ph.D. Thesis*, 2009.

[16] H. Nikolov, M. Thompson, T. Stefanov, A. Pimentel, S. Polstra, R. Bose, C. Zissulescu, and E. Deprettere. Daedalus: Toward composable multimedia MPSoC design. In *Proc. of the 45th ACM/IEEE Int. Design Automation Conference (DAC '08)*, 2008.

[17] H. Orsila, E. Salminen, and T. D. Hämäläinen. Parameterizing simulated annealing for distributing kahn process networks on multiprocessor socs. In *Proc. of the Int. Conference on System-on-chip*, pages 19–26, 2009.

[18] G. Palermo, C. Silvano, and V. Zaccaria. Respir: a response surface-based pareto iterative refinement for application-specific design space exploration. *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, 28, 2009.

[19] A. D. Pimentel, C. Erbas, and C. Polstra. A systematic approach to exploring embedded system architectures at multiple abstraction levels. *IEEE Trans. Comput.*, 55(2):99–112, 2006.

[20] A. Sangiovanni-Vincentelli and G. Martin. Platform-based design and software design methodology for embedded systems. *IEEE Des. Test*, 18, 2001.

[21] D. Sheldon, F. Vahid, and S. Lonardi. Soft-core processor customization using the design of experiments paradigm. In *In International Conference on Design and Test in*, 2007.

[22] T. Taghavi and A.D. Pimentel. Design metrics and visualization techniques for analyzing the performance of moeas in dse. In *Proc. of the 11th Int. Conference on Embedded Computer Systems: Architectures, MOdeling and Simulation (SAMOS '11)*, 2011.

[23] J. J. Yi, D. J. Lilja, and D. M. Hawkins. A statistically rigorous approach for improving simulation methodology. In *Proc. of the 9th International Symposium on High-Performance Computer Architecture*, 2003.

[24] E. Zitzler and L. Thiele. Multiobjective optimization using evolutionary algorithms - a comparative case study. In *Proc. of the 5th International Conference on Parallel Problem Solving from Nature*, 1998.