DeSpErate: Speeding-up Design Space Exploration by using Predictive Simulation Scheduling

Giovanni Mariani Gianluca Palermo Vittorio Zaccaria Cristina Silvano
Politecnico di Milano - Dipartimento di Elettronica, Informazione e Bioingegneria

***Abstract*—The design space exploration (DSE) phase is used to tune configurable system parameters and it generally consists of a multiobjective optimization (MOO) problem. It is usually done at pre-design phase and consists of the evaluation of large design spaces where each configuration requires long simulation. Several heuristic techniques have been proposed in the past and the recent trend is reducing the exploration time by using analytic prediction models to approximate the system metrics, effectively pruning sub-optimal configurations from the exploration scope. However, there is still a missing path towards the effective usage of the underlying computing resources used by the DSE process. In this work, we will show that an alternative and almost orthogonal approach — focused on exploiting the available parallelism in terms of computing resources — can be used to better schedule the simulations and to obtain a high speedup with respect to state of the art approaches, without compromising the accuracy of exploration results. Experimental results will be presented by dealing with the DSE problem of a shared memory multi-core system considering a variable number of available parallel resources to support the DSE phase[1].**

I. Introduction

The application-specific platform-based design approach is a widely used technique to deal with the design complexity of today's computing architectures [1]. In this approach, a parameterized platform template is customized to best fit in application specific requirements. The customization process consists of tuning the architectural parameters to optimize the target figures of merit (e.g. performance, power/energy consumption and chip area) using a multi-objective Design Space Exploration (DSE) approach.

In this context, executable simulation models are valuable tools to enable the accurate evaluation of the target optimization objectives for a given platform configuration. To implement the optimization process, nature inspired heuristics might be used, such as genetic algorithms [2] and ant colony optimization [3]. However, these heuristic techniques require the simulation of many candidate platform configurations. Due to the computational time required to carry out each simulation, the DSE process might become unreasonably long. To reduce the simulation time, a well-known solution is to use analytic models to predict the simulation results [4]. Once the analytic models are trained, they can be used to prune the design space by focusing the exploration on the most promising design space regions [4].

In this paper, we will demonstrate that state of the art analytic performance prediction techniques such as [4]–[7] do not represent the best DSE approaches when a parallel computing system (such as a multi-core processor or a computer cluster) can be exploited to run concurrently different simulations. When relying on analytic models to prune the design space we shrink the number of simulations to be run in parallel and we might reduce the benefits of a parallel simulation environment. To tackle this problem, we suggest for the first time to change the perspective and to predict the execution time required to run a simulation rather than to predict the simulation output itself. When simulating a computing system the simulation time might vary significantly, especially when considering a variable number of cores in a multi-core system since the need to model shared resource contention within the simulation model [8]. Variations in the simulation time might generate significant underutilization of the available computational resources when an iterative DSE approach [2], [3], [5] is stacked waiting for the termination of a lengthy simulation. Our intuition is that a better utilization of the parallel computing resources can be obtained by scheduling additional simulations when some computing resources are predicted to become idle, thus gathering more knowledge about the design space that lead to speedup the DSE process. The proposed simulation scheduling is based on analytic simulation time prediction modeling. When considering a 16-cores or a 32-cores simulation environment, the proposed technique has a speedup of about $2\times$ in reference to state of the art optimization algorithms [9]–[11].

The reminder of the paper is organized as follows. Section II reports representative work in the field of DSE for multi-core systems, while Section III is used to explain the work motivations more in detail. Sections IV and V respectively present the proposed methodology and its empirical evaluation. Finally in Section VI we draw our conclusions.

II. Background

In recent years, the multi-objective DSE problem for multi-core homogeneous [12] and heterogeneous [3] systems generated a lot of interest. Different heuristic algorithms have been proposed in literature to tackle this problem [3], [7], [11].

The main issue related with the DSE problem concerns the long simulation time required to evaluate a single system configuration. This problem can be efficiently addressed by exploiting analytic system models. These models are approximations of system performance learned to fit the output of some simulations [5], [13], e.g. by using the *Kriging* interpolation [5] or genetically programmed neural networks [13]. Another option to speedup the DSE process is to exploit a parallel computing environment to concurrently evaluate different configurations [14]. This approach has been exploited in the field of iterative compilation [15] and dynamic memory management [16].

So far, in the field of optimization of computing systems there has not yet been any comparison between the advantages of using a parallel computing environment and an analytic

[1]This work was supported in part by the EC under the grants HARPA FP7-612069 and CONTREX FP7-611146.

978-3-9815370-2-4/DATE14/©2014 EDAA

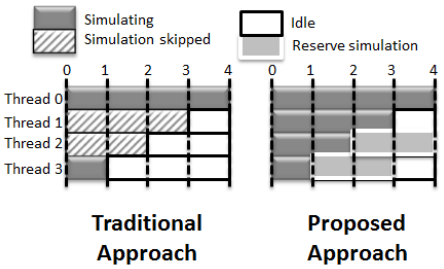


Fig. 1: Comparison of traditional prediction techniques aimed to prune the design space and the proposed one to minimize the idle times of simulation threads.

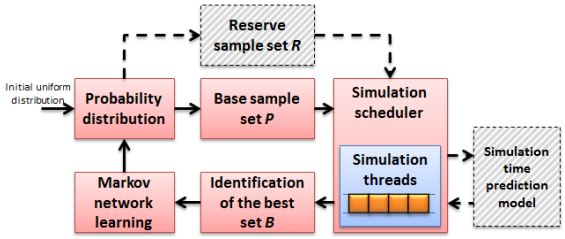


Fig. 2: The base MOA algorithm (continuous lines and boxes) and the additional modules to implement the proposed DESPERATE approach (dashed lines and boxes).

performance prediction model, nor their joint exploitation. In this paper we are going to present a comparison among the two approaches and demonstrate that the introduction of performance prediction model might lead to a suboptimal utilization of the underlying parallel computing environment. Then, we propose a novel technique to exploit prediction techniques in parallel computing environment for the DSE of multi-core architectures.

III. Motivations

So far, several works proposed to speedup the DSE process by pruning some simulations that have been predicted to be suboptimal by means of an approximate system model. We believe that this state-of-the-art approach is very good when considering a sequential simulation environment, but it might be suboptimal when considering a parallel computing system (where different simulations can run in parallel).

Let us consider for example that we need to simulate 4 architectural configurations whose simulations take 1, 2, 3 and 4 time units respectively. Let us consider that we decide to skip the 2^{nd} and 3^{rd} simulation since the approximate system model suggests us that these represent suboptimal solutions. Thus we simulate only the 1^{st} and the 4^{th} configuration. Considering a sequential simulation environment those simulations take 5 time units and the approximate model allows us to save 50% of the simulation time (simulating all configurations takes 10 time units). However if we would have had a parallel computing system where 4 simulation threads can run in parallel, we would have obtain no speedup by skipping the 2^{nd} and 3^{rd} simulation because we have to wait for the termination of the longest simulation. Thus, the simulation throughput is not improved since two simulation threads will execute no simulations (*traditional approach* in Figure 1).

In this paper, we propose a different methodology to better exploit parallel simulation environments. Our basic idea consists of scheduling more simulations (rather than to prune them). Additional simulations can be efficiently scheduled when some computing resources would be otherwise idle. To this end, we suggest to use approximate models to predict the time required to execute the simulations (rather than predicting the quality of the architectural configurations). This will allow us to improve the simulation scheduling to better exploit the available computing resources. In the example (*proposed approach* in Figure 1) we consider that 2 additional simulations can be scheduled (reserve simulations). These simulations taking 2 time units each will not affect negatively the DSE time but will improve the simulation throughput (number of simulations per time unit) thus providing additional information and potentially lead to the identification of better architecture configurations.

IV. The Proposed Methodology

The base optimization algorithm on which we develop the DESPERATE methodology is the *Markovianity-based Optimization Algorithm* (MOA) [9] that has been selected for its efficiency. In the experimental Section we will demonstrate that the chosen algorithm is faster than traditional genetic algorithms. MOA is an iterative optimization process as shown by continuous lines and boxes in Figure 2. At every iteration, a set P of n candidate configurations is sampled from the design space with a given probability distribution. This configuration set is simulated. We assume that all simulations are independent, thus they can be executed in parallel by using different simulation threads on a parallel computing infrastructure. Once all simulations are completed, simulation results are processed. First the set B representing the m best configurations is identified. Then, a *Markov network* is learned to model the probability distribution that best fits the distribution of the set B [9]. The next iteration is initialized by sampling this new probability distribution. The MOA algorithm starts by setting an uniform distribution as initial probability distribution and it ends by returning the Pareto optimal configurations found after running a given number of iterations. MOA belongs to the class of optimization algorithms known as Estimation of Distribution Algorithms (EDAs) since it iteratively estimates the probability distribution of the optimal solutions.

The DESPERATE approach extends the MOA algorithm to better exploit a parallel simulation environment when considering an optimization problem where evaluation time might vary significantly (as in the case of simulation of multi-core computing systems). The additional DESPERATE modules to extend the MOA algorithm are reported in Figure 2 using dashed lines and boxes. At every iteration, we update a simulation time prediction model $\hat{t}$. The first iteration proceeds as for the MOA algorithm by sampling n configurations uniformly distributed in the design space. The simulation time $t(x)$ of each configuration x is collected and used to fit the analytic prediction model $\hat{t}$. This analytic function is an approximation of the simulation time $\hat{t}(x) \sim t(x)$.

In the following iterations, the probability distribution fitting the best m configurations is sampled twice to generate two sets, i.e. P and R. The set P represents the set of n configurations to be simulated as for the MOA algorithm. The set R represents a reserve list of candidate configurations to be simulated if computational power is available.

Simulation time prediction model. To implement the analytic simulation time prediction model we have been inspired by fitness prediction models used in other EDAs. In particular, we consider the analytic model proposed in [17] for the Univariate Marginal Distribution Algorithm (UMDA). Let's identify with X_i the value of the i^{th} parameter for a configuration x. Let's define $\bar{t}$ as the mean simulation time computed over all the configurations executed so far and $\bar{t}(X_i)$ as the mean simulation time computed only over those configurations whose i^{th} parameter has the value X_i. The analytic simulation time prediction is computed as follows:

$$\hat{t}(x) = \bar{t} + \sum_i (\bar{t}(X_i) - \bar{t}) \quad (1)$$

Simulation Scheduler. The *simulation scheduler* in DESPERATE first schedules all configurations in P. Then, once launched the simulations for all $x \in P$, it takes decisions whether or not to launch simulations for the configurations $x \in R$. The *scheduler* keeps track of each configuration under execution. Let us indicate as S the set of configurations currently under simulation and $s(x)$ the time elapsed from the instant we launched the simulation of x. Thus, the expected remaining simulation time $\hat{r}(x)$ for the configuration $x \in S$ is approximated by using the analytic model $\hat{t}(x)$:

$$\hat{r}(x) = \begin{cases} \hat{t}(x) - s(x), & \text{if } \hat{t}(x) > s(x) \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

Given the set of executing simulations we can compute the overall expected idle time $w(S)$ as:

$$\mathring{r}(S) = \max_{x \in S}(\hat{r}(x)) \quad (3)$$

$$w(S) = \sum_{x \in S}(\mathring{r}(S) - \hat{r}(x)) \quad (4)$$

Where $\mathring{r}(S)$ is the time needed for terminating the longest simulation while $\mathring{r}(S) - \hat{r}(x)$ are the idle times, i.e. the time each computational thread is expected to be idle.

Once all configurations $x \in P$ have been launched, the *simulation scheduler* waits for the termination of the next simulation. At that point in time, the scheduler computes $w(S)$ by considering the new idle thread (for the simulation just terminated). Then it parses the configurations $x \in R$, and computes the idle time $w(S')$ considering that we would launch the simulation of the configuration x (i.e. $S' = S \cup x$). If launching the new simulation reduces the overall idle time (i.e. $w(S') < w(S)$), then the simulation is launched, otherwise the configuration x is discarded from the reserve set.

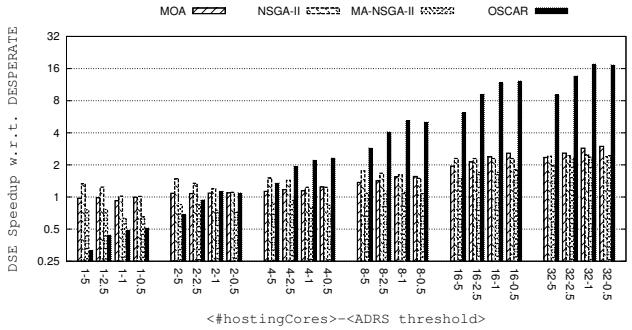
When no configuration $x \in R$ exists to reduce the idle time, DESPERATE waits for the termination of the simulations currently running. Then it continues as the traditional MOA algorithm by identifying the set B and estimating its distributions by means of a *Markov network* (Figure 2). Besides, actual simulation times are saved in the simulation time prediction model and will be used to improve the prediction accuracy.

V. Experimental Results

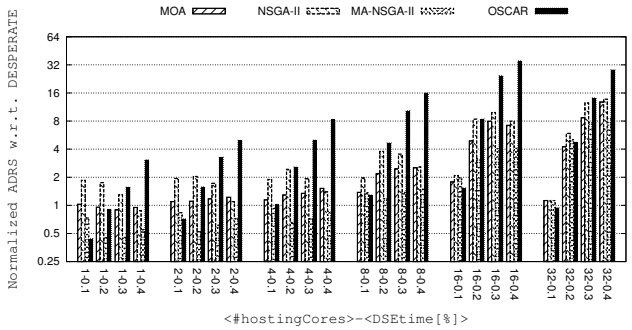
To validate the proposed methodology, we targeted the customization of a symmetric on-Chip Multi-Processor (CMP) architecture modeled by using the SESC [18] simulation infrastructure. The overall experimental setup, including the target design space and the target applications, is described in our previous works [5], [12]. We compare the DESPERATE approach in reference to two algorithms that do not use any analytic prediction model to speedup the optimization process: the Markovianity Optimization Algorithm (MOA) [9] and a multi-objective genetic algorithm named NSGA-II [10]. Additionally, we compare to state-of-the-art approaches that use analytic prediction model to approximate the simulation outputs and thus to speedup the DSE process by reducing the number of simulations. In particular we use a Metamodel-Assisted NSGA-II (MA-NSGA-II) where the approximation is based on an artificial neural network [11] and a last technique that iteratively search on a Kriging model the best architecture configuration to be simulated in the next iteration (named OSCAR) [5]. We used a population size of 64 elements for all the methodologies (except for OSCAR that is not a population-based algorithm). The exploration process is done by launching the simulations on a homogeneous infrastructure including several Intel Xeon processors running at 3GHz. The comparison has been carried out in terms of average (i) exploration speed-up and (ii) ADRS, *Average Distance from Reference Set* [4], improvement. The average values have been extracted over the SPLASH-2 applications used in [5].

(i) Exploration time. Figure 3a shows the average DSE speed-up time of the proposed DESPERATE methodology with respect to each one of the other exploration methodologies when considering the time needed to reach an ADRS of 5%, 2.5%, 1% and 0.5% for different host environment setup (from 1 to 32 cores to run simulations). The general trend shown is that the speed-up of the DESPERATE methodology increases by increasing the number of parallel resources used to host the DSE phase due to the better exploitation of the hosting computing facilities. In more detail, we can notice two different areas defined by the first half and second half of bars-clusters (respectively from 1 to 4 and form 8 to 32 hosting cores). In fact, while in the first set of results the meta-model assisted techniques (i.e. MA-NSGA and OSCAR) present the best results reducing the exploration time, this does not happen in the second set where DESPERATE presents the best results (over 1.5x) with respect to all the other methodologies. The reason for this is mainly due to the better usage of the computing resources exploited by DESPERATE that is able to even overcome the acceleration introduced by these metamodel-assisted methods. OSCAR does not scale over the parallel environments, due to the non-population based nature of the algorithm that does not use the parallel computing facilities differently from the others techniques.

(ii) Quality of the final solution. Figure 3b presents another perspective in the evaluation space by showing the average ADRS improvement of the proposed DESPERATE methodology with respect to the other exploration methodologies after an exploration time equal to 0.1%, 0.2%, 0.3% and 0.4% of the time needed to perform a full search on the different host environment setup, ranging from 1 to 32 cores. The global trend presented in Figure 3b is that increasing the number of hosting cores the proposed methodology demonstrates to overcome the other solutions due to a better usage of the computing resources. Again, two different sets can be analyzed depending on the number of the available computing resources, from 1 to 4 hosting cores and from 8 to 32. While in the second set the DESPERATE approach presents better ADRS than the other approaches, for a small number of computing resources (1 to 4 hosting cores) the behavior for NSGA-II and MOA is almost



(a) DSE speedup of the DESPERATE methodology w.r.t. state of the art approaches when considering different number of parallel simulation threads and different target ADRS thresholds.



(b) ADRS improvement when considering different number of parallel simulation threads and different exploration time budgets relatively to an exhaustive optimization process.

Fig. 3: Improvement of the DESPERATE methodology with respect to the reference methodologies.

constant by showing respectively more or less the same or two times the ADRS of DESPERATE. Regarding MA-NSGA and OSCAR, while the former is better than the proposed methodology up to 4 hosting cores, the latter is able to improve the ADRS found by DESPERATE only for an exploration time equal to 0.1% of the full search. The reason can be found in the fact that OSCAR is a very fast algorithm to reach a good solution but it suffers of falling in local minima (even if they are very close to the optimal solution).

(iii) Utilization of the host computing resources. To complete the comparative results, we want to give an outlook on the different usage of the underlining computing resources. While DESPERATE utilization is very close to 100% of the computing resource, for the other methods the utilization changes depending on the number of hosting cores. As stated before, OSCAR does not scale due to its sequential nature and uses only $1/n$ of the computing resources, where n is the number of computing cores. MOA, NSGA-II and MA-NSGA-II are very similar from the utilization point of view, since they are all population-based methods, showing an average reduction of usage of around $2.5\% \times n$ for the computing platform configurations we adopted during the experiments.

VI. Conclusion

In this paper, we proposed a simulation scheduling technique to exploit a parallel simulation environment during the optimization process. The proposed technique is based on the definition of an analytic model to predict the time required to execute the different simulations. This methodology has been implemented on top of the Markovianity based Optimization Algorithm (MOA). The underlying idea of the approach is that, additional simulations taken from a reserve list can be scheduled when some computing resources are predicted to become idle. Experimental results compare the proposed approach to state of the art DSE solutions. Overall, considering the selected experimental setup, the proposed approach reports a speedup up to $2\times$ when considering DSE systems capable to run 32 or 16 simulation threads in parallel.

References

[1] K. Keutzer, A. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli, "System-level design: orthogonalization of concerns and platform-based design," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 19, pp. 1523–1543, Dec 2000.

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

[3] F. Ferrandi, P. L. Lanzi, C. Pilato, D. Sciuto, and A. Tumeo, "Ant colony heuristic for mapping and scheduling tasks and communications on heterogeneous embedded systems," vol. 29, no. 6, pp. 911–924, 2010.

[4] G. Palermo, C. Silvano, and V. Zaccaria, "ReSPIR: A Response Surface-based Pareto Iterative Refinement for application-specific design space exploration," *IEEE Transactions on Computer Aided Design of Integrated Circuits*, vol. 28, pp. 1816–1829, Dec. 2009.

[5] G. Mariani, G. Palermo, V. Zaccaria, and C. Silvano, "OSCAR: An optimization methodology exploiting spatial correlation in multicore design spaces," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 31, no. 5, pp. 740–753, 2012.

[6] B. Li, L. Peng, and B. Ramadass, "Accurate and efficient processor performance prediction via regression tree based modeling," *Journal of Systems Architecture*, vol. 55, no. 10-12, pp. 457 – 467, 2009.

[7] G. Ascia, V. Catania, A. G. D. Nuovo, M. Palesi, and D. Patti, "Efficient design space exploration for application specific systems-on-a-chip," *Journal of Systems Architecture*, vol. 53, no. 10, pp. 733–750, 2007.

[8] B. C. Lee, J. Collins, H. Wang, and D. Brooks, "Cpr: Composable performance regression for scalable multiprocessor models," in *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 41, (Washington, DC, USA), pp. 270–281, IEEE Computer Society, 2008.

[9] S. Shakya and R. Santana, "An EDA based on local markov property and gibbs sampling," in *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, GECCO '08, (New York, NY, USA), pp. 475–476, ACM, 2008.

[10] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," vol. 6, no. 2, pp. 182–197, 2002.

[11] G. Mariani, G. Palermo, C. Silvano, and V. Zaccaria, "A design space exploration methodology supporting run-time resource management for multi-processor systems-on-chip," in *Proc. IEEE 7th Symp. Application Specific Processors SASP '09*, pp. 21–28, 2009.

[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 *Proc. 47th ACM/IEEE Design Automation Conf. (DAC)*, pp. 120–125, 2010.

[13] H. Cook and K. Skadron, "Predictive design space exploration using genetically programmed response surfaces," in *DAC '08: Proceedings of the 45th annual Design Automation Conference*, (New York, NY, USA), pp. 960–965, ACM, 2008.

[14] W. Zhi-xin and J. Gang, "A parallel genetic algorithm in multi-objective optimization," in *Control and Decision Conference, 2009. CCDC '09. Chinese*, pp. 3497 –3501, june 2009.

[15] H. Jordan, P. Thoman, J. J. Durillo, S. Pellegrini, P. Gschwandtner, T. Fahringer, and H. Moritsch, "A multi-objective auto-tuning framework for parallel codes," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, (Los Alamitos, CA, USA), pp. 10:1–10:12, IEEE Computer Society Press, 2012.

[16] J. L. Risco-Martín, D. Atienza, J. Manuel Colmenar, and O. Garnica, "A parallel evolutionary algorithm to optimize dynamic memory managers in embedded systems," *Parallel Comput.*, vol. 36, pp. 572–590, Oct. 2010.

[17] M. Pelikan and K. Sastry, "Fitness inheritance in the bayesian optimization algorithm.," in *GECCO (2)*, vol. 3103 of *Lecture Notes in Computer Science*, pp. 48–59, Springer, 2004.

[18] J. Renau, B. Fraguela, J. Tuck, W. Liu, M. Prvulovic, L. Ceze, S. Sarangi, P. Sack, K. Strauss, and P. Montesinos, "SESC simulator," January 2005. http://sesc.sourceforge.net.