DeSpErate++: An Enhanced Design Space Exploration Framework Using Predictive Simulation Scheduling

Giovanni Mariani, Gianluca Palermo, *Member, IEEE*, Vittorio Zaccaria, *Member, IEEE*, and Cristina Silvano, *Senior Member, IEEE*

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 meet the application-specific requirements. The customization process consists of a multiobjective design space exploration (DSE) approach to tune the architectural parameters to optimize the target figures of merit (e.g., performance, power/energy consumption, chip area, etc.).

To implement the optimization process, traditional nature-inspired heuristics such as genetic algorithms [2], simulated annealing [3], and ant colony optimization [4] are widely used. In this context, executable simulation models are valuable tools to enable the accurate evaluation of the target figures of merit for a given platform configuration. 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 can become unreasonably long. To reduce the simulation time, a solution is to use analytic models to predict the simulation results [5]. Once analytic models are trained to fit the behavior of some observed simulations, they can be used to prune the design space by focusing the exploration on the most promising design points [6].

An orthogonal path to solve the problem is to embrace the so-called parallel computer aided design (CAD) philosophy [8], [9] that imposes to design tools to scale over a large set of computing nodes. As for standard parallel programming, the main rule in parallel CAD is to keep busy all the available processors (computing units) doing useful work for the entire period to speedup the design process. However, the full exploitation of parallel resources to support the design process is still a big challenge in the design automation field [10] lacking of practical solutions covering all the areas.

In this paper, we demonstrate that state-of-the-art analytic performance prediction techniques such as [5], [6], and [11]–[13] do not represent the best DSE approach when a parallel computing system (e.g., a multicore processor or a computer cluster) is exploited to run concurrently different simulations. To clarify this idea, Fig. 1 shows the distribution of simulation times when considering a chip multiprocessor modeled using the super escalar simulator (SESC) simulator [7]. Similar considerations apply for many other computer architecture simulators such as Sniper [14], SlackSim [15], and MARSS [16]. Results in Fig. 1 are reported for four applications of the SPLASH-2 benchmark suite [17]. For each application, simulation times are collected for different architectural configurations without modifying the target dataset. For two out of four applications

Manuscript received May 14, 2014; revised July 25, 2014 and October 3, 2014; accepted November 15, 2014. Date of publication December 12, 2014; date of current version February 16, 2015. This work was supported by the European Commission under Grant FP7-612069-HARPA and Grant FP7-611146-CONTREX. This paper was recommended by Associate Editor T. Kim.

The authors are with the Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria, Milano 20133, Italy (e-mail: giovannisiro.mariani@polimi.it; gianluca.palermo@polimi.it; vittorio.zaccaria@polimi.it; cristina.silvano@polimi.it).

Color versions of one or more of the figures in this paper are available online.

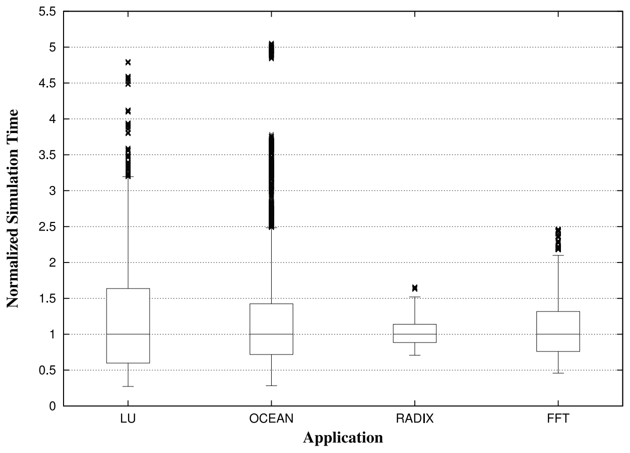


Fig. 1. Box-plot of the simulation time distribution for the SESC [7] simulator for different design configurations with respect to the target application. Simulation times are normalized by their median value.

(namely the blocked lower-upper matrix decomposition (LU) and the ocean simulation (OCEAN) applications) the simulation time variations consist of one order of magnitude (from 0.5 to 5). The integer radix sort (RADIX) application is characterized by the lowest simulation time variability with a $2\times$ difference between the slowest and the fastest simulations (about 0.75–1.5 normalized time, respectively). This means that, when using a computer cluster (or a multicore system) to run simulations in parallel, simulation scheduling becomes a relevant problem. One computing node might run different short simulations while another computing node runs a single long simulation. This consideration suggests us that, without an accurate scheduling of the parallel simulations, the simulation time variability might lead to a significant underutilization of the parallel computational resources. This intuition drove [18], where machine learning techniques were used to predict the simulation times associated with different architectural configurations and thus to schedule the simulations.

In this paper, we extend [18] by introducing DESPERATE++, a simulation scheduling technique for parallel design environments that includes two orthogonal analytic prediction models to characterize candidate simulations by predicting not only the simulation time—as done in [18]—but also the configuration quality. The analytic model of configuration quality is used to focus the exploration on the most promising design regions, while the analytic model of simulation time is used to schedule the simulations. In this paper, we also describe in detail the novel predictive scheduling technique. Finally, we demonstrate that DESPERATE++ overcomes state-of-the-art approaches for parallel computing environments, including [18], and we show some experimental results including model accuracy and related overheads.

The reminder of this paper is organized as follows. Section II reports the related works in the field of DSE for multicore systems, while Section III explains a motivating example. Section IV presents the proposed methodology. Empirical evaluation of the proposed approach in reference to the state-of-the-art is reported in Section V. Finally, the conclusion is provided in Section VI.

II. Related Works

In recent years, the automatic multiobjective DSE problem for multicore architectures generated an increasing interest in the electronic design automation research community [19]–[22].

The first main problem in the field of DSE is that current and next generation of homogeneous and heterogeneous systems are able to expose a large number of parameters that should be tuned to find the most suitable architectural configuration for the target application domain. To tackle this problem, many previous works propose heuristic solutions derived from different areas ranging from genetic algorithms [13], [23], [24] and other nature-inspired optimization algorithms [4], [25], [26], up to simulated annealing [3], [27] and structured design of experiments [28], [29].

The second main problem is related to the fact that DSE is a typical predesign phase, where the architecture has not been already tapeout or prototyped and thus the evaluation of system configurations are based on long simulations. To alleviate this problem, several techniques have been adopted to speed-up the simulation time considering both sampling techniques [30]–[32] and parallel simulators [33]–[35]. In the context of the DSE phase, researchers proposed to better tackle the problem with the usage of approximate system models. These models are analytic approximations of system performance that are learned after an initial training phase [36] and used to determine where to focus the costly evaluations [5], [6], [37]. Widely used methods are linear regressions [38], radial basis functions [5], [39], neural networks [40], regression trees [28], [41], and statistical techniques such as Gaussian processes [42] and Kriging interpolation [11], [43].

An orthogonal option to speedup the DSE process is to exploit a parallel computing environment to concurrently evaluate different system configurations [44]. This approach, that is the core of the proposed solution, cannot be based simply on a job scheduler for parallel and distributed environments [45], since it is not enough to support a structured DSE with optimization purposes. Combined with parallel optimization algorithms, the usage of a parallel computing environment has been successfully exploited in the field of iterative compilation [46], dynamic memory management [47], and microprocessor optimization [48]. On top of these works, the proposed method combines a parallel DSE algorithm with quality and time prediction models to support the scheduling of simulations on the parallel resources.

In [18], we combined the advantages of a parallel computing environment and analytic performance prediction models. We proposed DESPERATE, a prediction-based simulation scheduling technique for parallel design environments that adopts an analytic prediction technique for estimating the simulation run-time with the goal of balancing the workload on the computational nodes. In this paper, we extend [18] by proposing the DESPERATE++ DSE algorithm that adds a configuration quality predictor in an orthogonal way to the previous simulation time prediction model. Given some candidate design configurations, the combination of the two models predict: 1) the quality of these configurations, to focus the exploration

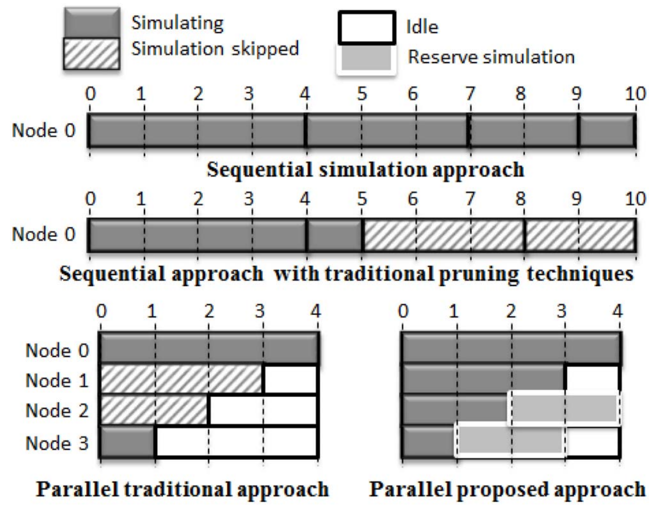

Simulating
Simulation skipped
Idle
Reserve simulation
Node 0
Sequential simulation approach
Sequential approach with traditional pruning techniques
Node 1
Node 2
Node 3
Parallel traditional approach
Parallel proposed approach

Fig. 2. Comparison of traditional prediction techniques based on pruning the design space and the proposed one minimizing idle times on simulation nodes.

effort on the most promising design regions and 2) their simulation times, to fully exploit the parallel environment. Finally, in this paper we demonstrate that, thanks to this new feature, the novel DESPERATE++ approach overcomes the exploration performance of its predecessor by providing a 1.3× speedup.

III. Motivating Example

Traditional heuristic optimization algorithms are very time consuming when considering the problem of tuning computer architecture parameters. Their lack of efficiency is mainly due to the high cost of computational intensive simulations.

So far, several works proposed to speedup the DSE process by pruning the number of simulations to be executed by means of an approximate system model [6], [11], [28], [37], [41]. This approach is promising when considering a single simulation node in the environment hosting the DSE process. With the term simulation node, we refer to each computational resource needed to run a single simulation, being this simulation either single-thread or multithread [49].

Let us clarify with a motivating example why the pruning approach becomes less appropriate when the goal is to speedup the exploration process given a host environment where different nodes are available to execute simulations in parallel [44], [46]–[48]. In this case, simply pruning simulations might result in an underutilization of the overall computing infrastructure leading no benefits in terms of the overall exploration time.

Let us consider that we need to evaluate four architectural configurations whose simulations take 1–4 time units, respectively. These four configurations might represent the individuals of a population based heuristic DSE algorithm (e.g., a genetic algorithm with a population size $\gamma = 4$). Simulating these four configurations by using a sequential approach would take ten time units (Fig. 2). Let us consider that we decide to prune the second and the third simulations since an approximate system model suggests us that these configurations are suboptimal. A sequential approach using traditional pruning techniques (Fig. 2) simulates the first and the fourth configuration taking five time units. Thus, in this specific case the approximate model allows to save 50% of the simulation time.

When considering a parallel computing environment, it is possible to achieve a significant speedup by running different simulations concurrently. In this example, when considering that four simulation nodes are available to run one simulation per node, the evaluation lasts four time units since this is the duration of the longest simulation. In this situation, we obtain no further speedup by skipping the second and the third simulation because, we still have to wait for the termination of the longest simulation (parallel traditional approach in Fig. 2).

In a parallel simulation environment, the simulation time variability typical of architectural simulators (Fig. 1) leads to an underutilization of the computational resources hosting the simulations. In this paper, we propose a methodology to exploit this simulation time variability. Rather than pruning the number of simulations to be executed, our idea consists of identifying an efficient simulation scheduling based on: 1) the available computing resources and 2) an analytic model predicting the time required to execute the simulation of each design configuration. In the example (parallel proposed approach in Fig. 2), we consider that simulation pruning is not necessary and two more simulations can be scheduled during the idle periods (reserve simulations). These reserve simulations (taking two time units each in the proposed example) will not affect negatively the overall DSE time but they will increase the number of simulations run, thus providing additional information and potentially lead to the identification of better architecture configurations. The proposed DESPERATE++ approach combines the simulation time prediction model with an analytic model on the quality of architectural configurations. While the simulation time prediction model is used to decide whether or not to run the reserve simulations, the configuration quality prediction model supports the selection of the configurations to be considered as reserve simulations to focus the exploration on the most promising design region.

IV. Proposed Methodology

In [11] and [43], we demonstrated that the quality of design configurations are correlated in the design space (intuitively similar configurations have similar quality). This finding implies that optimal design configurations are not uniformly distributed in the design space but rather their distribution follows a problem-specific density function. Given that, we believe that estimation of distribution algorithms (EDAs) represent a good matching for the architectural DSE problem [26]. EDAs are optimization algorithms explicitly designed to learn the probability distribution of good solutions to focus the DSE process toward design regions densely populated by optimal design configurations. EDAs are particularly suitable for implementing the proposed DSE approach because the design configurations to be simulated are independently sampled with a given probability distribution and can be launched in parallel. This mechanism is very flexible because, when a computing node is expected to become

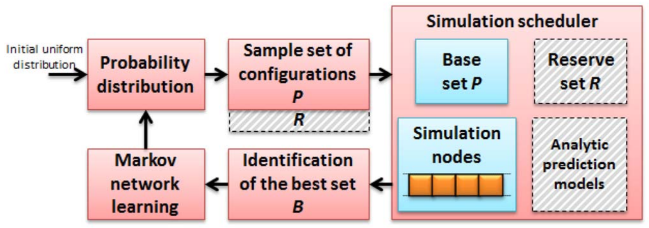

Initial uniform distribution
Probability distribution
Sample set of configurations P
R
Simulation scheduler
Base set P
Reserve set R
Simulation nodes
Analytic prediction models
Markov network learning
Identification of the best set B

Fig. 3. Base MOA algorithm and (represented as dashed boxes) the additional modules introduced by the proposed approach.

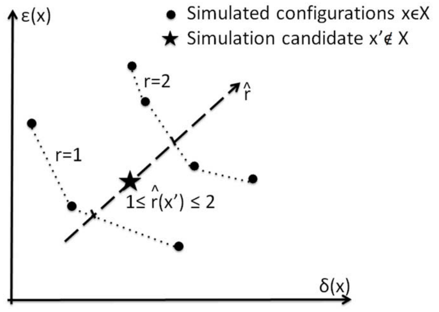


Fig. 4. Actual and estimated rank for simulated configurations $x \in X$ and nonsimulated configurations $x' \notin X$, given the set X of configurations simulated so far.

idle, we can sample a new reserve simulation from the given probability distribution and launch the simulation. The base EDA over which we develop the proposed methodology is the Markovianity-based optimization algorithm (MOA) [50] that has been selected for its efficiency.

MOA is an iterative optimization process as presented in Fig. 3. At each iteration, a set of γ configurations is sampled from the design space with a given probability distribution. These simulations represent the base set P and are all scheduled for execution on the available computing resources. Once all simulations are completed, simulation results are processed as follows. 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 [50]. In this model, the probability distribution of good design configurations is represented by an undirected graph whose nodes represent design variables and their probability distributions, while edges represent dependencies between different variables. 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 so far after running a given number of iterations. Off-the-shelf tools exist to implement the MOA algorithm. In this paper, we use the MATLAB implementation derived from [51].

The proposed approach extends the MOA algorithm to better exploit a parallel simulation environment. The additional modules required to extend the MOA algorithm are reported in Fig. 3 by using dashed boxes. At each iteration of DESPERATE++, the probability distribution model is sampled to generate a set of $(\gamma + \rho)$ design configurations. The scheduler partitions this input set in a base set P of γ elements (as in MOA) and a reserve set R of ρ elements. The set P represents the set of configurations to be simulated as for the MOA algorithm. The set R represents the reserve list of candidate configurations to be simulated if computational resources are available.

In DESPERATE++, a prediction model that estimates the quality of architectural configurations is responsible for partitioning the sample set of configurations—input to the simulation scheduler—into the two sets P and R. Other scheduling decisions are taken based on the simulation time prediction model. At each iteration of the algorithm, the collected simulation results are used to update these analytic prediction models to improve their accuracy. Besides, the probability distribution model based on a Markov network is also updated as in MOA. In the proposed approach, the first iteration proceeds by sampling γ configurations uniformly distributed in the design space that are all simulated. These simulations are used to bootstrap the analytic models.

A. Predicting the Quality of Architectural Configurations

We measure configuration quality in a multiobjective perspective [11] by using the concept of the nondominated rank $r(x)$. The analytic prediction model of configuration quality $\hat{r}(x)$ represents an approximation of the nondominated rank $\hat{r}(x) \sim r(x)$.

Without loss of generality, let us consider a multiobjective minimization problem where each objective $o_i(x)$ has to be minimized. Let us call X the set of configurations simulated so far. The Pareto front $\mathcal{P}(X) \subseteq X$ represents the best configurations among the set X

$$x \prec x' \iff \forall i, o_i(x) \le o_i\left(x'\right) \wedge \exists i, o_i(x) < o_i\left(x'\right) \qquad (1)$$

$$\mathcal{P}(X) = \left\{x \in X \mid \nexists x' \in X, x' \prec x\right\} \qquad (2)$$

where $\prec$ represents the Pareto dominance operator. The nondominated rank $r(x)$ of a configuration $x \in X$ is a measure of how deep x is nested in sub-optimal (nonPareto) solutions. This concept is graphically represented in Fig. 4 for a 2-D minimization problem whose objectives are δ and ε. All the design configurations x belonging to the Pareto front have $r(x) = 1$. Configurations with $r(x) = 2$ are those which are dominated only by design configurations with $r(x) = 1$. The higher $r(x)$ is, the worse the design configuration x is.

More formally, given the set of design configurations simulated so far X, we define a sequence of sets $\Lambda_n \subseteq X$ and Pareto fronts $\mathcal{P}_n \subseteq X$ such that

$$\Lambda_n = X, n = 1 \qquad (3)$$

$$\mathcal{P}_n = \mathcal{P}(\Lambda_n), n \ge 1 \qquad (4)$$

$$\Lambda_n = \Lambda_{n-1} \setminus \mathcal{P}_{n-1}, n > 1. \qquad (5)$$

The nondominated rank of a design configuration $x \in X$ is then defined as

$$r(x) = n \leftrightarrow x \in \mathcal{P}_n. \qquad (6)$$

At each iteration of DESPERATE++, an artificial neural network (ANN) is trained to fit the nondominated rank of the

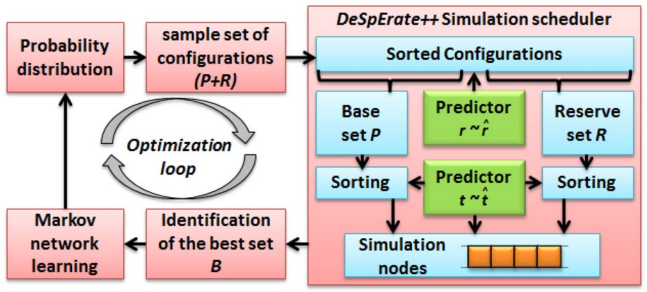


Fig. 5. Proposed DESPERATE++ simulation scheduler.

design configurations simulated so far. We use a fully connected network with a single hidden layer. The number of neurons in the hidden layer is set to half the number of the input neurons [52]. The feedforward backpropagation training scheme is adopted for the learning mechanism. The sigmoid tangent activation function is used for the hidden neurons while the output neuron uses a linear activation function. Nondominated rank of design configurations not yet simulated $x' \notin X$ (i.e., simulation candidates) is predicted in a continuous domain (Fig. 4).

1) Selecting the Base and Reserve Sets: The simulation scheduler of DESPERATE++ is represented in Fig. 5. At each iteration of the optimization loop, $(\gamma + \rho)$ configurations are sampled from the design space given the probability distribution in the current Markov network model. These configurations are sorted by their predicted quality $\hat{r}(x)$ and then partitioned to shape out the base and the reserve sets. In particular, the base set P includes the γ most promising configurations in reference to their predicted quality $\hat{r}(x)$, while the reserve set R includes the remaining ρ configurations.

Then, all configurations $x \in P$ are scheduled for simulations, while configurations $x \in R$ are scheduled only if their execution is expected to reduce the overall idle time of the computing system hosting simulations. Thus, in DESPERATE++, the optimization is driven toward the optimal design configurations thanks to two concepts: 1) the nondominated rank prediction model $\hat{r}(x)$ that selects the most promising configurations in the base population P and 2) the Markov network learning (Fig. 5) that learns the probability distribution to drive the search for the best configuration candidates as in MOA.

B. Simulation Scheduler

The simulation scheduler of DESPERATE++ exploits a simulation time prediction model to carefully schedule the simulations with the goal of better utilizing the available computing resources. The simulation time $t(x)$ of a configuration x is approximated with an analytic predictor function $\hat{t}(x) \sim t(x)$. To implement this analytic model, we have been inspired by fitness prediction techniques used in other EDAs. In particular, we consider the analytic model proposed in [53] for the univariate marginal distribution algorithm. Let us identify with X_i the value of the ith parameter for a configuration x. Let us 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 ith parameter has the value X_i. The simulation time prediction is computed analytically as follows:

$$\hat{t}(x) = \bar{t} + \sum_i \left(\bar{t}(X_i) - \bar{t}\right). \tag{7}$$

The simulation scheduler in DESPERATE++ (Fig. 5) sorts the configurations in each set P and R with respect to the expected simulation time $\hat{t}(x)$ in descending order. Then, all configurations $x \in P$ are scheduled starting from the one expected to take the longest time. New simulations are launched as soon as a simulation node terminates the simulation currently running.

The scheduler keeps track of each simulation under execution. Let us call S the set of configurations currently under simulation and $s(x)$ the time elapsed from the instant we launched the simulation of $x \in S$. Thus, the expected remaining simulation time $\hat{\tau}(x)$ for the configuration $x \in S$ is approximated by using the analytic model $\hat{t}(x)$

$$\hat{\tau}(x) = \begin{cases} \hat{t}(x) - s(x), & \text{if } \hat{t}(x) > s(x) \\ 0, & \text{otherwise.} \end{cases} \tag{8}$$

Let us refer to the number of nodes available to host simulations as θ. Once the last configuration $x \in P$ has been launched, the set S has a size equal to θ. Then, as soon as simulations are terminated, some nodes might become idle. However, we will consider a set S always of size θ where idle nodes have an expected remaining simulation time $\hat{\tau} = 0$. Given the set S of currently running simulations, we can compute the overall expected idle time $w(S)$ as

$$\mathring{\tau}(S) = \max_{x \in S} \left(\hat{\tau}(x)\right) \tag{9}$$

$$w(S) = \sum_{x \in S} \left(\mathring{\tau}(S) - \hat{\tau}(x)\right) \tag{10}$$

where $\mathring{\tau}(S)$ is the time needed for terminating the longest simulation while $\mathring{\tau}(S) - \hat{\tau}(x)$ are the idle times, i.e., the time each computational node 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, the scheduler computes $w(S)$ by considering the new idle node (for the simulation just terminated). Then, it parses the configurations $x' \in R$, and computes the idle time $w(S')$ considering that we are launching 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. Thus, the simulation schedule depends on the sorting of sets P and R as well as on the point in time when the running simulations are terminated that, in turn, influences the values of $\hat{\tau}$ and thus of $w(S)$, determined by (8) and (10).

When no one configuration $x \in R$ exists for reducing the idle time, DESPERATE++ waits for the termination of the simulations currently running. Then, it continues as the traditional MOA algorithm, i.e., it identifies the set B representing the m best configurations; it models their distribution with a Markov network and returns this distribution model as input to the

TABLE I
DESIGN SPACE FOR THE TARGET SHARED-MEMORY MULTICORE ARCHITECTURE

Parameter	Min.	Max.
Number of Processors	2	16
Processor Issue Width	1	8
L1 Instruction Cache Size	2K	16K
L1 Data Cache Size	2K	16K
L2 Private Cache Size	32K	256K
L1 Instruction Cache Associativity	1w	8w
L1 Data Cache Associativity	1w	8w
L2 Private Cache Associativity	1w	8w
I/D/L2 Block Size	16	32

next iteration (Fig. 5). Once the elaboration of the configuration sets P and R is terminated, actual simulation results are learned by the simulation time prediction model and by the nondominated rank prediction model to improve the prediction accuracy.

Finally, we would like to highlight how the proposed simulation scheduler also manages the case of elastic design environments, where the number of resource allocated to the DSE phase is variable (depending on cluster-level power-aware policies or when other design tasks are requesting/releasing computing resources). To manage this case, the previous formulation should include the following two considerations: 1) when a resource is no more available, its expected remaining simulation time $\hat{\tau}$ is always equal to $\mathring{\tau}(S)$ and 2) when a resource is back into the available set, its expected remaining simulation time is $\hat{\tau} = 0$, as for standard idle resources.

V. EXPERIMENTAL RESULTS

To validate the proposed methodology, we targeted the customization of a symmetric multicore architecture modeled by using the SESC [7] simulation infrastructure. This infrastructure models out-of-order MIPS-based (R10K architecture) multiprocessors. It estimates the energy and execution time metrics associated to a pair of application-architectural configuration. These two system level metrics are the objectives of the multiobjective optimization problem.

The target design space is composed of parameters related to task level and instruction level parallelism and to the memory configuration. A detailed view of the parameters and their ranges is shown in Table I. In particular, their values can assume only integer levels representing a power of two thus generating a design space of roughly 128 000 (2^{17}) possible configurations.

Regarding the application scenarios, we considered four benchmarks derived from the SPLASH-2 parallel suite [17]: complex 1-D fast Fourier transform (FFT), RADIX, OCEAN, and LU. Additionally, we adopted three different input parameter settings for each application creating 12 different application scenarios. In reference to the SPLASH-2 framework [17], the three input parameter settings for the different benchmarks are as follows.

1) FFT: *-m14*, *-m12*, and *-m10*.
2) RADIX: *-n64k*, *-n32k*, and *-n16k*.
3) OCEAN: *-n34*, *-n18*, and *-n10*.
4) LU: *-n128*, *-n64*, and *-n32*.

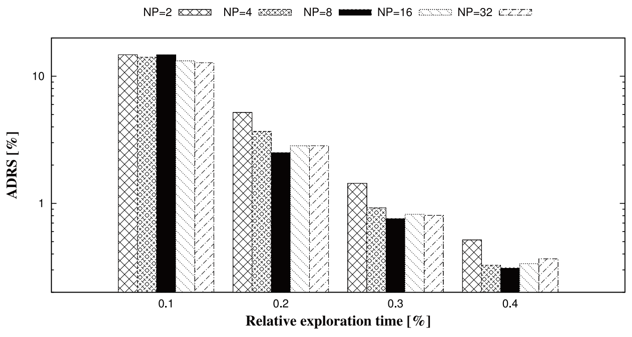


Fig. 6. Tuning of the normalized population size *NP*.

Given this experimental setup, the exhaustive exploration of the whole design space would have required 354 days of simulations by using a single computing node design environment composed of an Intel Xeon processors running at 3 GHz. In this paper, we used a cluster including up to 64 computing nodes to generate the reported experimental results.

A. Tuning of the Normalized Population Size

First, we want to tune the proposed algorithm in terms of ρ, that is the size of the reserve population R. The size of the base population γ is set to 64. The value 64 has been chosen because it represents the maximum number of parallel simulation nodes θ considered in this paper.

Let us define the normalized population size *NP* as

$$NP = \frac{(\gamma + \rho)}{\gamma}. \tag{11}$$

NP represents the populations' size $(\gamma + \rho)$ relatively to the base population size γ. For example, $NP = 2$ means that the overall populations' size sum is twice the base population (i.e., reserve and base sets have the same size). Values larger than two mean that the reserve population is larger than the base population. We investigate the *NP* values in the set $\{2, 4, 8, 16, 32\}$. Results reported for the *NP* parameter tuning consider four computing nodes to host simulations in parallel, $\theta = 4$.

Each setting of *NP* is evaluated in terms of average distance from reference set (ADRS) reached by DESPERATE++ when the DSE is stopped after a certain amount of time. The ADRS metric [54] evaluates the quality of the Pareto front found with a multiobjective optimization algorithm by measuring its distance from the reference Pareto-optimal solutions obtained through an exhaustive exploration. The lower the ADRS, the better is the Pareto-front.

In DESPERATE++, increasing *NP* means to increase the pressure on the analytic model that estimates the quality of the architectural configurations. As *NP* grows, a larger number $(\gamma + \rho)$ of configurations is exposed to the simulation scheduler during the selection of the γ best configurations that compose the base population P. Fig. 6 reports the effect of the parameter *NP* on the accuracy of the Pareto solution identified after a given exploration time.[1] For a very short exploration

[1]The exploration time is measured with respect to the time required for carrying out an exhaustive exploration using the same number of simulation nodes.

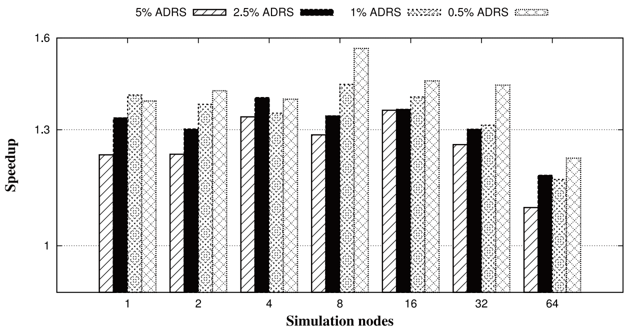


Fig. 7. Speedup of the proposed DESPERATE++ algorithm in reference to the baseline DESPERATE algorithm by varying the number of simulation nodes and the ADRS.

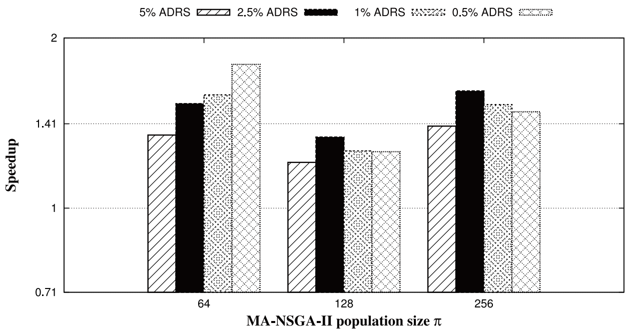


Fig. 8. Speedup of DESPERATE++ in reference to the MA-NSGA-II algorithm considering 64 simulation nodes by varying the population size γ of the MA-NSGA-II and the ADRS.

time (0.1% of the exhaustive exploration time), the value of *NP* is rather irrelevant. The analytical model is inaccurate during the initial exploration phase because too few configurations have been simulated. For longer exploration time, a larger *NP* leads to a better ADRS. The best *NP* value estimated empirically is 8. Further increasing *NP* lets the analytic prediction model become too much relevant in the DSE process. Thus, the effects of model errors emerge. Model errors push the optimization toward local optima slowing down the optimization for *NP* values higher than 8.

B. Comparison with respect to the State-of-the-Art

In this section, we compare DESPERATE++ with respect to several state of the art techniques. First of all, we focus on comparison between the enhanced DESPERATE++ algorithm and its baseline implementation DESPERATE [18] to prove the benefits introduced by the proposed extension.

As highlighted in Section II, DESPERATE differs from DESPERATE++ because there is no nondominated rank prediction model, thus the base and reserve populations are picked up at random from the sample set of configurations input to the simulation scheduler. Note that the simulation time prediction model (7) and the method for selecting whether or not to run a reserve simulation do not change for the two techniques.

Fig. 7 reports the speedup obtained by DESPERATE++ with respect to DESPERATE when both methods are run up to solutions characterized by a given ADRS value. Results for different ADRS values (5%, 2.5%, 1%, and 0.5%) are reported by different bars. The analysis is carried out by varying the number of nodes concurrently hosting the simulations (from 1 to 64 by power of 2). The speedup of DESPERATE++ is generally higher when targeting more accurate solutions characterized by lower ADRS. However, a clear trend in relation to the number of nodes θ cannot be pointed out. Generally, when considering from 1 to 32 simulation nodes, the resulting speedup is around 1.3×. When using 64 simulation nodes the speedup is in the range of 1.1×–1.25×.

We also compare DESPERATE++ in reference to five state-of-the-art optimization algorithms. Two out of them do not use any analytic prediction model to speedup the optimization process: 1) the MOA [50] and 2) the multiobjective genetic algorithm named NSGA-II [55]. Two other algorithms use a moderate mix of accurate simulations and analytic prediction models to avoid unnecessary simulations. These algorithms are: 3) a metamodel-assisted NSGA-II (MA-NSGA-II) where the approximation is based on an artificial neural network [56] and 4) OSCAR [11], a technique based on a Kriging model to iteratively search the best architecture configuration to be simulated in the next iteration. So far, OSCAR has been proven to be the best sequential exploration algorithm for the target design space. The last algorithm: 5) RESPIR [5] extensively uses an analytic prediction model and simulates only those configurations expected to be Pareto optimal on this approximate model. These reference methodologies except the MOA algorithm have been previously applied for the design optimization of computer architectures. Evolutionary algorithms such as the NSGA-II are proposed in [24] and [57], the MA-NSGA-II algorithm has been applied in [23] and [58] while OSCAR and RESPIR have been proposed in [5] and [11].

Given the randomness introduced by the optimization process in the initial sampling for all the methods, algorithm evaluations are carried out by repeating ten times the optimization of each application scenario, each time changing the initial population. We use a population size $\gamma = 64$ for MOA and NSGA-II (the same size of the base population in the proposed approach). For the MA-NSGA-II algorithm, the simulation pruning mechanism reduces the number of simulations to run at each generation. This implies that, by setting $\gamma = 64$ for this algorithm, less than one simulation per node would be launched when having 64 simulation nodes (64 is the maximum number of simulation nodes considered in this paper). Thus, for MA-NSGA-II, we investigate larger population size γ as well, within the set $\{64, 128, 256\}$. Fig. 8 shows the speedup of the proposed DESPERATE++ algorithm in reference to the MA-NSGA-II for different settings of γ, when considering an exploration environment of 64 simulation nodes. The MA-NSGA-II provides the best performance with $\gamma = 128$ that leads to the minimum DESPERATE++ speedup. Thus, the setting $\gamma = 128$ is used in the next analysis for the MA-NSGA-II algorithm.

1) Convergence Frequency Analysis: The use of analytic performance models to approximate the simulation output introduces an estimation error that can drive the exploration toward local optima. This impacts negatively—especially in case of pruning configurations—on the probability of identifying accurate solutions characterized by low ADRS values.

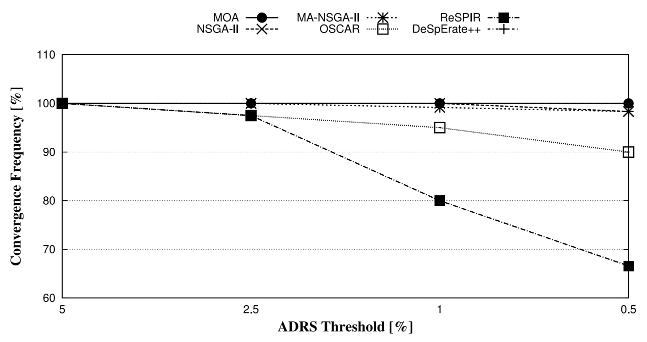


Fig. 9. Probability of converging toward solutions whose ADRS is lower than a given threshold.

Fig. 9 shows the probability of converging toward solutions characterized by a given ADRS for the different methodologies. This probability is estimated over the different algorithm executions. In this paper, MOA, NSGA-II, MA-NSGA-II, and DESPERATE++ are stopped after processing 50 generations. RESPIR is run until it converges and stops issuing the simulation of any new configuration [5]. OSCAR is stopped after 500 iterations (i.e., OSCAR simulates 500 design configurations).

MOA and NSGA-II algorithms are not subject to any model error and their convergence probability is very high, almost 100%. To escape from local optima, MA-NSGA-II simulates some configurations even if these are suboptimal in reference to the approximate model [56]. This mechanism lets the algorithm converge toward accurate solutions with a higher probability.

Also DESPERATE++ has a high probability of converging toward accurate solutions. At each iteration of the proposed algorithm, a set P of γ configurations is selected from a pool of $\gamma + \rho$ configurations. This selection process is robust with respect to approximation errors since it allows to simulate some design configurations that are predicted as suboptimal by the approximate model. In fact, whenever in the pool of $\gamma + \rho$ configurations the optimal designs are fewer than γ, some suboptimal configurations are included in P.

OSCAR exploits a statistical analysis to estimate model uncertainty and it uses this information for escaping from local optima. Its convergence probability is approximately 90% when targeting accurate solutions characterized by 0.5% ADRS.

RESPIR is characterized by a low convergence probability since it prunes most of the design configurations. This algorithm simulates only design configurations that are optimal for the approximate analytic models. When these configurations turn to be suboptimal for the real system this algorithm fails to identify solutions having low ADRS value.

2) Speedup Analysis: We analyze the exploration speedup obtained by DESPERATE++ in reference to the alternative optimization algorithms when targeting the identification of solutions with a given ADRS. The RESPIR methodology is excluded from the analysis since it is likely to miss the identification of solutions with low ADRS values. All remaining methodologies have been parallelized at population-level, i.e., all design configurations to be evaluated in a generation are executed in parallel as long as simulation nodes are available. Regarding OSCAR, it represents an exception because it is not a population based algorithm but it is a sequential DSE approach.

Comparison results in Fig. 10 report the average speedup of DESPERATE++ obtained by varying the number of nodes available to host the simulations (from 1 to 64 parallel nodes). The proposed approach surpasses MOA and NSGA-II algorithms always providing a speedup greater than 1 [Fig. 10(a) and (b)]. These techniques are population-level parallelized but do not make use of any prediction technique.

MA-NSGA-II is an enhanced version of NSGA-II where some simulations are pruned on the basis of the approximate prediction model. Thus, the speedup of DESPERATE++ in reference to MA-NSGA-II [Fig. 10(c)] is significantly lower than for the NSGA-II case [Fig. 10(b)]. When using 64 simulation nodes and considering solutions with 0.5% of ADRS, the speedup in reference to NSGA-II is about 4× while a speedup of 1.26× is measured in reference to MA-NSGA-II algorithm.

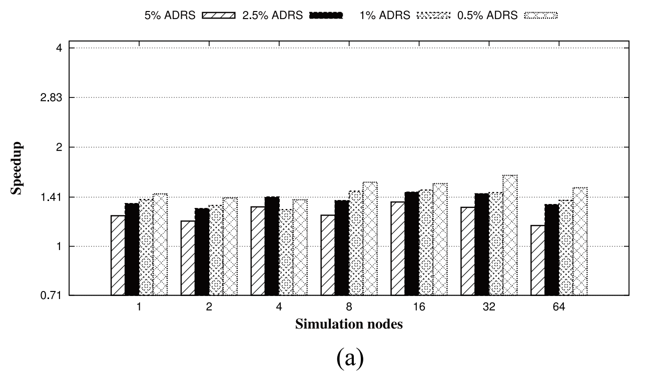
In reference to the sequential prediction-based technique named OSCAR [Fig. 10(d)], the speedup grows significantly when increasing the number of simulation nodes. Note that the comparison with OSCAR [Fig. 10(d)] has been reported with a different scale since the speedup range (y-axis) is significantly larger than for the other three methods. The speedup in reference to OSCAR ranges from 0.25×–0.5× when using a single simulation node (sequential optimization) to 8×–16× when using 64 simulation nodes.

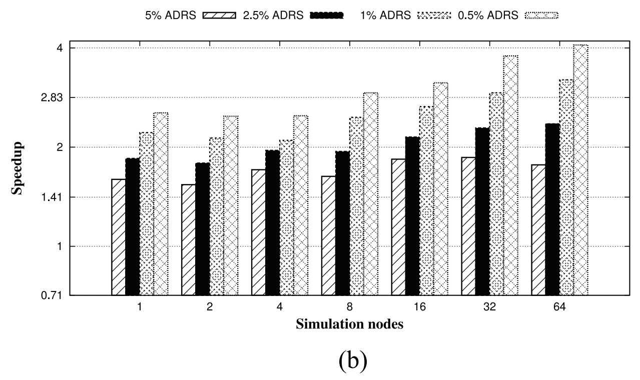
To summarize, when considering 64 simulation nodes and targeting a solution with 0.5% ADRS, DESPERATE++ provides a speedup from 1.26× up to 4× in reference to population-level parallelized DSE algorithms and a speedup of 16× in reference to OSCAR.

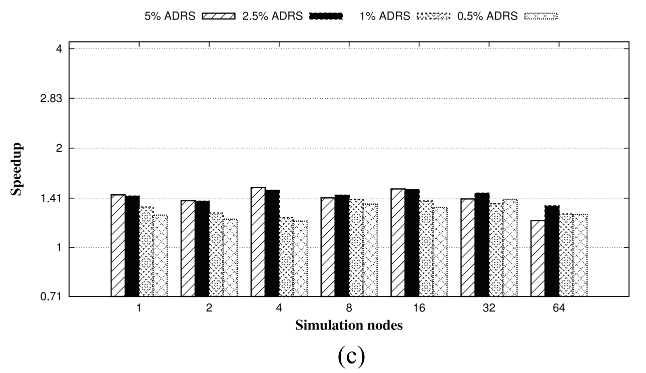
3) Accuracy Analysis: When final Pareto solutions are not available, a practical stopping criterion is to define the amount of time given for the optimization process and to stop the exploration once that time is elapsed. In this analysis, we decided to fix the stopping criteria after 3.5 days of exploration run-time, representing the 1% of the sequential exhaustive exploration time for the target experimental setup . This value has been considered constant over the different design environments and we focused on the quality of the solution found by the different methods.[2]

Fig. 11 shows the improvement in the final result quality obtained by the proposed DESPERATE++ in reference to the alternative algorithms when considering 3.5 days of simulation run-time as stopping criterion. The quality improvement is measured in terms of ADRS. In fact, evaluating with a

[2]The tuning of a time-based stopping criterion is usually a hard task since there is the possibility to do not find any good solution within the allocated time slot. To reduce this risk, the designer can allocate a fraction of the exhaustive search (e.g., 1% as for the analyzed case) that makes him confident with the results and then converting it in the actual stopping criterion. For tuning this stopping criterion in an unknown design space, an approximate estimation of the length of the sequential exhaustive exploration time can be extrapolated once the first γ architectural configurations in the base population P are simulated, i.e., after the first algorithm iteration. This is done by computing the average simulation time over P and multiplying it for the number of design configurations in the design space.







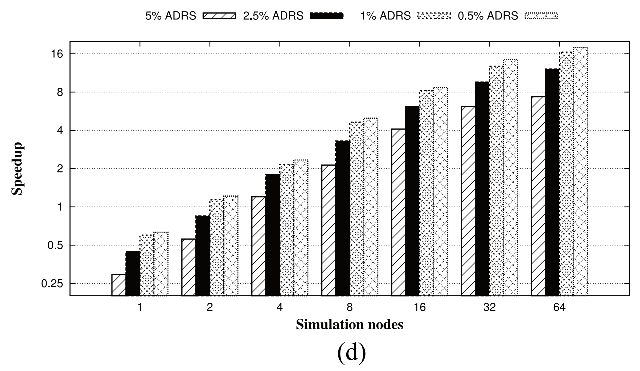
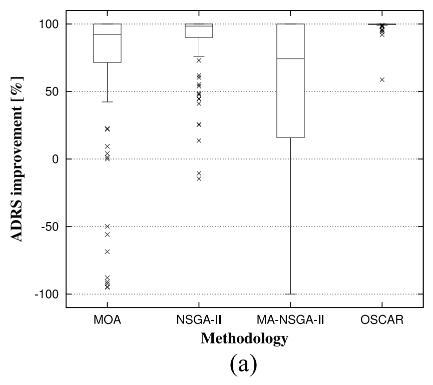
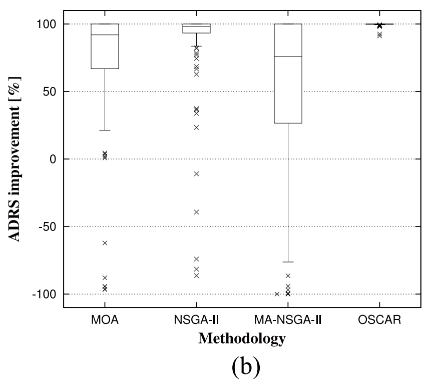


Fig. 10. Speedup of DESPERATE++ with respect to the reference state-of-the-art algorithms by varying the number of simulation nodes and the ADRS. Comparison to (a) MOA (parallel), (b) NSGA-II (parallel), (c) MA-NSGA-II (parallel and prediction-based, $\gamma = 128$), and (d) OSCAR (sequential and prediction-based).





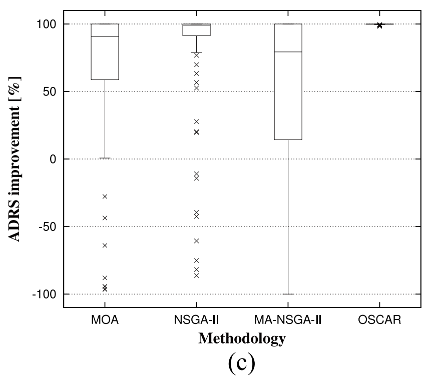


Fig. 11. ADRS improvement after 3.5 days of exploration run-time (i.e., 1% of the sequential exhaustive exploration time) for different optimization algorithms and number of simulation nodes. (a) Using 16 simulation nodes. (b) Using 32 simulation nodes. (c) Using 64 simulation nodes.

single number two sets of Pareto points in a multidimensional objective space would be otherwise troublesome [54].

Given the variability introduced by the randomness in the optimization algorithms, Fig. 11 reports by box-plot diagrams the median and the distribution of the ADRS improvement. We focused the analysis for those cases where the number of simulation nodes are equal to 16, 32, and 64. In those cases, DESPERATE++ overcomes on average all reference methodologies with an ADRS improvement greater than 50%. More compact distributions are shown for the ADRS improvement in the cases of NSGA-II and OSCAR with respect to MA-NSGA-II and MOA. This was expected considering the results shown previously in Fig. 10 since DESPERATE++ is converging faster to good solutions and thus the additional time can be used to refine the quality of the results.

4) Utilization of the Host Computing Resources: Fig. 12 reports the analysis of the average system utilization obtained when running the exploration until reaching a Pareto front characterized by 1% of ADRS.

In this analysis, MA-NSGA-II algorithm is evaluated for all the considered settings $\gamma \in \{64, 128, 256\}$. The system utilization improves significantly when increasing the population size. However, when using 64 simulation nodes, MA-NSGA-II provides the best exploration performance with $\gamma = 128$ (Fig. 8). In fact, by increasing the population size to $\gamma = 256$, MA-NSGA-II performance degrades and DESPERATE++ reports larger speedups (Fig. 8). Increasing the population size to 256 makes every generation longer. Regardless of the better utilization (Fig. 12), a population setting of $\gamma = 256$ has a worse exploration performance than

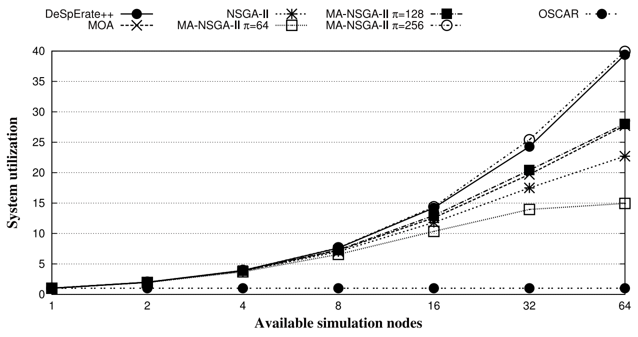


Fig. 12. Utilization of the simulation nodes when running the exploration until a Pareto front with 1% ADRS is reached.

a population setting of $\gamma = 128$ (Fig. 8). In fact, a smaller population size might reduce the system utilization but avoids to waste time in simulating suboptimal configurations during the earliest generations and gives the possibility to quickly exploit the new knowledge gathered at each generation. The best setting of γ is strongly problem dependent and cannot be generalized.

The worst case in terms of system utilization is OSCAR that is a sequential DSE strategy [11] thus using a single simulation node regardless the available number of nodes (Fig. 12). The MA-NSGA-II $- \gamma = 64$ (population size set to $\gamma = 64$) follows with the lowest utilization among the remaining approaches. The NSGA-II and MOA algorithms (both with a population size set to $\gamma = 64$) are characterized by an utilization higher than MA-NSGA-II $- \gamma = 64$ since their population is not shrunk down by predictive simulation pruning techniques.

The proposed predictive simulation scheduler integrated in DESPERATE++ represents the best methodology to exploit parallel DSE environments with an utilization very close to the one obtained for MA-NSGA-II $- \gamma = 256$ (Fig. 12) but with a significant exploration speedup (Fig. 8).

C. Validation of the Prediction Models

Results on the accuracy of the analytic models estimating the simulation time and the nondominated rank are reported respectively in Fig. 13(a) and (b). Relative model errors are inferred at run-time during the algorithm execution. At each generation g, we estimate the mean relative error (MRE) relatively to the metric excursion measured so far.

Fig. 13 shows the MRE trends depending on the algorithm generation in terms of mean, 10th and 90th percentile of the MRE. The simulation time prediction model [Fig. 13(a)] has an initial mean MRE of 10% that decreases quickly below 5% within few generations. The MRE for the nondominated rank model [Fig. 13(b)] has a peculiar behavior where the minimum mean MRE is obtained between the 5th and the 10th generation. After reaching its minimum, the MRE increases slightly. In fact, the nondominated rank is defined on the base of the simulations launched so far (Fig. 14). By increasing the number of simulated design configurations, the nondominated ranking function $r(x)$ becomes more complex. We represent this concept graphically in Fig. 14, where an additional simulation launched during the generation g increases the granularity

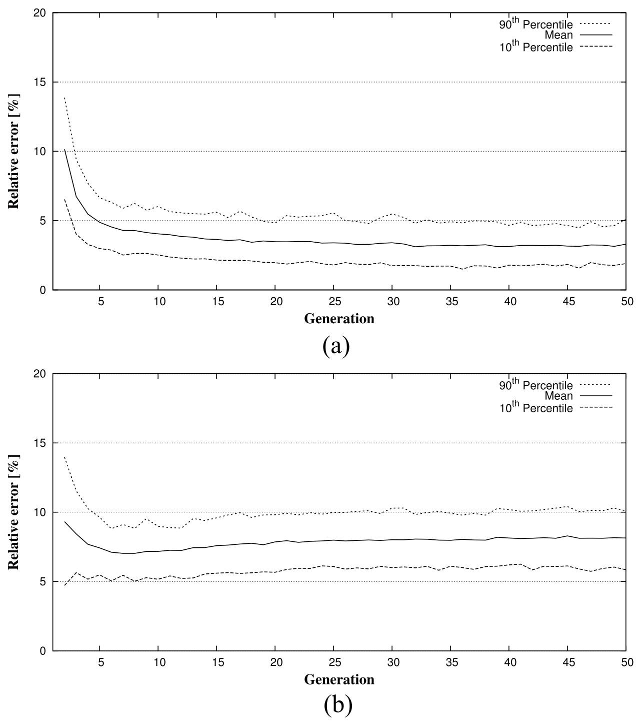
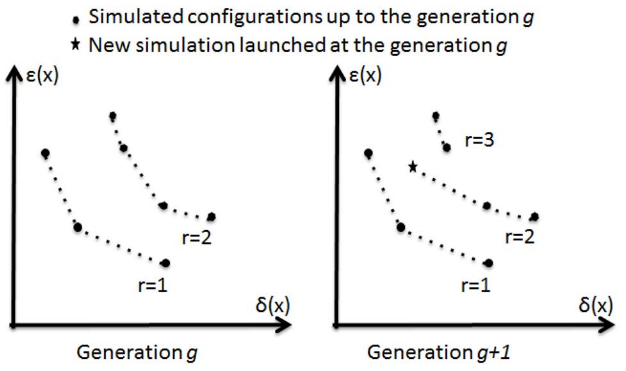


Fig. 13. Relative errors for the analytic prediction models. (a) Relative error for the simulation time prediction model. (b) Relative error for the rank prediction model.



Simulated configurations up to the generation g
New simulation launched at the generation g
ε(x)
δ(x)
r=1
r=2
Generation g
ε(x)
δ(x)
r=3
r=2
r=1
Generation g+1

Fig. 14. Variation in the nondominated ranking function introduced by the execution of a single simulation after the generation g.

representation of the nondominated rank for the generation $g + 1$. By adding one simulation, the ranking of some points changes and additional complexity in modeling the new ranking function is introduced. At initial generations, the prediction accuracy of $\hat{r}(x) \sim r(x)$ is improving, but after some generations the MRE increases again. Regardless this peculiar behavior, the rank model is characterized by an MRE between 15% and 5%.

D. Overhead Analysis

The overall exploration time is the sum of two components: 1) the evaluation time needed to gather results from the simulation nodes and 2) the overhead of the exploration algorithm needed to implement the optimization including the time spent to train and to use the analytic models.

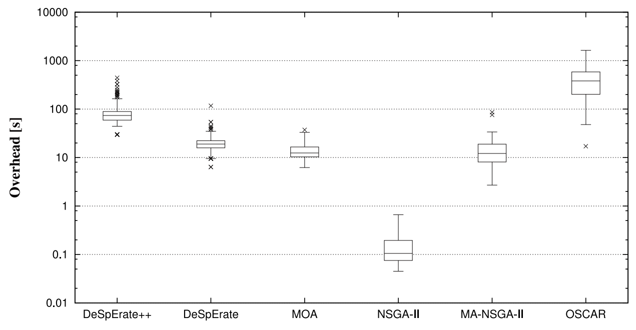


Fig. 15. Box-plot of execution time distribution of algorithm overheads for identifying a solution characterized by 1% of ADRS.

Fig. 15 reports the box-plot of the distribution of the algorithms' overhead accumulated during the exploration process until solutions characterized by 1% ADRS are identified. Overheads are subject to variations that depend on different sources such as the number of points in the model training sets and the number of simulations actually launched at each generation.

OSCAR shows the highest overhead that is most of the times from 100 to 1000 s with a mean of 450 s. The proposed DESPERATE++ follows with an overhead from 50 to 125 s and a mean value of 80 s. The difference between the two approaches is due to the simpler prediction model adopted by the proposed techniques with respect to OSCAR.

The overhead of DESPERATE++ is mainly related to the nondominated rank prediction model that must be retrained at every generation in order to learn the new ranking function. The baseline DESPERATE version not including the nondominated rank prediction model has a significantly lower overhead with a mean of 20 s. The overhead difference of 60 s in average between the two methods shall be traded off in reference to the DESPERATE++ exploration speedup of 1.3× (Fig. 7). The proposed methodology saves about 30% of simulation time that is generally much longer than 60 s overhead.

The MOA algorithm has an overhead very close to the DESPERATE algorithm. The main difference of DESPERATE with respect to MOA is the inclusion of the simulation time prediction model [the same model as in DESPERATE++, (7)]. The small overhead difference of DESPERATE in reference to MOA demonstrates the efficiency of the simulation time prediction model.

The NSGA-II has a very low overhead due to its simplicity in selecting the simulations to run. The MA-NSGA-II overhead is about 15 s due to the necessity of managing the analytic performance model [23]. A similar overhead characterizes the MOA algorithm. The MOA overhead is mainly related to the management of the Markov model for the probability distribution of good solution candidates.

Note that, the average simulation speed of the SESC simulator on our machines (Intel Xeons cores at 3 GHz) for executing the applications under study is about 381 Kcycles/s. This means that the average execution time overhead for the entire exploration of DESPERATE++ is equivalent to the simulation run-time of approximately 30 Mcycles. Thus, it is negligible with respect to the actual simulation requirements of the applications investigated in this paper. It is worth noting that, the longer the simulations the lower is the relative algorithm overhead, since it is not dependent on the complexity of the simulated application.

E. Analysis on the Generality of Results

The results presented so far have been obtained by using the SESC simulator [7] for a set of 12 benchmarks representing three data-sets for each one of the four considered SPLASH-2 applications [17]. The target platform modeled by SESC is homogeneous, however, the proposed approach is not only limited on this class of architectures and we envision the possibility to adopt it for other problems, such as in the context of heterogeneous architectures.

To support our claim, we complement the results reported so far by evaluating the proposed approach for different DSE problems. We adopt the DTLZ test problems [59], a set of seven analytic multiobjective test problems explicitly designed for evaluating the performance of DSE heuristics. By using these analytic test problems it is possible to compute the exact Pareto optimal solutions without facing the computational burden of actually running several computational expensive simulations.

For modeling the execution time variability typical of architectural simulators (Fig. 1), the optimization algorithms are evaluated by considering the last objective function of each DTLZ problem as simulation time cost. Thus, we consider as cost of evaluating a given design configuration the value returned by an analytic function at that design configuration.

The DTLZ design space we used is composed of seven design parameters, each one spanning over 30 different values. Overall, the design space consists of about $2 \cdot 10^{10}$ different configurations.

Results in terms of exploration speedup are reported in Fig. 16. As noted for the SESC optimization, the proposed DESPERATE++ approach overcomes all the reference methodologies by reporting in general a speedup factor higher than 1. An exception is the OSCAR methodology, that represents the fastest sequential exploration approach reporting a speedup factor smaller than one when considering one or two simulation nodes. Speedup values are generally lower for the DTLZ problems (Fig. 16) than for the SESC optimization problem (Fig. 10). This result is mainly related to the error in the simulation time prediction model of (7). For each of the multiobjective DTLZ problems, we represent the simulation time as the last objective function of the multiobjective problem. The DTLZ problems include strong nonlinearities that let the prediction be more difficult than for the simulation time of the SESC simulator. Fig. 17 reports the MRE of the approximate simulation time prediction model in (7) for the DTLZ test problems. We can observe that its mean value at the first generation is about 30% that is significantly larger than what measured for the SESC problem [Fig. 13(a)]. This error in turn influences negatively the performance of the proposed DESPERATE++ algorithm.

Note that, the simulation time prediction model presented in this paper can be easily replaced by other analytic models.

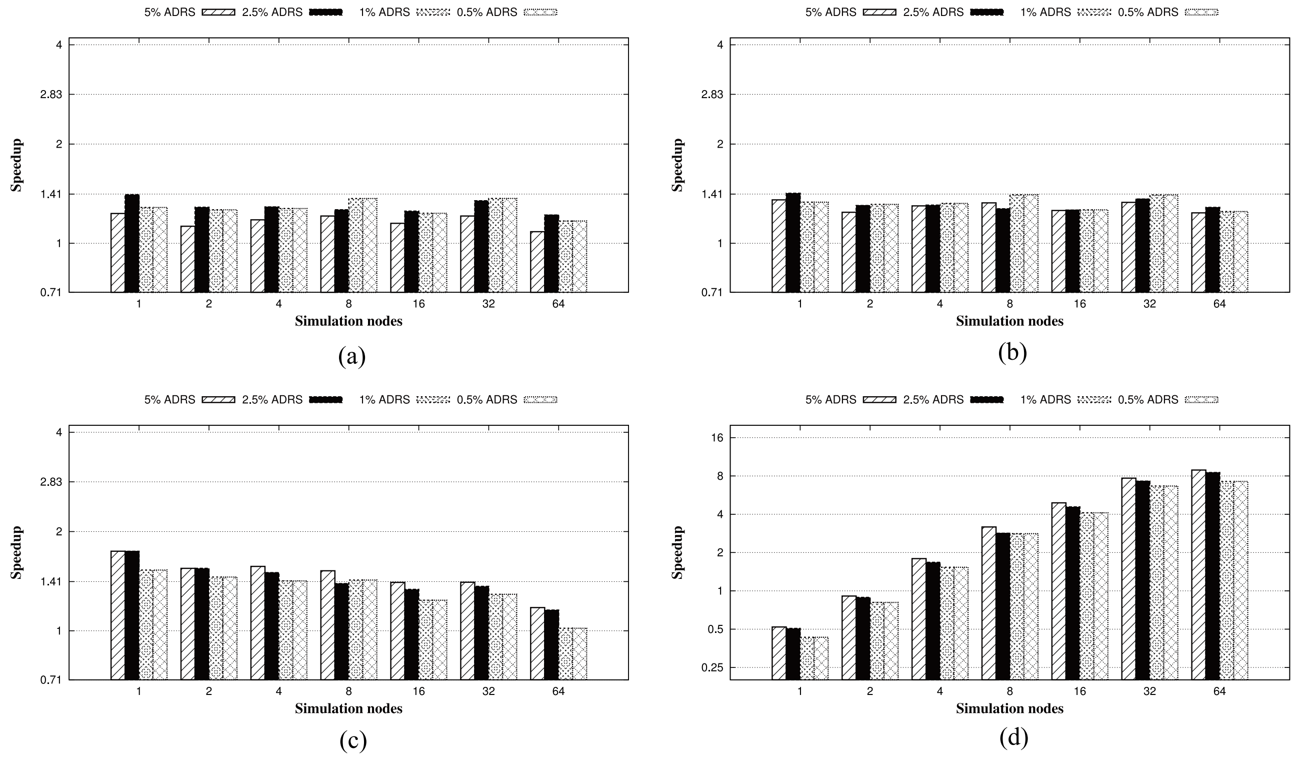


Fig. 16. Speedup of DESPERATE++ by varying the number of nodes hosting the simulations and the ADRS. Comparison to (a) MOA (parallel), (b) NSGA-II (parallel), (c) MA-NSGA-II (parallel and prediction-based, $\gamma = 128$), and (d) OSCAR (sequential and prediction-based).

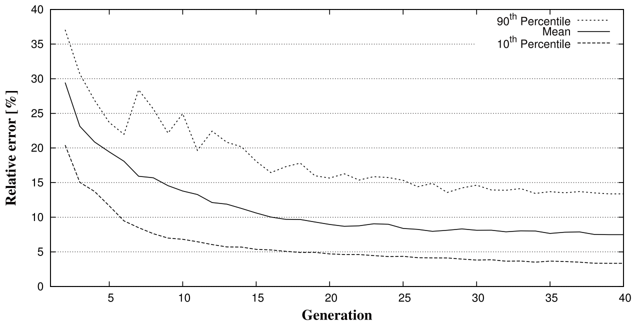


Fig. 17. Simulation time prediction error for the DTLZ test problems.

The model in (7) has been chosen in this paper because: 1) it has been previously proposed in the context of EDA such as the baseline MOA [53]; 2) it is accurate enough to provide acceptable approximation errors [Figs. 13(a) and 17]; and 3) it introduces low overheads (see Fig. 15).

VI. Conclusion

In this paper, we presented DESPERATE++, a DSE methodology to jointly using analytic prediction models and parallel computing resources to speedup the optimization process for multicore systems. The proposed approach exploits two orthogonal analytic prediction models. A simulation time prediction model defines a simulation schedule aware of the underlying computational resources. A configuration quality prediction model speeds-up the optimization by focusing the exploration process on the most promising design region.

The proposed approach has been evaluated in reference to state of the art DSE strategies reporting up to 4× speedup in reference to parallel DSE approaches. A speedup of up to 16× has been obtained with respect to the best sequential exploration method.

References

[1] K. Keutzer, A. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli, "System-level design: Orthogonalization of concerns and platform-based design," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 19, no. 12, 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. Evol. Comput.*, vol. 10, no. 3, pp. 358–374, Jun. 2006.

[3] G. Palermo, C. Silvano, and V. Zaccaria, "Multi-objective design space exploration of embedded system," *J. Embedded Comput.*, vol. 1, no. 3, pp. 305–316, 2006.

[4] 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," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 29, no. 6, pp. 911–924, Jun. 2010.

[5] G. Palermo, C. Silvano, and V. Zaccaria, "ReSPIR: A response surface-based Pareto iterative refinement for application-specific design space exploration," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 28, no. 12, pp. 1816–1829, Dec. 2009.

[6] R. Jahr, H. Calborean, L. Vintan, and T. Ungerer, "Boosting design space explorations with existing or automatically learned knowledge," in *Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance* (Lecture Notes in Computer Science 7201), J. Schmitt, Ed., Berlin, Germany: Springer, 2012, pp. 221–235.

[7] J. Renau *et al.*, (Jan. 2005). *SESC Simulator*. [Online]. Available: http://sesc.sourceforge.net

[8] B. Catanzaro, K. Keutzer, and B.-Y. Su, "Parallelizing CAD: A timely research agenda for EDA," in *Proc. 45th ACM/IEEE Design Autom. Conf. (DAC)*, Anaheim, CA, USA, Jun. 2008, pp. 12–17.

[9] L. Stok, "Developing parallel EDA tools [the last byte]," *IEEE Des. Test*, vol. 30, no. 1, pp. 65–66, Feb. 2013.

[10] K. Keutzer, P. Li, L. Shang, and H. Zhou, "A special section on multicore parallel CAD: Algorithm design and programming," *ACM Trans. Design Autom. Electron. Syst.*, vol. 16, no. 3, pp. 21:1–21:2, Jun. 2011.

[11] G. Mariani, G. Palermo, V. Zaccaria, and C. Silvano, "OSCAR: An optimization methodology exploiting spatial correlation in multicore design spaces," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 31, no. 5, pp. 740–753, May 2012.

[12] B. Li, L. Peng, and B. Ramadass, "Accurate and efficient processor performance prediction via regression tree based modeling," *J. Syst. Archit.*, vol. 55, nos. 10–12, pp. 457–467, 2009.

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

[14] T. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, Seatle, WA, USA, Nov. 2011, pp. 1–12.

[15] J. Chen, M. Annavaram, and M. Dubois, "SlackSim: A platform for parallel simulations of CMPs on CMPs," *SIGARCH Comput. Archit. News*, vol. 37, no. 2, pp. 20–29, Jul. 2009.

[16] A. Patel, F. Afram, S. Chen, and K. Ghose, "MARSS: A full system simulator for multicore x86 CPUs," in *Proc. 48th ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, New York, NY, USA, Jun. 2011, pp. 1050–1055.

[17] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," in *Proc. 22nd Annu. Int. Comput. Archit. Symp.*, Santa Margherita Ligure, Italy, 1995, pp. 24–36.

[18] G. Mariani, G. Palermo, V. Zaccaria, and C. Silvano, "DeSpErate: Speeding-up design space exploration by using predictive simulation scheduling," in *Proc. Design Autom. Test Europe Conf. Exhibit. (DATE)*, Mar. 2014, pp. 1–4.

[19] C. Silvano, W. Fornaciari, and E. Villar, *Multi-objective Design Space Exploration of Multiprocessor SoC Architectures: The MULTICUBE Approach*. New York, NY, USA: Springer, 2011.

[20] C. Erbas, *System-level Modelling and Design Space Exploration for Multiprocessor Embedded System-on-chip Architectures*. Amsterdam, The Netherlands: Amsterdam Univ. Press, 2007.

[21] J. Panerati and G. Beltrame, "A comparative evaluation of multi-objective exploration algorithms for high-level design," *ACM Trans. Design Autom. Electron. Syst.*, vol. 19, pp. 15:1–15:22, Mar. 2014.

[22] G. Martin, "Overview of the MPSoC design challenge," in *Proc. 43rd Annu. Design Autom. Conf. (DAC)*, New York, NY, USA, 2006, pp. 274–279.

[23] 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. Appl. Specific Process. (SASP)*, San Francisco, CA, USA, 2009, pp. 21–28.

[24] R. Piscitelli and A. Pimentel, "Design space pruning through hybrid analysis in system-level design space exploration," in *Proc. Design Autom. Test Europe Conf. Exhibit. (DATE)*, Dresden, Germany, 2012, pp. 781–786.

[25] G. Palermo, C. Silvano, and V. Zaccaria, "Discrete particle swarm optimization for multi-objective design space exploration," in *Proc. 11th EUROMICRO Conf. Digit. Syst. Design Archit. Methods Tools (DSD)*, Parma, Italy, 2008, pp. 641–644.

[26] A. Tumeo *et al.*, "Mapping pipelined applications onto heterogeneous embedded systems: A Bayesian optimization algorithm based approach," in *Proc. 7th IEEE/ACM Int. Conf. Hardw./Softw. Codesign Syst. Synth. (CODES+ISSS)*, New York, NY, USA, 2009, pp. 443–452.

[27] M. Ceriani, F. Ferrandi, P. L. Lanzi, D. Sciuto, and A. Tumeo, "Multiprocessor systems-on-chip synthesis using multi-objective evolutionary computation," in *Proc. ACM 12th Annu. Conf. Genet. Evol. Comput. (GECCO)*, New York, NY, USA, 2010, pp. 1267–1274.

[28] H.-Y. Liu and L. Carloni, "On learning-based methods for design-space exploration with high-level synthesis," in *Proc. 50th ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Austin, TX, USA, May 2013, pp. 1–7.

[29] D. Sheldon, F. Vahid, and S. Lonardi, "Soft-core processor customization using the design of experiments paradigm," in *Proc. Conf. Design Autom. Test Europe (DATE)*, Nice, France, 2007, pp. 821–826.

[30] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe, "SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling," in *Proc. ACM 30th Annu. Int. Symp. Comput. Archit. (ISCA)*, New York, NY, USA, 2003, pp. 84–97.

[31] E. Perelman, G. Hamerly, M. V. Biesbrouck, T. Sherwood, and B. Calder, "Using SimPoint for accurate and efficient simulation," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 31, no. 1, pp. 318–319, 2003.

[32] M. V. Biesbrouck, B. Calder, and L. Eeckhout, "Efficient sampling startup for SimPoint," *IEEE Micro*, vol. 26, no. 4, pp. 32–42, Jul. 2006.

[33] M. Monchiero, J. H. Ahn, A. Falcón, D. Ortega, and P. Faraboschi, "How to simulate 1000 cores," *SIGARCH Comput. Archit. News*, vol. 37, pp. 10–19, Jul. 2009.

[34] O. Villa, A. Tumeo, S. Secchi, and J. B. Manzano, "Fast and accurate simulation of the Cray XMT multithreaded supercomputer," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 12, pp. 2266–2279, Dec. 2012.

[35] R. Sinha, A. Prakash, and H. Patel, "Parallel simulation of mixed-abstraction SystemC models on GPUs and multicore CPUs," in *Proc. 2012 17th Asia South Pac. Design Autom. Conf. (ASP-DAC)*, Sydney, NSW, Australia, Jan. 2012, pp. 455–460.

[36] S. Khan, P. Xekalakis, J. Cavazos, and M. Cintra, "Using predictive modeling for cross-program design space exploration in multicore systems," in *Proc. 16th Int. Conf. Parallel Archit. Compilation Tech. (PACT)*, Brasov, Romania, Sep. 2007, pp. 327–338.

[37] H. Cook and K. Skadron, "Predictive design space exploration using genetically programmed response surfaces," in *Proc. ACM 45th Annu. Design Autom. Conf. (DAC)*, New York, NY, USA, 2008, pp. 960–965.

[38] P. J. Joseph, K. Vaswani, and M. J. Thazhuthaveetil, "Construction and use of linear regression models for processor performance analysis," in *Proc. Symp. High Perform. Comput. Archit.*, Austin, TX, USA, 2006, pp. 99–108.

[39] P. J. Joseph, K. Vaswani, and M. J. Thazhuthaveetil, "A predictive performance model for superscalar processors," in *Proc. 39th Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, Washington, DC, USA, 2006, pp. 161–170.

[40] E. Ipek *et al.*, "Efficient architectural design space exploration via predictive modeling," *ACM Trans. Archit. Code Optim.*, vol. 4, no. 4, pp. 1–34, 2008.

[41] Q. Guo, T. Chen, Y. Chen, L. Li, and W. Hu, "Microarchitectural design space exploration made fast," *Microprocess. Microsyst.*, vol. 37, no. 1, pp. 41–51, 2013.

[42] M. Zuluaga, A. Krause, P. Milder, and M. Püschel, "'Smart' design space sampling to predict Pareto-optimal solutions," in *Proc. 13th ACM SIGPLAN/SIGBED Int. Conf. Lang. Compilers Tools Theory Embedded Syst. (LCTES)*, New York, NY, USA, 2012, pp. 119–128.

[43] G. Mariani *et al.*, "A correlation-based design space exploration methodology for multi-processor systems-on-chip," in *Proc. 47th ACM/IEEE Design Autom. Conf. (DAC)*, Anaheim, CA, USA, 2010, pp. 120–125.

[44] W. Zhi-xin and J. Gang, "A parallel genetic algorithm in multi-objective optimization," in *Proc. Control Decis. Conf. (CCDC) Chinese*, Guilin, China, Jun. 2009, pp. 3497–3501.

[45] D. Thain, T. Tannenbaum, and M. Livny, "Distributed computing in practice: The Condor experience," *Concurr. Pract. Exp.*, vol. 17, nos. 2–4, pp. 323–356, 2005.

[46] H. Jordan *et al.*, "A multi-objective auto-tuning framework for parallel codes," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, Los Alamitos, CA, USA, 2012, pp. 10:1–10:12.

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

[48] T. Stanley and T. Mudge, "A parallel genetic algorithm for multiobjective microprocessor design," in *Proc. 6th Int. Conf. Genet. Algorithms*, San Francisco, CA, USA, 1995, pp. 597–604.

[49] E. K. Ardestani and J. Renau, "ESESC: A fast multicore simulator using time-based sampling," in *Proc. 2013 IEEE 19th Int. Symp. High Perform. Comput. Archit. (HPCA)*, Washington, DC, USA, pp. 448–459.

[50] S. Shakya and R. Santana, "An EDA based on local Markov property and Gibbs sampling," in *Proc. 10th Annu. Conf. Genet. Evol. Comput. (GECCO)*, New York, NY, USA, 2008, pp. 475–476.

[51] R. Santana *et al.*, "MATEDA: A suite of EDA programs in Matlab," Dept. Comput. Sci. Artif. Intell., Univ. Basque Country, Leioa, Spain, Tech. Rep. EHU-KZAA-IK-2/09, Feb. 2009.

[52] A. Blum, *Neural Networks in C++: An Object-Oriented Framework for Building Connectionist Systems*. New York, NY, USA: Wiley, 1992.

[53] M. Pelikan and K. Sastry, "Fitness inheritance in the Bayesian optimization algorithm," in *Genet. Evol. Comput.* (Lecture Notes in Computer Science 3103), Berlin, Germany: Springer, 2004, pp. 48–59.

[54] T. Okabe, Y. Jin, and B. Sendhoff, "A critical survey of performance indices for multi-objective optimization," in *Proc. IEEE Congr. Evol. Comput.*, Canberra, ACT, Australia, 2003, pp. 878–885.

[55] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: NSGA-II,” *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.
[56] Y. Jin, M. Olhofer, and B. Sendhoff, “Managing approximate models in evolutionary aerodynamic design optimization,” in *Proc. Congr. Evol. Comput.*, vol. 1. Seoul, Korea, 2001, pp. 592–599.
[57] P. van Stralen and A. D. Pimentel, “Using chip multithreading to speed up scenario-based design space exploration: A case study,” in *Proc. ACM 6th Workshop Rapid Simulat. Perform. Eval. Methods Tools (RAPIDO)*, New York, NY, USA, 2014, pp. 1:1–1:7.
[58] G. Mariani, G. Palermo, V. Zaccaria, and C. Silvano, “Design-space exploration and runtime resource management for multicores,” *ACM Trans. Embedded Comput. Syst.*, vol. 13, pp. 20:1–20:27, Sep. 2013.
[59] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler, “Scalable multi-objective optimization test problems,” in *Proc. Congr. Evol. Comput. (CEC)*, vol. 1. Honolulu, HI, USA, 2002, pp. 825–830.