5-10-2018

Uncertainty Aware Mapping of Embedded Systems for Reliability, Performance, and Energy

Wenkai Guan
Marquette University

Milad Ghorbani Moghaddam
Marquette University

Cristinel Ababei
Marquette University, cristinel.ababei@marquette.edu

Contents

Uncertainty Aware Mapping of Embedded Systems for Reliability, Performance, and Energy

Wenkai Guan
Marquette University, Electrical and Computer Engineering Milwaukee, WI
Milad Ghorbani Moghaddam
Marquette University, Electrical and Computer Engineering Milwaukee, WI
Cristinel Ababei
Marquette University, Electrical and Computer Engineering Milwaukee, WI

Abstract:

Duo to technology downscaling, embedded systems have increased in complexity and heterogeneity. Increasingly large process, voltage, and temperature variations negatively affect the design and optimization process of these systems. These factors contribute to increased uncertainties that in turn undermine the accuracy and effectiveness of traditional design approaches. In this paper, we formulate the problem of uncertainty aware mapping for multicore embedded systems as a multi-objective optimization problem. We present a solution to this problem that integrates uncertainty models as a new design methodology constructed with Monto Carlo and evolutionary algorithms. The methodology is uncertainty aware because it is able to model uncertainties in design parameters and to identify robust design solutions that limit the influence of these uncertainties onto the objective functions. The proposed design methodology is implemented as a tool that can generate the robust Pareto frontier in the objective space formed by reliability, performance, and energy consumption.

Keywords

Embedded systems, uncertainties, robust mapping, reliability, performance, energy consumption

Introduction

Continuous technology downscaling and the increase in size of embedded systems resulted in new design challenges: increased design uncertainties due to variations in fabrication processes, supply voltage, and temperatures;[1] poor reliability and performance degradation caused by elevated rates of faults and increasingly adverse aging mechanisms;[2] and increased design complexity caused by heterogeneity of the hardware platform, diversity in hardware and software components, and new communication infrastructures such as networks-on-chip.[3] These factors make for design parameters not to be deterministic anymore; instead they become less precisely known or more uncertain. If these design parameters become uncertain then, the path of explored solutions during design space exploration (DSE) may become uncertain and divergent from the path towards the true optimal solution.

In this context, it becomes desirable to be able to quantify such divergence and to develop a design methodology capable of finding design solutions that are the most likely, with certain confidence, to be robust against uncertainties. Normally, such design solutions would be points on the Pareto frontier generated during the design space exploration, an example of which is shown in Fig. 1.a. However, when one considers uncertainties in the design process, the traditional Pareto surface in the solution space becomes uncertain as shown in Fig. 1.b.

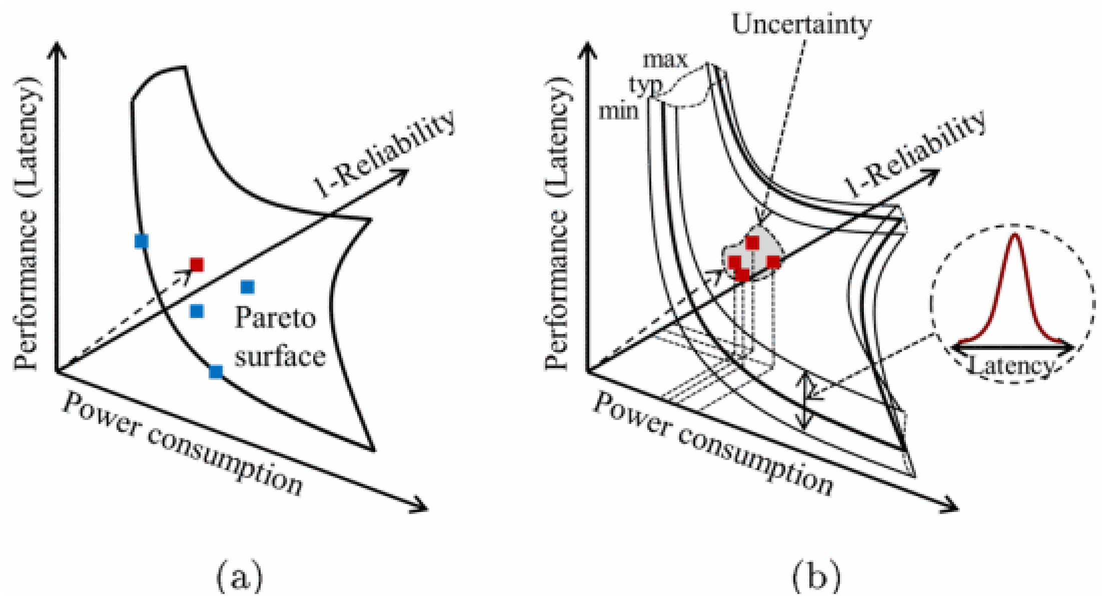

Fig. 1. (a) Pareto frontier surface in traditional embedded systems design. (b) Uncertain pareto surface where a design point degenerates into multiple solution points.

In this paper, we propose a design method that is able to identify robust design points on this uncertain Pareto frontier. The proposed method models and handles uncertainties directly. This method is implemented as a computer program (i.e., a design tool) that integrates uncertainty models and algorithms to solve the problem of mapping for hardware/software (HW/SW) design of embedded systems. Our tool chooses as the best final solution the one closest to the “origin” of the 3D objective space from Fig. 1.b. This represents a compromise among all three objectives. However, the designer can pick a different solution. For example, if performance is really the most important for some application, then, a design point with the best performance can be selected, but likely with worse reliability and power consumption.

Related Work

The problem of HW/SW co-design for embedded systems has been studied extensively in the past. It was formulated as multi-objective optimization in studies of system-level synthesis[4,5] as well as of platform configuration.[6] Several previous solutions have been integrated into computer aided design automation tools.[7–8,9] These tools facilitate flexible system-level performance evaluation by providing support for mapping a behavioral application specification to an architecture specification.[10,11] Also, reliability has become a primary design concern alongside traditional design objectives.[12,13] However, the majority of the previous work did not consider uncertainty or reliability in the design process of embedded systems. The studies in[14,15] are recent attempts to capture uncertainty in the process of optimization of embedded systems. In this paper, we integrate such techniques in a comprehensive approach that considers also performance and energy consumption, not only reliability. The main contributions include: **1**) We solve the mapping problem for general purpose embedded systems while considering simultaneously reliability, execution time, and energy consumption. The proposed solution is implemented as a design space exploration method that uses the Nondominated Sorting Genetic Algorithm (NSGA-II). 2) We model and deal with uncertainty in design parameters. We investigate different levels of injected uncertainty and provide simulation results. 3) We assume the architecture platform to be comprised of both hardware and software components. To the best of our knowledge, our work is the first to address the problem of multi-objective (reliability, performance, and energy) mapping for general purpose embedded systems under uncertainties.

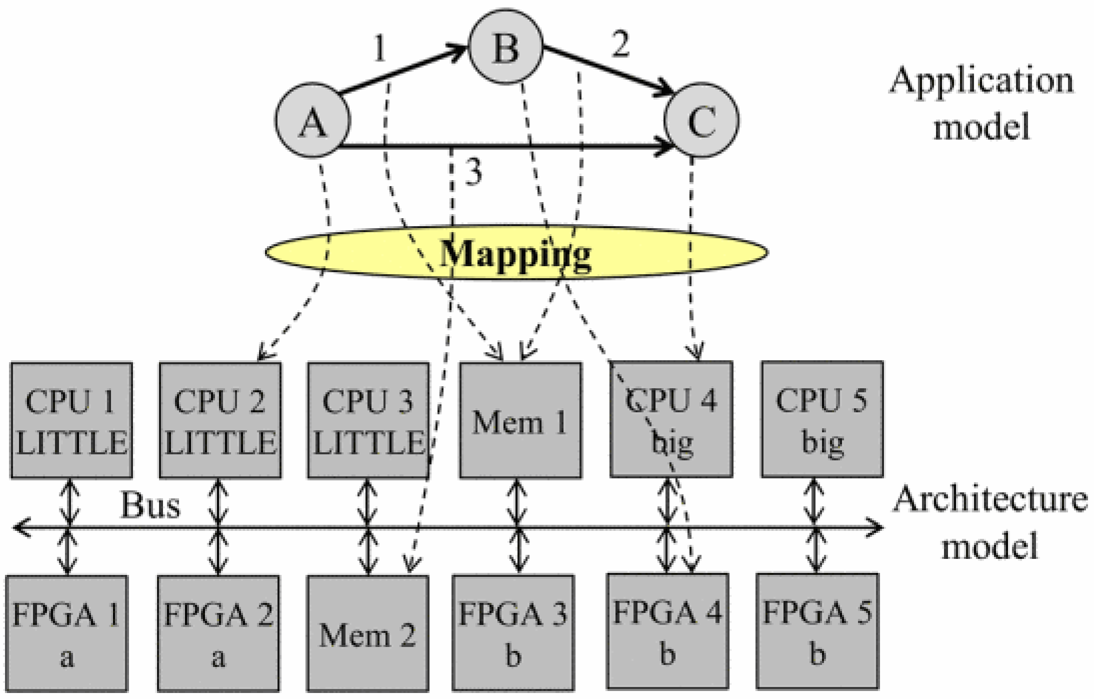

Fig. 2. Illustration of the mapping problem. Tasks A, B, C, and communications 1, 2, 3 are mapped to components of the architecture platform.

Proposed Design Methodology

A. Block Diagram

The proposed design method is an iterative process that uses an enhanced evolutionary algorithm, to solve the problem of *mapping*. The problem of application mapping is the problem of finding the best placement of application tasks and communications between tasks to the architecture platform as illustrated in Fig. 2.

Finding the best placement is done by exploring the design space formed by all possible solutions. This exploration is implemented as an iterative optimization algorithm. The *outer loop* of this iterative process is illustrated in Fig. 3, which shows the block diagram of the proposed design method. The *inner loop*represents the iterative process of the Monte Carlo (MC) simulation technique that we employ for the estimation of objective functions under uncertainty. The primary objectives that we consider in this paper include reliability, performance (measured as execution time), and energy consumption. Thus, the problem we address is a multi-objective objective problem under specified levels of uncertainty. The output of the optimization process illustrated in Fig. 3 is a set of robust solutions that form the robust Pareto frontier in the three dimensional objective space. In the next sections, we describe the primary steps of the proposed design flow.

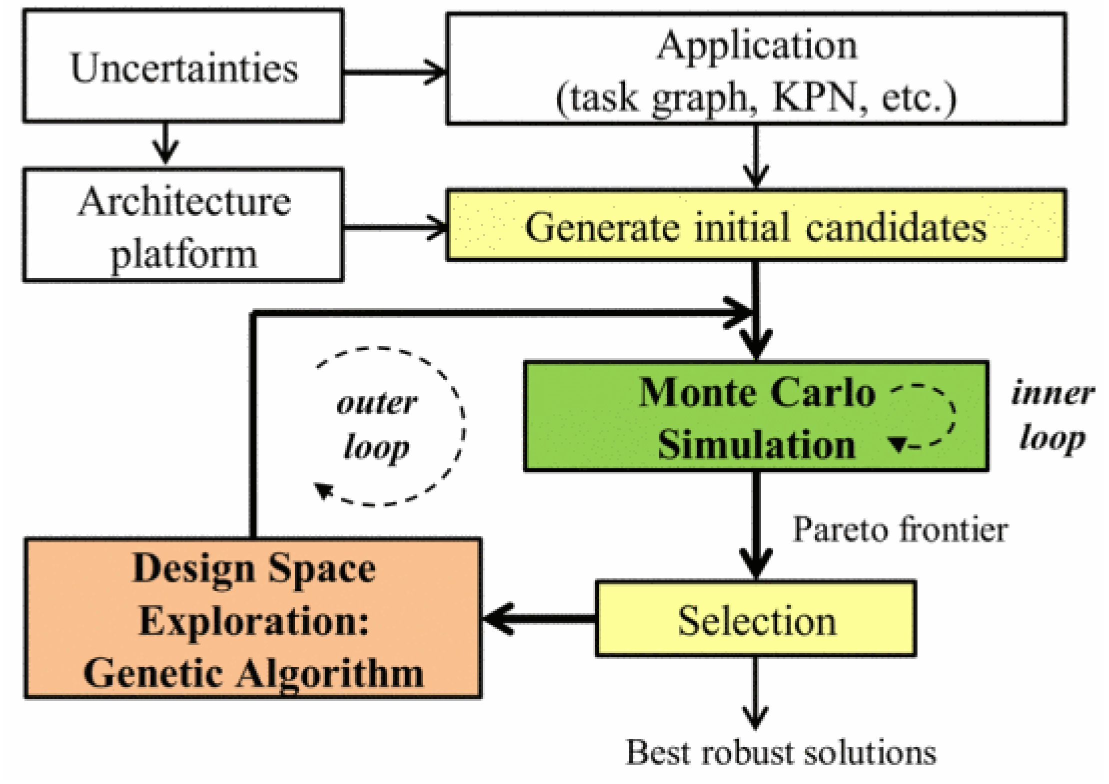

Fig. 3. Block diagram of the proposed design methodology for embedded systems mapping under uncertainties.

B. Uncertainty Modeling

The *Uncertainties* block on the top left-hand side from the diagram in Fig. 3 represents the uncertainty injection process. There has been significant work studying uncertainty in various fields including engineering, mathematics, and other sciences.[16] However, it is generally agreed that there is no single model for handling any type of imperfect information. Therefore, similarly to,[14] we propose to adopt the most general approach to capture uncertainty: design parameters and their variation can be specified as generalized, continuous or discrete, probability distributions in any mixture. Aside from its generality and ability to accommodate any probability distribution, this approach has the advantage of being able to accommodate complementary approaches as well. For instance, we can use uniform distributions to convert interval estimates into the proposed framework. On the limitations side, combining different probability distributions is usually analytically intractable, and therefore we must resort to Monte Carlo simulation based techniques in order to quantify figures of merit (described later). This, in turn, may increase the computational runtime.

Uncertainty can be injected into the application or/and the architecture, depending on what design parameters are assumed to be affected by uncertainties and to what degree. This injection will be done in different amounts or degrees during the design space exploration depicted in Fig. 3. The injection process amounts to generating samples from prespecified probability distributions during the Monte Carlo simulation technique used to evaluate reliability, execution time, and energy. Because we allow working with any type of probability distribution, we must define what is meant by *injecting a given percentage of* *uncertainty* into the design parameters of interest. We do that by pre-specifying the mean and the variance of the probability distributions out of which the sampling is done according to the rules listed in Table I.

The rationale behind the rules presented in Table I can be explained with the help of Fig. 4.

Table I Rules for defining mean and variance of distributions from which sampling must be done to achieve a certain degree of uncertainty injection.

Probability Distribution	Uncertainty 1%	Uncertainty 5%	Uncertainty 10%
$Uniform(\mu,\sigma)$	$\sigma = 0.01 \cdot \mu/\sqrt{3}$	$\sigma = 0.05 \cdot \mu/\sqrt{3}$	$\sigma = 0.1 \cdot \mu/\sqrt{3}$
$Gaussian(\mu,\sigma)$	$\sigma = 0.01 \cdot \mu/3$	$\sigma = 0.05 \cdot \mu/3$	$\sigma = 0.1 \cdot \mu/3$
$Beta(\mu,\sigma)$	$\sigma = 0.01 \cdot \mu/3$	$\sigma = 0.05 \cdot \mu/3$	$\sigma = 0.1 \cdot \mu/3$

For example, let us assume that the uncertainty is modeled for some design parameter with a uniform distribution. Then, modeling 5% of uncertainty in this design parameter during the design space exploration is achieved by having the MC simulation (discussed later in a different section) generate samples from an interval as shown in Fig. 4.a for the case when for example the mean is $\mu = 100$.

That is because the variance (whose square root is the standard deviation, σ) is given by the expression $Var = (b-a)^2/12$. In the case of a Gaussian distribution, samples are generated randomly from a distribution $Gaussian(\mu,\sigma)$ but only samples falling inside the interval $[\mu - 3\sigma, \mu + 3\sigma]$, as shown in Fig. 4.bare accepted, which represent 99.7% of all generated samples. The case of the beta distribution is similar to that of the Gaussian case. The difference is only in the actual confidence level, which can be different from 99.7%. When no uncertainty is injected, the mean value μ becomes the deterministic fixed value for that design parameter. Note that similar rules can be derived for any other type of distribution that we may be interested in using to model parameter uncertainty. For simplicity, in this paper, we restrict ourselves to using uniform and Gaussian distributions for modeling the execution time and the power consumption of architecture components and for modeling the transition probabilities inside the reliability model (discussed later). In addition, beta distribution is used to model failure rates of components, similarly to the study in.[14] However, our framework is flexible and can easily accommodate other probability distributions if embedded designers find their data to fit better such distributions.

C. Application and Architecture Modeling

To model the *Application* in Fig. 3, we use the notation from[17] and model applications using Kahn Process Networks (KPNs), which are very popular models of computation used in embedded systems design.[5,17] An KPN is represented as an application directed graph $G_{AP}(V_{AP}, E_{AP})$ (see Fig. 2). Each vertex $v_i, i \in \{1,..,|V_{AP}|\}$ corresponds to a process or task of G_{AP}. For each vertex vi, we define $B_i = \{e_j \in E_{AP}\}$ to be the set of application channels connected to vertex v_i. When a vertex is mapped to a hardware component, ht_i represents the hardware execution time. When the task can be executed on multiple hardware cores, ht_i becomes a set $ht_i = \{ht_{i1}, ht_{i2}, .., ht_{iU}\}$, where U is the number of hardware cores on which the task can be executed. When a vertex is mapped to a software component, st_i is the software execution time. When the task can be executed on multiple software components, st_i becomes a set $st_i = \{st_{i1}, st_{i2}, .., st_{iV}\}$, where V is the number of software components on which the task can be executed. Each edge $e_j, j \in \{1,..,|E_{AP}|\}$ corresponds to link between two different tasks of G_{AP}. If a communication link is mapped onto a memory core, mt_jrepresents the memory access time, which will be added to the path delay. When the link can be mapped to multiple memory components, mt_j becomes a set $mt_j =$

$\{mt_{j1}, mt_{j2}, .., mt_{jW}\}$, where W is the number of memory components to which the link can be mapped to.

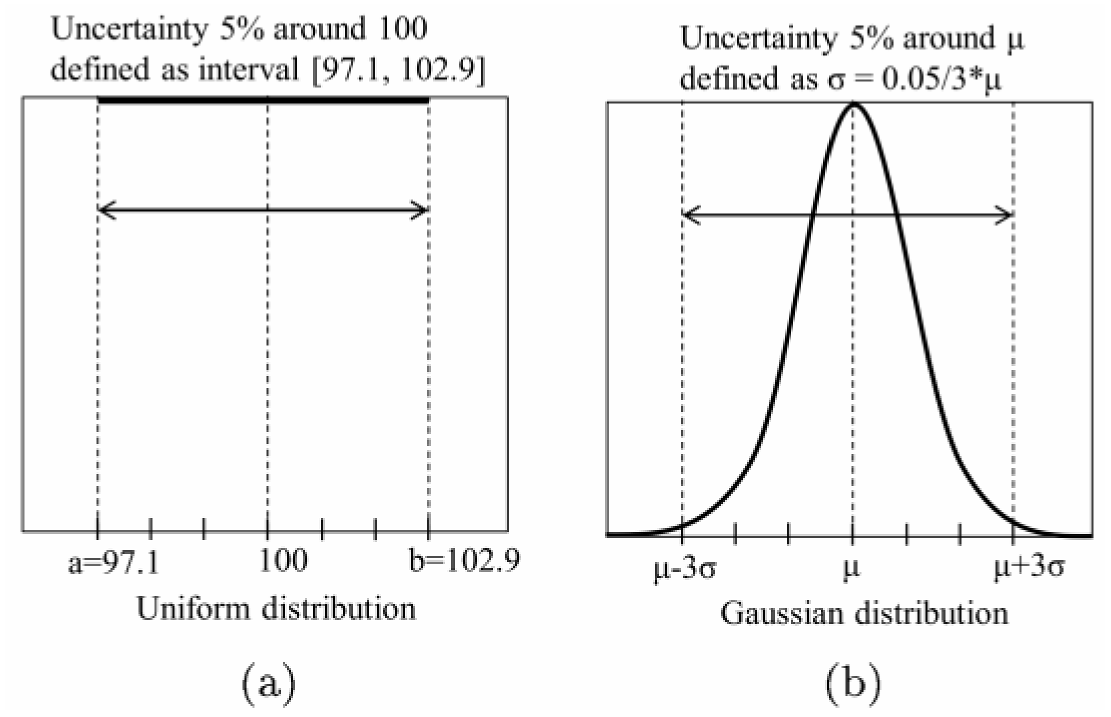

Fig. 4. (a) To inject 5% uncertainty for a parameter characterized by a uniform distribution whose mean is 100 for example, we generate samples from a uniform distribution defined on the interval $[a = \mu - 0.05 \cdot \mu/\sqrt{3}, b = \mu + 0.05 \cdot \mu/\sqrt{3}]$. (b) The interval used for the case of a gaussian distribution whose mean is μ.

The *Architecture* model is also represented by a graph $G_{AR}(V_{AR}, E_{AR})$, where the sets V_{AR} and E_{AR} denote the architecture components and the connections between them. The set of architecture components consists of two disjoint subsets: the set of processing cores (P) that include hardware and software elements and the set of memories (M), $V_{AR} = P \cup M$. The delay of a communication **link** between two different architecture components is denoted as lt_{pq}, with $p, q \in \{1, .., |E_{AR}|\}$. The power dissipations are denoted as w_{pe} for the core p during execution, as w_{me} for the memory core m, and as w_{le} for the communication links. In this paper, we assume that the architecture platform is given because we do not address the problem of architecture synthesis.

D. Design Space Exploration Using Genetic Algorithms

The *Design Space Exploration* block from Fig. 3 is where new mapping solutions are generated and the optimization process takes place. This is a challenging step not only because of the complexity of the problem but also because it must model uncertainties. The mapping problem is a multi-objective optimization problem whose objective functions or quality attributes often conflict. In this paper, we consider the following objectives.

D.1 Objective 1: Reliability

The first objective function is the reliability of the system, which needs to be maximized. To estimate reliability, we use the approach described in[18,19] due to its simplicity. Note that other reliability models can be used here as well. Our framework is generic enough and can employ any reliability model of interest such as that presented in[15] for example.

The reliability model is based on absorbing discrete time Markov chain (DTMC) models, which are graphical models consisting of finite state machine like state graphs.[18] For a given mapping solution, the DTMC model is constructed from the architecture platform of the system. The expression to estimate the architecture based reliability of the system is:

$$R = S(1, n) R_n \quad (1)$$

Where S is called the fundamental matrix of the **DTMC,** $S(i,\ \boldsymbol{j})$ is the expected number of visits to state j starting from state I before it is absorbed, and n is the number of states in the model. The objective of maximizing the reliability of the system can be written as a minimization objective as follows:

$$\min\ \{1 - R\}\ (2)$$

D.2 Objective 2: Execution Time

The second objective function is the one that minimizes the maximum execution or processing time of the critical path from the set of all paths (set denoted as *Path*) inside the application task graph. This minimum value is used as a direct measure of performance, and, using the notations introduced earlier, can be expressed as follows.

$$\min\{\max_{Path}\{\sum_{i \in V_{AP}, i \in Path} ht_{iu} x_{iu} + \sum_{i \in V_{AP}, i \in Path} st_{iv} x_{iv} + \sum_{j \in E_{AP}, j \in Path} [lt_{kl} + (mt_{jw} + lt_{mn}) x_{jw}] x_j\}\} \quad (3)$$

The first term in the above equation represents the contribution of the hardware cores to the execution time of the critical path. Similarly, the second term captures the contribution from the tasks executed as software modules. Finally, the third term is the contribution to the processing time of the delay due to direct links between different architecture cores and possibly of the memory access time if the application communication channel j is mapped onto a memory core. Here, mt_{jw} is the memory access time with $w \in \{1, .., M\}$, lt_{kl} is the link delay between architecture cores k and l with $k, l \in \{1, .., |V_{AR}|\}$ and lt_{mn} is the link delay between architecture cores m and n also with $m, n \in \{1, .., |V_{AR}|\}$.

The variables $x_{iu},\ x_{iv},\ x_{jw}$, and x_j are *decision variables* that capture whether a task i is mapped to a hardware core u or a software core v, whether a communication channel j is mapped to a memory core w, and whether a communication channel is contained within a core (i.e., two communicating tasks are mapped to the same core, in which case $x_j = 0$) or not. The values of these decision variables are different for different mapping solutions, which are generated during the genetic algorithm based design space exploration from Fig. 3.

D.3 Objective 3: Energy Consumption

The third objective function minimizes the energy consumption.

$$\min\left\{\sum_{i\in V_{AR}} t_p^e w_{pe} + \sum_{j\in E_{AR}} (t_l^c w_{le} + t_m w_{me})\right\} \quad (4)$$

In the above equation, t_p^e is the time spent by the processing cores for execution, t_l^c is the time spent on communication, and t_m is the total processing time of the memory cores.

D.4 Solving the Multi-Objective Problem

Once all three objective functions are defined as discussed in the previous sections, the overall optimization problem - which in our case is the mapping problem - can be written in a generalized form as follows:[20]

$$\underset{\mathbf{x}}{min}\mathbf{z} = \mathbf{f}(\mathbf{x}) = (\mathbf{f}_1(\mathbf{x}), \mathbf{f}_2(\mathbf{x}), \mathbf{f}_3(\mathbf{x}))^{\mathbf{T}} \\ \text{s.t.}\, \mathbf{x} \in \mathbf{X} \quad (5)(6)$$

In the above equation, x represents a particular solution, and X is a set of feasible solutions. In our case, a mapping solution is captured by the individual decision variables discussed earlier that completely describe how application tasks are assigned to the cores of the architecture platform. The three individual objective functions f_1, f_2, and f_3 effectively evaluate the expressions from equations (2), (3), and (4) for a given mapping solution. The overall objective function **z** = **f**(**x**) translates a solution x from the *decision space* defined by the decision variables to a point in the *objective space* defined by the three objective or cost functions. In our case, the objective space is three dimensional and the overall objective function is defined as the equally weighted summation of the three individual objective functions from equations (2), (3), and (4).

Because multi-objective optimization problems usually do not have a single best solution which optimizes all objectives at the same time, we are interested in finding a set of solutions that form the so called *Pareto frontier*. The solution points that form the Pareto frontier are points that are *non-dominated* by any other solution point among all solutions from the feasible set. To solve the multi-objective mapping problem and to generate the Pareto frontier, we use evolutionary algorithms due to their ability to handle multiple objectives at the same time. More specifically, we use the NSGA-II[21] because it was shown to offer benefits over other types of evolutionary algorithms including ease of implementation and lower computational complexity.[20] The pseudocode description of this algorithm is shown in Algorithm **1.**This algorithm implements the outer loop of the method described in Fig. 3.

Algorithm 1: Design space exploration based on NSGA-II

```
Input: $N$ population size, $M$ max number of generations
Output: Pareto frontier, non-dominated solutions in $P_M$

  $P_0$ = GenerateInitialPopulation(); // size $N$
  $Q_0 = \emptyset$; // start with children set empty
  EvaluateObjectiveFunction($P_0$); // calculate fitness
  RankPopulation($P_0$); // done according to fitness values
  for ($i = 0$ to $M - 1$) do
    // Create children population:
    $Q_i$ = SelectionCrossoverMutation($P_i$);
    // Uncertainty aware, Monte Carlo based:
    EvaluateObjectiveFunction($Q_i$);
    $P_{i+1}$ = CombineParentsAndChildren($P_i$, $Q_i$);
    RankPopulation($P_{i+1}$);
    // Elitism: keep non-dominated:
    $P_{i+1}$ = SelectNIndividuals($P_{i+1}$);
  end for
```

The genetic algorithm iteratively generates new children solution populations from previous parent solution populations using crossover and mutation. This generation is usually realized using different forms of crossover and mutation. In the beginning, the genetic algorithm requires an initial set of solutions, the initial population, which in our implementation we generate randomly for simplicity. The way mapping solutions are encoded is similar to the approach discussed in.[20] Each genotype (or representation of the possible mapping) consists of task nodes that can mapped to SW components, task nodes that can be mapped to HW components, and communication arcs that can be mapped to memory components. Each gene in the chromosome (or genotype) has its own feasible set. For example, for genes representing nodes that must be mapped to SW components, only the set of CPUs in the architecture model form the feasible set. During the iterative process of the outer loop in Fig. 3, new solutions are evaluated by *EvaluateObjectiveFunction(*), which uses equation (5). It is also this evaluation step that distinguishes our approach from previous work. Here, we assume uncertainties to affect design parameters. The evaluation step employs a Monte Carlo simulation technique to deal with uncertain quantities and is discussed in the next section. For more details about the NSGA-II algorithm, please see.[21]

E. Estimation Under Uncertainty

During the NSGA-II genetic algorithm based DSE depicted in Fig. 3, each new solution candidate must be evaluated in order to estimate: reliability, execution time, and energy. Their deterministic calculation, in a traditional design flow, can be done using the main expressions from equations (2), (3), and (4). However, when design parameters are affected by uncertainties, any of the design attributes that is affected by uncertainties cannot be estimated anymore using deterministic equations. Analytic solutions are extremely difficult or impossible to derive when dealing with a wide variety of distributions. Instead, estimation techniques that model and can handle uncertainty must be employed. In such situations, a Monte Carlo simulation based technique represents the only effective technique that is capable of accommodating multiple types of probability distributions.[13,14] This technique is represented by the *Monte Carlo Simulation* block in Fig. 3 and is described in more details in Fig. 5.

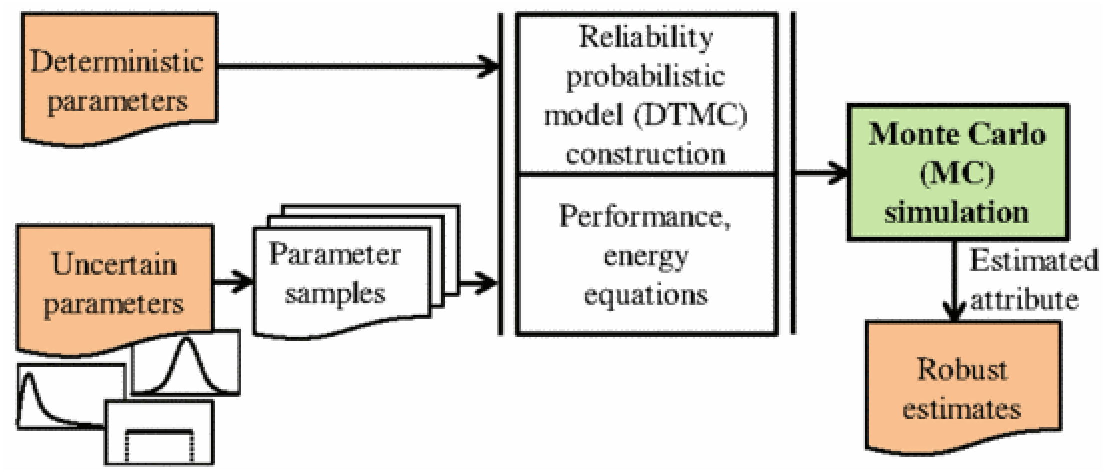
Deterministic parameters
Reliability probabilistic model (DTMC) construction
Performance, energy equations
Monte Carlo (MC) simulation
Uncertain parameters
Parameter samples
Estimated attribute
Robust estimates

Fig. 5. Block diagram of the monte carlo simulation based technique to estimate reliability, execution time, and energy.

More specifically, to estimate *reliability* we employ the enhanced Monte Carlo estimation technique proposed in.[14] In this case, the input to the *Monte Carlo Simulation* block from Fig. 5 is the probabilistic DTMC reliability model, which includes parameters affected by uncertainty and specified as probability distributions. During the MC iterations, these distributions are sampled to generate instances that are then used as numerical values to compute the attribute of interest. In this way, the impact of uncertainties on the estimation process is captured. The estimated reliability metric becomes a variable quantity itself whose distribution is unknown. The variation of this quantity will represent an important measure that summarizes the impact of uncertainties.

To estimate the *performance* and *energy consumption* attributes, the Monte Carlo simulation technique is simpler because here we do not need to build the probabilistic DTMC model. During multiple MC runs, parameters affected by uncertainties are also sampled from their respective probability distributions and used as numerical values inside equations (3) and (4).

F. Robustness of the Design Solution Points

The output of the MC simulation technique to estimate a certain attribute of interest for a given mapping solution is a number of samples out of the probability distribution that characterizes the unknown attribute. We use the 95 percentile estimate as the actual value used to generate and plot the robust Pareto frontier in the objective space. Working with percentile estimates provides a means to quantify or specify the robustness of the solution. The higher the percentile, the more robust the given solution is against uncertainties. Aside from generating the robust Pareto frontier in the three dimensional objective space, during each of the genetic algorithm iterations (see Fig. 3), solution points that are found to be better than previously found solutions are selected and added to the list of *best robust solutions*. This is a short list of potential design solution points from which embedded systems designers can select a final solution.

Simulation Results

The proposed design method was implemented as a *C++*computer program, which also integrates the publicly available implementation of NSGA-II.[22] For simulations, we use four testcases. The first two testcases are from the automotive application domain, ABS (anti-lock break system) and *ACC*(adaptive cruise control). We adopted these two testcases from the study in.[14] The last two testcases are from the multimedia application domain and include H.264 (video decoder)[23] and JPEG (picture compression).[24]

However, due to lack of space we report results for the first testcase only. The other testcases have similar plots and the conclusions that that we arrived at are the same for each of these testcases. The only difference between these testcases is the computational runtime, which increases linearly with the testcase size.

A. Architecture Platform

Because reliability, performance, and energy consumption represent objective functions, the only constraints that we used in our problem formulation consist of the architecture platform being given and the HW/SW partitioning of the given application. Specifically, in our case we assume that the architecture platform has twelve components in order to be able to accommodate the largest application task graph that we investigated. That includes five software components, five hardware components, and two memory components. The communication arcs in the graph are assumed to be implemented via memory mapping; that is, the source task writes into a memory component and the destination tasks read from the memory component. In our assumed architecture platform (see Fig. 2), we assume two types of CPUs similar to the recent multicore proposals that integrate high-performance “big” and energy efficient “little” cores.[25–26,27] As HW components, we assume also two different types of FPGAs; one type that is slower but consumes less power and the other type that is faster but consumes more power. The FPGAs are assumed to be faster than the CPUs because they can offer increased parallelism; they may not be as fast as ASIC cores, but, have the flexibility of reconfiguration. We adopt execution times and failure rates similarly to the study in[14] and power consumption values similar to those reported in.[28]

B. Pareto Frontiers

In the first set of simulations, we use our tool to identify the robust Pareto frontier in the (*1-reliability) vs. performance vs. energy* objective space. All attributes are assumed to be affected by uncertainties and therefore, they are estimated using the Monte Carlo technique described in section 3.E. In order to generate Pareto frontiers that are scale independent, we normalize the performance and energy cost functions such that all values are inside the range [0, 1]. The normalization of a given cost function is done according to: $f_{norm} = (f - f_{min})/(f_{max} - f_{min})$, where f_{min} and f_{max} are the minimum and maximum or worst case scenario values of the respective objective cost function *f*. The cost function (*1-reliability*) is already with values in the [0, 1] range, hence, it does not require normalization.

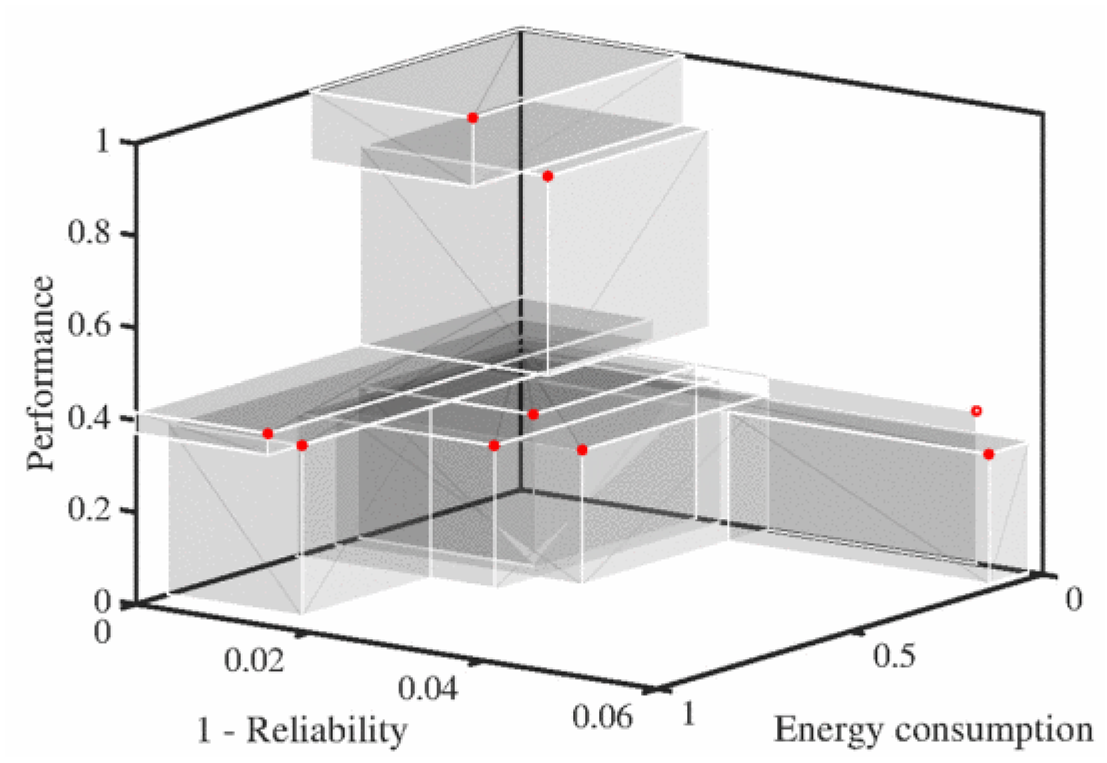

Fig. 6. Robust pareto frontier of the ABS testcase for 5% injected uncertainty.

The simplified Pareto frontier for the *ABS* testcase is shown in Fig. 6, for a level of 5% injected uncertainty. It is simplified in the sense that it does not show all the actual solution points that were found to be on the frontier during the execution of the tool. During this simplification, we basically select nine solution points: three solution points that are as close as possible to the center of coordinates, and three pairs of two solution points that are very good in terms of only one of the three costs. We do this in order to keep this figure simple, yet to give the user enough solutions to choose from (the number of nine can be changed to a different number if the user desired).

The solution points that are the closest to the system of coordinates represent solutions that the tool reports as being the best compromise among all three objectives. The other solution points can be selected if any of the three objectives is very important, depending on the application at hand. For example, if execution time or performance is highly critical, one of the two solution points that were found to offer very good performance (but with worse energy consumption and worse reliability) can be selected. The ability to generate these 3D Pareto frontiers comprised of robust solution points (robust in the sense described in section 3.F) represents one of the main contributions of this paper.

C. Different Levels of Uncertainty

Second, we investigate how the Pareto frontiers change for different levels of injected uncertainty. Being able to study different levels of uncertainty can help in scenarios where we want to conduct *what if* type of investigations.

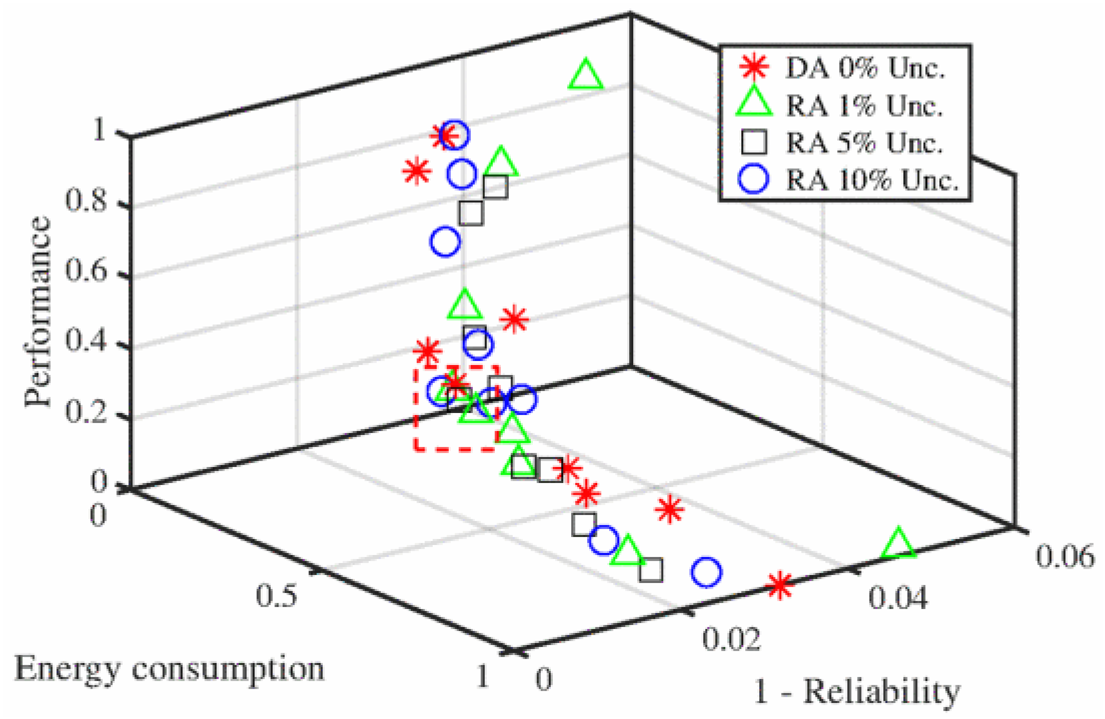

Fig. 7. Pareto frontier of the ABS testcase for different levels of injected uncertainty: 0% (deterministic approach, da), 1%, 5%, and 10% (robust approach, ra).

For example, let us say that for a given technology node the uncertainty level is assumed to be 5%, but, that this value in not completely certain. In this case, we could investigate how the selected best solution found by the tool for uncertainty 5% would change if the assumed uncertainty level itself is varied. Such an investigation can help to see how the solution point moves in the 3D space and whether it still satisfies the desired figures of merit for the application at hand. This scenario is what we focus in this section. The different levels of uncertainty that we simulated are: 0% (no uncertainty, this is the deterministic case), 1%, 5%, and 10%. The Pareto frontier for these levels of uncertainty is shown in Fig. 7.

Having the deterministic case as a reference, when uncertainty is injected, the previously deterministic and fixed parameter values are replaced with samples generated out of various probability distributions, each characterized by a certain mean and standard deviation pair. The standard deviation value that is used is directly related to the amount of desired uncertainty to be injected as discussed earlier in the paper. Thus, a previously deterministic design solution point degenerates into a probability distribution, whose 95 percentile estimate represents the robust solution point that we use for constructing the robust Pareto frontier. The location of this point is most likely different than the location of the previously deterministic design solution point. The amount of this change is within a vicinity whose size is dictated by the amount of uncertainty injected.

For example, this can be seen in the zoom-in picture from Fig. 8, which shows the four solution points for each of the four levels of injected uncertainty for a given mapping solution. The zero injected uncertainty represents the deterministic approach. This figure illustrates how a solution point found by traditional deterministic approaches can be off from the robust design solution point identified by our tool for a given level of injected uncertainty. However, by using our tool, we can identify this shift and quantify each of the found solution points in terms of reliability, performance, and energy per assumed amount of uncertainty. Therefore, such a tool can aid embedded designers in finding the appropriate solution points to be selected for a given application domain. Our tool provides the means to investigate these changes.

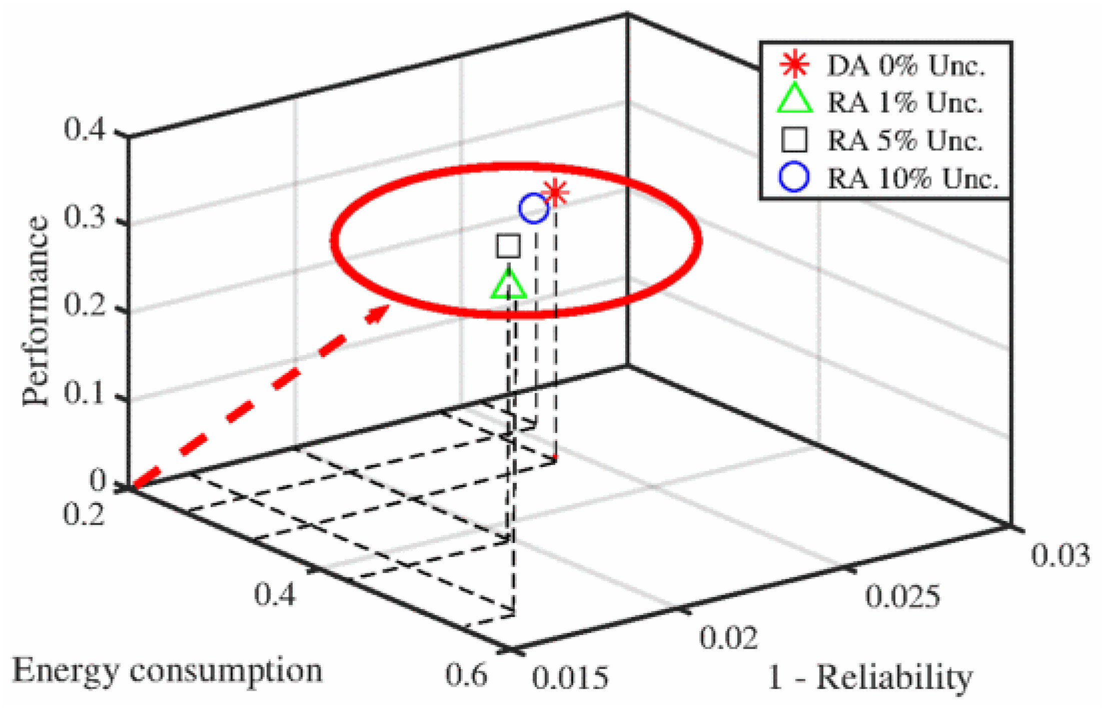

Fig. 8. Zoom-in of fig. 7.

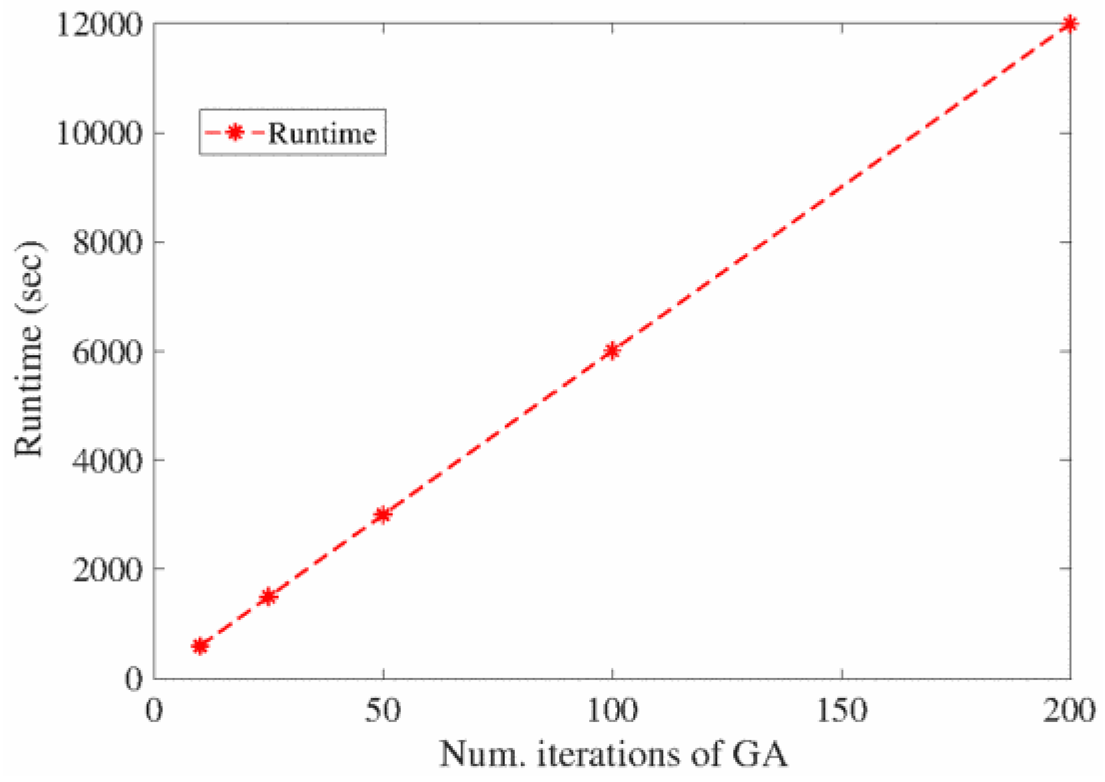

Fig. 9. Computational runtime of our tool versus the number of iterations of the NSGA-II genetic algorithm.

D. Computational Complexity and Convergence

The computational complexity of the proposed tool is primarily affected by two factors, for a given testcase size. These factors are the number of iterations of the outer and inner loops from Fig. 3. To study the scalability of the computational runtime with the number of iterations of the outer loop, which corresponds to different number of solution populations explored by the genetic algorithm, we plot in Fig. 9 the computational runtime of our tool versus the number of iterations of the outer loop. Each iteration of the outer loop includes 2000 iterations of the Monte Carlo algorithm; this number was found to provide satisfactory convergence. The plot in Fig. 9 shows that the computational runtime scales linearly.

In addition, we are interested in finding out what is the minimum number of MC runs after which convergence in the process of estimation is achieved. To answer this question, we looked at how the number of MC runs impacted the convergence of the estimation of the objective cost functions. This is illustrated by the plot in Fig. 10, where we can see that after about 2000 iterations of the MC algorithm, the estimated value of reliability converges to a stable value. Similar results were obtained during the estimation of performance and energy consumption.

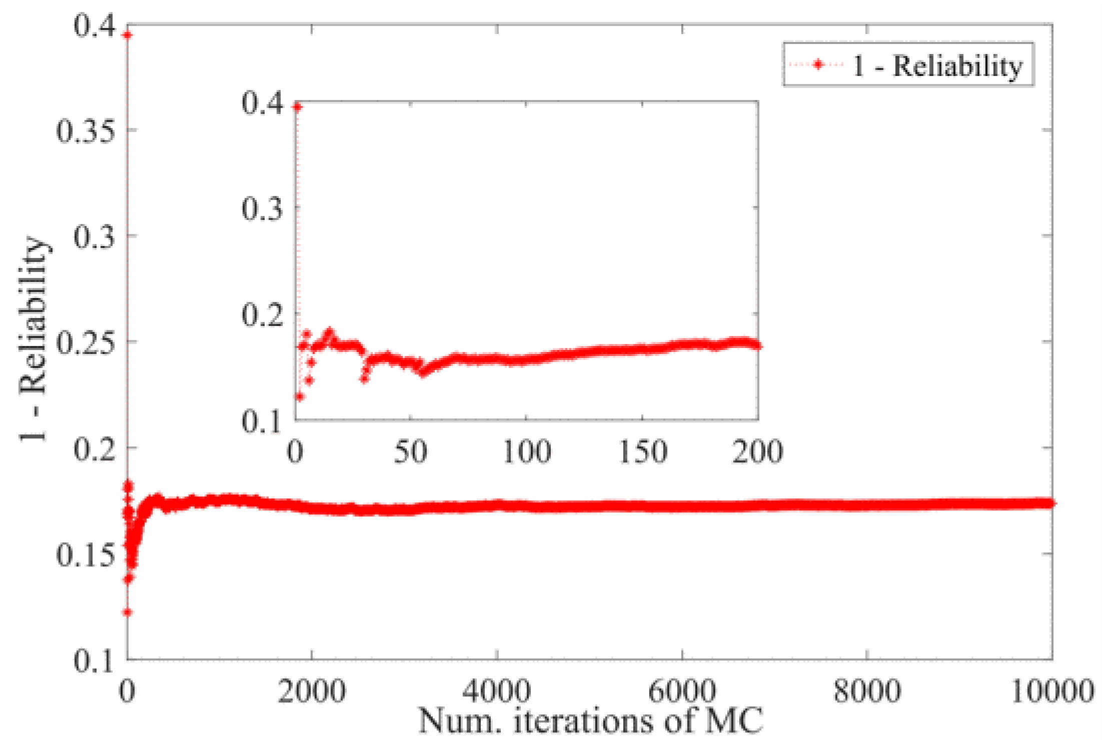

Fig. 10. Illustration of the convergence of the MC simulation based estimation.

Conclusion

We presented a design methodology for the design of embedded systems under uncertainties. The proposed methodology integrates uncertainty models and optimization algorithms constructed with Monte Carlo and evolutionary algorithms and is capable of finding the robust Pareto frontiers in the objective space for a given test-case application, architecture platform, and given levels of injected uncertainties. Simulation results demonstrated the effectiveness of the proposed design method. In future work, we plan to also include scheduling into our problem formulation and to investigate architecture models that use networks-on-chip for communication. Architecture platform synthesis with direct consideration of all three objectives is also an interesting problem to investigate.

Acknowledgement

This work was supported by the National Science Foundation (NSF), grant CCF-1524909. Any findings and conclusions or recommendations expressed herein are those of the authors and do not necessarily reflect the views of the NSF.

References

1. M. Lombardi, M. Milano, L. Benini, "Robust scheduling of task graphs under execution time uncertainty", *IEEE Tran. on Comp.*, 2013.

2. A. DeHon, H.M. Quinn, N.P. Carter, "Vision for cross-layer optimization to address the dual challenges of energy and reliability", *ACM/IEEE DATE*, 2010.

3. S. Borkar, " Thousand core c hips: a technology perspective ", *ACM/IEEE DAC*, 2007.

4. K. Sigdel, C. Galuzzi, K. Bertels, M. Thompson, A.D. Pimentel, "Evaluation of runtime task mapping using the Sesame framework", *Int. J. Reconfig. Comp.*, 2012.

5. S.-H. Kangz, H. Yang, L. Schor, I. Bacivarov, S. Ha, L. Thiele, "Multi-objective mapping optimization via problem decomposition for many-core systems", *ESTImedia*, 2012.

6. G. Ascia, V. Catania, M. Palesi, "A GA-based design space exploration framework for parameterized system-on-a-chip platforms", *IEEE Trans. on Evolutionary Computation*, 2004.

7. F. Balarin et al., "Metropolis: an integrated electronic system design environment", *IEEE Computer*, 2003.

8. A. Cassidy, J. Paul, D. Thomas, "Layered multi-threaded high-level performance design", *ACM/IEEE DATE*, 2003.

9. R. Domer et al., "System-on-chip environment: a SpecC-based framework for heterogeneous MPSoC design", *EURASIP JES*, 2008.

10. H. Nikolov, T. Stefanov, E. Deprettere, "Systematic and automated multiprocessor system design programming and implementation", *IEEE TCAD*, 2008.

11. P. Meloni et al., "System adaptivity and fault-tolerance in NoC-based MPSoCs: the MADNESS project approach", *EUROMI-CRO Conf. on Digital System Design Architectures Methods and Tools*, 2012.

12. C. Bolchini, M. Carminati, A. Miele, A. Das, A. Kumar, B. Veeravalli, "Run-time mapping for reliable many-cores based on energy/performance trade-offs", *IEEE DFT*, 2013.

13. A.Y. Yamamoto, C. Ababei, "Unified reliability estimation and management of NoC based chip multiprocessors", *Microprocessors and Microsystems*, 2014.

14. I. Meedeniya, A. Aleti, L. Grunske, "Architecture-driven reliability optimization with uncertain model parameters", *J. of Systems and Software*, 2012.

15. F. Khosravi, M. Muller, M. Glaß, J. Teich, "Uncertainty-aware reliability analysis and optimization", *ACM/IEEE DATE*, 2015.

16. Y. Li, J. Chen, L. Feng, "Dealing with uncertainty: a survey of theories and practices", *IEEE Trans. on Knowledge and Data Engineering*, vol. 25, no. 11, pp. 2463-2482, Nov. 2013.

17. A.D. Pimentel, C. Erbas, S. Polstra, "A systematic approach to exploring embedded system architectures at multiple abstraction levels", *IEEE Trans. on Comp.*, 2006.

18. R.C. Cheung, "A user-oriented software reliability model", *IEEE Transactions on Software Engineering*, 1980.

19. Indika Meedeniya, *Architecture Optimisation of Embedded Systems under Uncertainty in Probabilistic Reliability Evaluation Model Parameters*, 2012.

20. Cagkan Erbas, *System-Level Modeling and Design Space Exploration for Multiprocessor Embedded System-on-Chip Architectures*, 2006.

21. K. Deb, S. Agrawal, A. Pratap, T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II", *IEEE Trans. on Evolutionary Computation*, 2002.

22. K. Deb, *Multi-objective NSGA-II code in C*, 2016, [online] Available: http://www.egr.msu.edu/....kdeb/codes.shtml.
23. J.R. Smith, "The H.264 video coding standard", *IEEE Computer Society*, 2006.
24. G. Wallace, "The JPEG still picture compression standard", *IEEE Trans. on Consumer Electronics*, 1992.
25. *ARM big.LITTLE technology*, 2017, [online] Available: https://developer.arm.com/technologies/big-little.
26. N. Chitlur, G. Srinivasa, S. Hahn, P.K. Gupta, D. Reddy, D. Koufaty, P. Brett, A. Prabhakaran, L. Zhao, N. Ijih, S. Sub-haschandra, S. Grover, X. Jiang, R. Iyer, "QuickIA: exploring heterogeneous architectures on real prototypes", *IEEE HPCA*, 2012.
27. X. Liang, M. Nguyen, H. Che, "Wimpy or brawny cores: a throughput perspective", *J. of Parallel and Distributed Computing Archive*, 2013.
28. V. Konstantakos, A. Chatzigeorgious, S. Nikolaidis, T. Laopoulos, "Energy consumption estimation in embedded systems", *IEEE Trans. on Instrumentation and Measurement*, 2008.