# Building Faithful High-level Models and Performance Evaluation of Manycore Embedded Systems

Ayoub Nouri, Marius Bozga, Anca Molnos, Axel Legay, Saddek Bensalem

# Building Faithful High-level Models and Performance Evaluation of Manycore Embedded Systems Technical Report

Ayoub Nouri*, Marius Bozga*, Anca Molnos†, Axel Legay‡, and Saddek Bensalem*
*Univ. Grenoble Alpes, VERIMAG, F-38000 Grenoble, France
CNRS, VERIMAG, F-38000 Grenoble, France
†CEA/LETI, Grenoble, France
‡INRIA/IRISA, Rennes, France

***Abstract*—Performance and functional correctness are key for successful design of modern embedded systems. Both aspects must be considered early in the design process to enable founded decision making towards final implementation. Nonetheless, building abstract system-level models that faithfully capture performance information along to functional behavior is a challenging task. In contrast to functional aspects, performance details are rarely available during early design phases and no clear method is known to characterize them. Moreover, once such system-level models are built they are inherently complex as they usually mix software models, hardware architecture constraints and environment abstractions. Their analysis by using traditional performance evaluation methods is reaching the limits and the need for more scalable and accurate techniques is becoming urgent. In this paper, we introduce a systematic method for building stochastic abstract performance models using statistical inference and model calibration and we propose statistical model checking as performance evaluation technique upon the obtained models. We experimented our method on a real-life case study. We were able to verify different timing properties.**

## I. Introduction

The increasing complexity of modern embedded systems, together with the growing competition and the time-to-market constraints, has forced designers to consider more elaborate and systematic approaches to system design. *System-level design* [1] has emerged as a new methodology to address these challenges. Among the key concepts in system-level design is *high-level modeling*, that is, capturing the system functionality at a high-level of abstraction. A high-level model can be obtained with less effort and enables fast design space exploration which is of paramount importance for early and founded design decisions making towards the final implementation.

For many years, functional aspects where in the center of the high-level modeling process, while extra-functional ones where considered as a second-class citizen. In modern systems such as wearable, where a limited amount of resources is available, extra-functional aspects are becoming equally important and not considering them may lead to dramatic results later in the design process. For instance, a smart-phone that quickly losses energy is not going to sell even if it provides all required services. This points out two additional important design requirements. First, building high-level models that captures extra-functional aspects, especially performance, is a must for a successful design. Second, given the importance of such aspects, traditional performance evaluation techniques, decorrelated from the functional aspects, are no more sufficient. Hence, the need for rigorous *system-level verification* techniques.

Performance aspects are related to the physical part of the system, that is, the execution of the application functions on specific architecture components, e.g., execution time of a function by a processing unit, communication delay of a bus or a Network on Chip (NoC), amount of consumed energy or dissipated temperature. Nonetheless, such details are rarely available in early design phases, which makes the process of building high-level performance models quite challenging.

*Contradictory goals, Abstract Vs. Faithful:* In one hand, one wants to deal with abstract models that minimize modeling effort and exploration time. In the other hand, these models are required to capture low-level performance details in order to precisely reflect the reality and enable accurate reasoning about the whole system performance. This raises several natural questions: How to capture performance information in early design phases? What kind of formalism is appropriate to characterize them? And, how to integrate them in the abstract system model?

In this paper, as a first contribution, we propose a technique based upon statistical approaches for characterizing low-level performance details. These are extracted from automatically generated and instrumented implementation. The high-level models are then calibrated using these performance details as to obtain more faithful representations.

Characterizing low-level performance details statistically is motivated by their significant variability which cannot be captured by point estimates. The variability in performance data is mainly due to two reasons. First, the inputs (the workload) are generally variable albeit some systems are data-independent. The second reason is the inherent hardware components behavior, e.g., caches, interference, memory contention, etc. These cannot be modeled in details in early

design phases because of the lack of detailed specification and the required abstraction level.

Our second contribution concerns system-level verification. Traditionally, once performance models are built, pure simulation or analytical approaches are used for analysis. Our proposal is a trade-off between these two techniques. It consists of Statistical Model Checking (SMC) [2], [3] which combines simulation and statistical techniques. Moreover, it provides quantitative evaluation of the requirements which is more appropriate for performance evaluation. To the best of our knowledge, this is the first time SMC is being used to performance evaluation of manycore embedded systems.

We experimented this approach as part of the BIP design-flow for rigorous systems design [4]. In the BIP flow, all the design phases are driven by a single component-based semantics [5], [6]. Our contributions are used within the whole flow as a method for design space exploration. We built tool-support for most of the method parts, i.e., automatic code generation, statistical model checking, statistical inference and validated it on a real-life case study namely the HMAX Models algorithm [7] for image recognition. We considered the STHORM platform [8] as a target architecture and we were able to verify a bench of timing requirements.

*Organization:* Section II introduces the BIP formalism and its stochastic extension in addition to the Statistical Model Checking technique. In Section III, we detail our method for building and analyzing high-level performance models. Section IV presents a concrete application of our method on a real-life case study. The end tail portion of the paper depicts related work and conclusions.

## II. Background

In this section, we introduce the BIP formalism and its stochastic extension. We also briefly describe the Statistical Model Checking technique.

### A. BIP and Stochastic BIP Models

BIP (Behavior-Interaction-Priority) [5] is a formal framework for building complex systems by coordinating the behavior of a set of atomic components. Behavior is defined as a transition system extended with data and functions described in C/C++. The description of coordination between components is layered. The first layer describes the interactions between components. The second layer describes dynamic priorities between interactions and is used to express scheduling policies. BIP has a clean operational semantics that describes the behavior of a composite component as the composition of the behaviors of its atomic ones. This allows a direct relation between the underlying semantic model (transition systems) and its implementation.

In BIP, atomic components are finite-state automata extended with variables and ports. Variables are used to store local data. Ports are action names, and may be associated with variables. They are used for interaction with other components. States denote control locations at which the components await for interaction. A transition is a step, labeled by a port, from a control location to another. It has associated a guard and an action, that are respectively a Boolean condition and a computation defined on local variables. Connectors relate ports from different sub-components. They represent sets of interactions, that are, non-empty sets of ports that have to be jointly executed. For every such interaction, the connector provides the guard and the data transfer, that are, respectively, an enabling condition and an exchange of data across the ports involved in the interaction. Finally, priorities provide a means to coordinate the execution of interactions within a BIP system. They are used to specify scheduling policies between simultaneously enabled interactions. More concretely, priorities are rules, each consisting of an ordered pair of interactions associated with a condition. When the condition holds and both interactions of the corresponding pair are enabled, only the one with the highest priority can be executed.

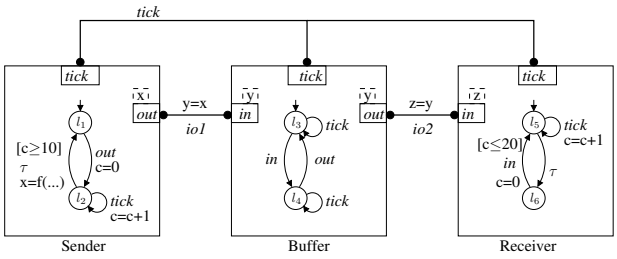


Figure 1: BIP example: Sender-Buffer-Receiver system.

Figure 1 shows a graphical representation of an example model in BIP. It consists of three atomic components, namely *Sender*, *Buffer* and *Receiver* for which the behavior is described as transition systems. For instance the *Sender* has two control locations $l_1$ and $l_2$ and communicates through ports *tick* and *out* associated with the variable *x*. The components are connected using *io1, io2*, and *tick* connectors. These enforces strong synchronization, that is, executing the involved ports in parallel. For example, the *io1* connector represents an interaction with data transfer from the *out* port of the *Sender* to the *in* port of the *Buffer*. As a result, the value of *x* is assigned to *y* in the *Buffer*.

The stochastic extension of BIP [6] allows (1) to specify stochastic aspects of individual components and (2) to provide a purely stochastic semantics for the parallel composition of components through interactions and priorities. Syntactically, stochastic behavior at the level of BIP atomic components is obtained by using probabilistic variables $x^P$. These are attached to probability distributions $\mu_{x^P}$ (implemented as C functions) and are updated during transition firing where they get random values accordingly. The semantics on transitions is thus fully stochastic. The stochastic semantics also covers the interaction level. When several interactions are enabled after application of priority

rules, a probabilistic choice among them is performed using a user-specified probability distribution.

### B. Statistical Model Checking

The previous section introduced our high-level formalism for stochastic systems. Now, we focus on verifying bounded temporal properties over them. We will be mostly interested in checking if the probability that the execution time $t$ of the system always stay within a bound $\Delta$ is greater than some value, say $\theta$. As we will rely on simulation, we will only monitor the above property on a finite horizon $l$. In this context, the problem can be denoted in Linear Temporal Logic (LTL) [9] as $\varphi = P_{\geq\theta}[\Box^l(t < \Delta)]$. There are several ways to check such properties.

A first solution to the above mentioned problem is to rely on probabilistic model checking whose main purpose is to compute the exact probability $p$ to satisfy $\varphi$. In our context, we are only interested in knowing whether $p$ is greater or equal to $\theta$. To solve this qualitative problem, we will rely on a sequential hypothesis testing engine.

The idea is to monitor $\varphi$ on several finite executions $\rho$ (of size $l$) of the system. If the actual measure of paths satisfying $\varphi$ is $p$ then this is a Bernoulli experiment with parameter $p$ — with probability $p$ we will conclude the simulation by observing that all extensions of $\rho$ satisfy $\varphi$ and with probability $1 - p$ we will conclude the simulation by observing that all extensions of $\rho$ do not satisfy $\varphi$. The goal is to determine if the unknown parameter $p$ is at least $\theta$, by conducting this experiment multiple times *independently* [1].

Checking if the property holds is determined as follows. Let $B_i$ be the random variable that takes value 1 if the $i$th sample satisfies $\varphi$ and 0 otherwise. Finally let $b_i$ be its realization in the specific sample. Consider two hypotheses — $H_0$, which is the hypothesis that $p \geq p_0 > \theta$, and $H_1$, which is the hypothesis that $p \leq p_1 < \theta$. Hypothesis $H_0$ is accepted if $X = \sum_i b_i$ is greater than a threshold $c$, and hypothesis $H_1$ is accepted otherwise. There are two types of errors associated with such a statistical test: Type I error measures the probability of accepting $H_0$ when $H_1$ actually holds and Type II error measures the probability of accepting $H_1$ when $H_0$ holds. Typically, we denote the bound on the Type I error by $\alpha$, and the bound on Type II error by $\beta$. Ideally, we would like $p_0 = \theta = p_1$, but it is too difficult to bound both values simultaneously. Therefore, we choose $\delta$ such that $p_0 = \theta + \delta$ and $p_1 = \theta - \delta$. The interval $[p_1, p_0]$ is referred to as the *indifference region* of the test.

In the sequential probability ratio test, one has to choose two values $A$ and $B$, with $A > B$. These two values should be chosen to ensure that the strength of the test is respected. Let $m$ be the number of observations that have been made so far. The test is based on the following quotient:

$$\frac{p_{1m}}{p_{0m}} = \prod_{i=1}^{m} \frac{Pr(B_i = b_i \mid p = p_1)}{Pr(B_i = b_i \mid p = p_0)} = \frac{p_1^{d_m}(1-p_1)^{m-d_m}}{p_0^{d_m}(1-p_0)^{m-d_m}}, \tag{1}$$

where $d_m = \sum_{i=1}^{m} b_i$. The idea behind the test is to accept $H_0$ if $\frac{p_{1m}}{p_{0m}} \geq A$, and $H_1$ if $\frac{p_{1m}}{p_{0m}} \leq B$. An algorithm for sequential ratio testing consists of computing $\frac{p_{1m}}{p_{0m}}$ for successive values of $m$ until either $H_0$ or $H_1$ is satisfied. In his thesis [3], Younes proposed the SPRT algorithm (Algorithm 2.3 page 27) that given $p_0, p_1, \alpha$ and $\beta$ implements the sequential ratio testing procedure. We should observe that computing ideal values $A_{id}$ and $B_{id}$ for $A$ and $B$ in order to make sure that we are working with a test of strength $(\alpha, \beta)$ is a laborious procedure (see Section 3.4 of [10]).

## III. Methodology

This section introduces our approach for high-level modeling and analysis of performance in the context of system-level design. A general overview of the method is first presented followed by a detailed explanation of its steps.

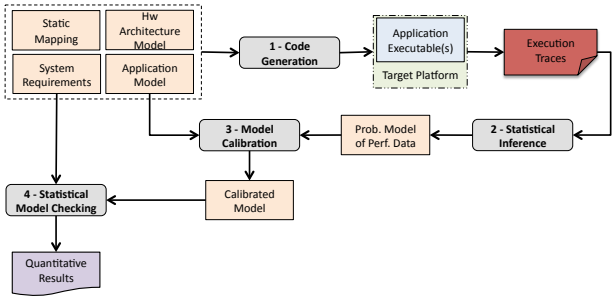


Figure 2: Overview of the proposed method.

The proposed approach perfectly integrates Y-chart-based flows [11], which consider separate models for the application and the hardware architecture. As shown in Figure 2, it takes as inputs, the application and the architecture models, a static mapping of the application to the architecture, and a set of requirements. We rely on the BIP formalism as a computation model to capture the application and the architecture behaviors. These may be obtained automatically through refinement from higher level specifications [4] or provided directly by the designer.

The method consists of generating an implementation of the application on the target architecture (1). The latter could be, depending on the design phase, an already existing board, a virtual prototype, or an Instruction-Set Simulator (ISS). The code generation step produces instrumented code with respect to the input requirements. This specifies the performance aspects to estimate. After model execution/simulation, traces are obtained and statistically analyzed to infer probabilistic characterization of the performance

[1] Ensuring independent sampling has its own challenges which will not be addressed in this paper.

data (2). These are used to calibrate the BIP application model (3). Finally, quantitative analysis of the input requirement is performed using Statistical Model Checking (4).

### A. Generating Implementations

The BIP framework encompasses different code generation back-ends for centralized and distributed targets [12]. In the context of manycore architectures, we mainly consider the application model and the mapping, in addition to the performance requirements to instrument the generated implementation accordingly. There are basically two steps in order to produce implementations from BIP models:

*Computation/Communication Objects Generation:* In this phase, each computation BIP component is systematically transformed to a process. The notion of process is used in an abstract meaning. Its concrete interpretation depends on the target runtime, e.g., POSIX threads. The behavior of each generated process consists of the corresponding BIP component automaton where synchronizations are transformed to communication primitives calls provided by the target runtime. Communication objects are usually shared memory objects, e.g., fifo channels. This are similarly generated given the target runtime.

*Deployment/Glue Code Generation:* This mainly produces code that maps the generated objects on the target hardware architecture. That is, the processes to specific processing units and the communication objects to memory.

Remark that the code generation process depends on the targeted runtime. In this section, we described the general shape of this process. Additional details are presented in Section IV. In this work, we consider STHROM [8] as a target architecture and an implementation of the MCAPI[2] standard as the underlying runtime. We implemented a new BIP back-end code generator targeting this architecture.

### B. Characterizing Performance Data

We propose a statistical method to characterize performance data coming from concrete execution or low-level simulation of functional models. The idea is to fit a good probabilistic model to the obtained data [13]. This could be a probability distribution (Normal, Exponential, etc.) or a more sophisticated model such as combination of distributions or a Markov model. In this paper, we first consider probability distributions as potential model and use *Distribution Fitting* [13], [14] as to probabilistically characterize the data. Given execution traces (a set of observations of the performance metric), *Distribution Fitting* allows to statistically learn the best distribution that fits the data.

From this perspective, data is assumed to be generated by a stochastic process for which the governing law is unknown. Our goal is to infer such a law from a subset of observations, called a sample, since the whole population is generally not available. Formally, given $x_1, ..., x_n$ a set of observations, there exists $X_1, ..., X_n$ independent and identically distributed (iid) random variables such that $x_i$ is a possible realization of $X_i$. Independence is to be understood in the sense that the outcome of a random variable does not affect the outcome of another. Identically distributed random variables basically means that they came from the same probability distribution $D(\omega)$ where $\omega \in \Theta$ is the set of parameters of the distribution defined over the space $\Theta$.

One should pay attention to the independence assumption above since this will enable accurate generalizations of the inference results. Note that the goal is not only to characterize the available data. The most important is to be able to generalize the result to the generating process. That is, to conclude that the generating process follows the learned distribution. Concretely, the Independence assumption states that the observations are made independently. Two possible configurations are generally possible. The first is when an experiment is conceived with the aim to observe a specific phenomenon. In such a case, independence is easy to guarantee since the procedure is completely controlled. The second case is when we perform observations on a process which is not under our control (or partially controlled), e.g., simulation or execution of a system. In this case, independence cannot be assumed but must be checked. Several ways exists to check independence although not always easy to understand. One can use, for example, specific plots which require expertise for interpretation or rely on existing statistical tests such as Box-Pierce [15], Ljung-Box [16], and runs test[3].

The process of fitting a probability distribution to a set of observations follows three main steps:

1) *Exploratory Analysis.* In this step, one aims to identify a set of candidate distributions that can potentially fit the data. This may be performed qualitatively using plots (histogram, box plot, etc), or quantitatively using summary parameters of the data (mean, median, variance, symmetry and skewness measures, etc). During this phase, one would use check independence using the above mentioned tests.
2) *Parameters Estimation.* The goal of this step is to estimate the parameters of the candidate distributions. To this end, one may use Maximum of Likelihood Estimate (MLE), Moments Matching Estimate (MME), Maximum Goodness-of-fit Estimate (MGE), or Quantile Matching Estimate (QME) [17].
3) *Goodness-of-fit Test.* The obtained fits are evaluated using well-known tests, e.g., Kolmogorov-Smirnov, Anderson-Darling, and Carmer-Von Mises [14]. It is also useful to use plots like Q-Q plot and Cumulative Distribution Functions (CDF) to visually validate the fit. This phase is important because it allows selecting

[2]www.multicore-association.org/workgroup/mcapi.php

[3]http://www.itl.nist.gov/div898/handbook/eda/section3/eda35d.htm

the best fit based on the aforementioned test and other criterion like Bayesian Information Criterion (BIC) and Akaike Information Criterion (AIC).

In some cases, a pre-processing phase of the data may be required before performing the fitting. For instance, data may need rescaling and/or log transformations. Such requirements are usually detected during the exploratory analysis. To simplify the above process, we implemented a tool that assists the designer in the different steps of the distribution fitting process in R [18], [19].

### C. Calibrating Functional Models

The calibration process aims to augment functional BIP models with performance data learned as in the previous step and to produce stochastic BIP models that enables SMC. Inferred probability distributions are thus used as sampling functions for probabilistic variables representing the corresponding performance metric.

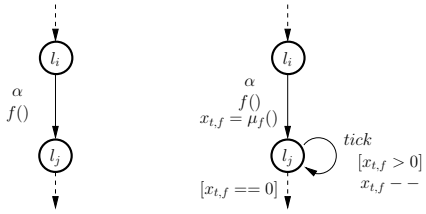


Figure 3: Calibration of BIP models with execution time.

For instance, in order to introduce the execution time of a specific function within a process, a probabilistic variable related to the corresponding learned distribution is first added. Let $f()$ be the function to calibrate, $\mu_f$ the learned probability distribution of the execution time of $f()$, and $x_{t,f}$ related to $\mu_f$ be the probabilistic variable that models its execution time. A transition $\alpha$ that calls $f()$ in the functional model is transformed as shown in Figure 3. A sampling step that updates $x_{t,f}$ is introduced on $\alpha$. The value of $x_{t,f}$ specifies the amount of time to be spent as execution time of $f()$. For timing aspects, we currently use time transitions called *tick* that models discrete time progress in BIP. We may also use the real-time capabilities of BIP [20] to capture continuous time. In the *tick* transition, the variable $x_{t,f}$ is decremented as to model time progress. Guards are used to prevent firing the next transition before the sampled execution time has completely elapsed. Therefore, a certain amount of time, modeling the execution time of $f()$ on the hardware architecture, is spent.

To correctly represent time in BIP, all the timed components, having *tick* transitions, have to be correctly synchronized to enable overall time progress. A bit of care is needed to build such representations since bad synchronization of timed components lead inevitably to deadlocks. In the above description we mainly focused on timing aspects. Other performance aspects such as energy or temperature, can be similarly handled by introducing probabilistic variables modeling temperature or energy evolution.

## IV. Case Study: HMAX Models Algorithm

In this section we aim to illustrate our approach on a real-life case study for image recognition. The goal is to give a first insight on its concrete applicability.

*Application Overview:* HMAX [7] is a hierarchical computational model of object recognition which attempts to mimic the object recognition of human brain. Recognition typically involves the computation of a set of target features at one step, and their combination in the next step. A combination of target features at one step is called a layer, and can be modeled by a 3D array of units which collectively represent the activity of set of features (F) at a given location in a 2D input grid. HMAX starts with an image layer of grayscale pixels and successively computes higher layers, alternating "S" and "C" layers. Simple ("S") layers apply local filters that compute higher-order features by combining different types of units in the previous layer. Complex ("C") layers increase invariance by pooling units of the same type in the previous layer over limited ranges.

In this case study, we only focus on the first layer of HMAX (see Figure 4) as it is the most computationally intensive. In a pre-processing phase, the input raw image is converted to grayscale (only one input feature: intensity at pixel level) and the image is then sub-sampled at several resolutions (12 scales in our case). For the S1 layer, a battery of three 2D-Gabor filters is applied to the sub-sampled images and then for C1 layer, the spatial max of computed filters across two successive scales is taken. In this application, parallelism can be exploited at several levels. First, at layer level, where independent features can be computed simultaneously. Second, at pixel level, that is, the computation of contribution to a feature may be distributed among computing resources.

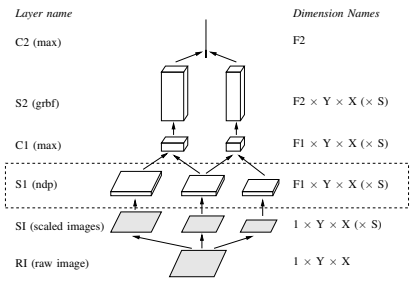


Figure 4: HMAX Models algorithm overview.

*Hardware Architecture Overview:* STHORM [8] is a power efficient manycore architecture consisting of a host processor and a manycore fabric. The host processor is a dual-core ARM cortex A9 and the fabric comprises 4 computing clusters, inter-connected via a NoC. Each cluster aggregates 16 tightly-coupled customizable 32 bits RISC

processors sharing a multi-banked level-1 (L1) data memory of 256 KBytes. Each processor has its private instruction cache with a size of 16 KBytes. All clusters share 1 MByte of level-2 (L2) memory, accessible via the NoC. A DDR3 level-3 (L3) memory of 1 GByte is also available off-chip.

*Performance Requirements:* We will mainly focus on the timing aspects of the system, that is, the overall execution time and the time to process single lines of the input image. More precisely we will compute the probabilities that the overall execution time is always lower than a given bound $\Delta$ and that the variability of processing time of successive lines is always bounded by $\Psi$. To this end, we specify our requirements as $\phi_1 = \Box^l(t < \Delta)$ where $t$ is the monitored execution time and $\phi_2 = \Box^l(|tl| < \Psi)$, where $tl$ is the difference between processing times of successive lines.

*Modeling and Code Generation:* We developed a parametric BIP model for the S1 layer of HMAX. It uses a certain number of reconfigurable processes for implementing the 2D-Gabor filtering and image splitting/joining as shown in Figure 5. Every image is handled by one "processing group" consisting of a *Splitter*, one or more *Worker* processes and a *Joiner*, connected through FIFO channels. This model exploits parallelism both at image level, as different images are processed in parallel by different processing groups and at pixel level, as different stripes of the image are processed in parallel by different *Worker* processes.

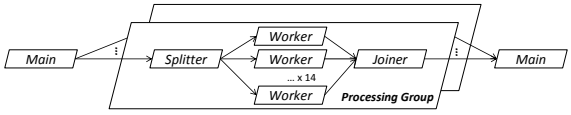


Figure 5: The S1 layer of HMAX Models algorithm.

The computation of the entire S1 layer is coordinated by a single main process. Several image scales are handled concurrently. That is, the twelve scaled images are statically pre-allocated and mapped on different processing groups. For every image scale, the processing is pipelined as follows. Initially, the main process sends the first 10 + $P$ lines to the corresponding processing group, where $P \geq 0$ is an integer parameter called *line pressure* that specifies the pipelining rate. In normal regime, one input line is sent and one output line is received, for every filter rotation (that is, actually three output lines). Finally, once all the input image has been sent, the main process receives $P$ more output lines. At this point, the processing group is ready (empty) and can be reconfigured to restart computation for another image scale.

Within the processing group, the *Splitter* receives input images, line by line from the main process. Every line is split into a number of equal length (and overlapping) fragments, one for every *Worker* process, and sent to these processes. *Worker* processes implement the computation of the 2D-filter itself (Figure 6). Filter size is fixed to 11 $\times$ 11 in the case study. Hence, *Worker* processes need to accumulate 11 line fragments in order to perform computation. Henceforth, they maintain and compute the result operating on an internal "sliding" window. Finally, the resulting fragments are sent further to the *Joiner*, which packs them into complete output lines and send them to the main process.

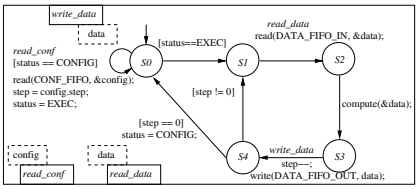


Figure 6: The BIP model of the *Worker* process.

In this first experiment, for the sake of simplicity, we only consider one image scale ($256 \times 256$), that is, one processing group will be actually used. Besides, reducing the model size, this restriction relaxes data-dependency since a single input size is considered. It is worth mentioning that, given one image scale as input, each process will always handle the same workload (amount of data) which increases the statistical learning confidence. Once the functional BIP model is available, we produce an implementation of the application using the STHORM code generator. A sample of generated code is shown below. The produced code is instrumented in order to observe execution and communication time of each process. In this paper, we rely on a physical STHORM test-board in order to gather low-level performance data. The generated implementation is therefore executed and corresponding performance traces are produced.

```
void worker_ins_execute(void* arg) { ...
while(Wlcontinue){
 switch(BIP_CTRL_LOC){
  case S0 : {
   if (status == CONFIG) { ...
    status = EXEC; BIP_CTRL_LOC = S0;}
   if (status == EXEC) BIP_CTRL_LOC = S1;
  break;}
  case S1 : {
   mcapi_pktchan_recv(h_WORKER_read_data,
   (void**)&mcapi_buffer, &mcapi_received,
   &mcapi_status);
   ...
   BIP_CTRL_LOC=S2 ; break;}
  case S2 : {
   compute(&data); BIP_CTRL_LOC=S3; break;}
  case S3 : {
   mcapi_pktchan_send(h_WORKER_write_data,
   data, size, &mcapi_status);
   ...
  break;}
  ... }}}
```

*Performance Characterization and Model Calibration:* Distribution fitting is used to learn probability distributions that fits the obtained data. We illustrate the different steps of the process on the execution time of the *Worker* process. Exploratory analysis is first performed to observe if the data provides any clues to belongs to a usual probability

distribution. Runs and Box plots are initially used to observe the data evolution and to detect the presence of outliers that may distort the analysis. The corresponding plots are presented in Figure 7 and do reveal presence of outliers. Figure 8 shows the same plots after removing these points.

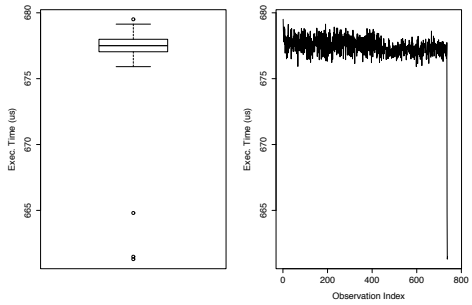


Figure 7: Box and Runs plots of the *Worker* execution time.

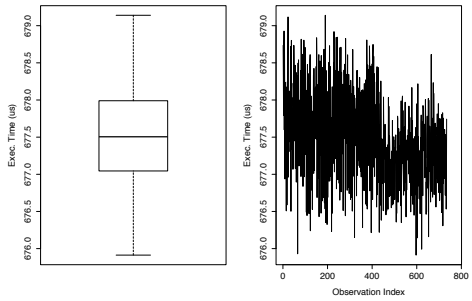


Figure 8: Box and Runs plots of the *Worker* execution time after outlier elimination.

To check if the data observations are independent, we first use the Lag plot. This draws the observations $x_i$ in the y-axis and $x_j$ in the x-axis, where $i-j$ is the fixed lag. For instance, Figure 9 shows the Lag plot of the *Worker* execution time with lag equal to 1. The figure clearly shows a random repartition of the observations. To get more confidence, we used the Ljung-Box and the Box-Pierce tests at significance level of $0.05$. These gave respectively $0.0531$ and $0.0533$ as p-values which confirms the independence assumption.

Finally, we used the histogram and the CDF in Figure 10 to observe the shape of the data. One can see out of this figure that the data is uni-modal and symmetric which means that it may be potentially generated from bell-curved process. We use the Cullen and Fray graph illustrated in Figure 11 to get more insight with respect to the Skewness and the Kurtosis of the data. The figure shows that the observations are seemingly Normal.

The second step in the distribution fitting process is to fit the candidate distribution to the data, that is, in this case, the Normal distribution. This consists of estimating its

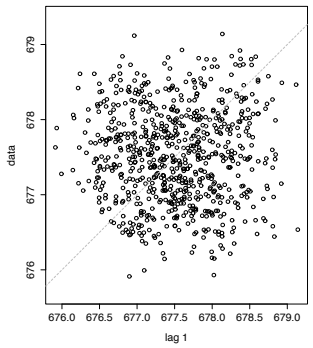


Figure 9: Lag plot of the *Worker* execution time.

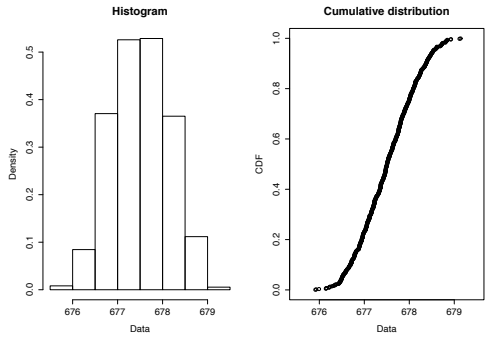


Figure 10: Histogram and CDF of the *Worker* execution time.

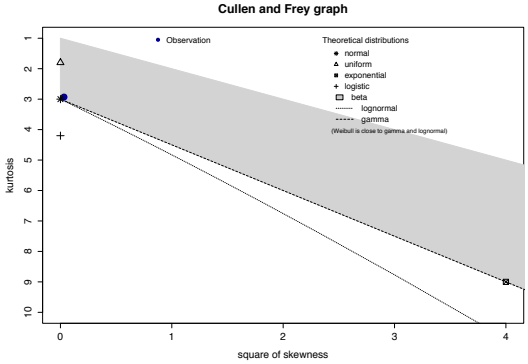


Figure 11: Cullen-Frey graph for the *Worker* execution time.

parameters out of the data. Since the candidate distribution is a Normal one, its mean $\mu$ and its standard deviation $\sigma$ have to be estimated. To this end, we used the method of moment which gave the following estimates $\mu = 2262.265\mu s$ and $\sigma = 1.28\mu s$. The fitted Normal distribution is shown in Figure 12 which illustrates several comparisons between the learned Normal distribution and the actual data. It mainly compares the density functions, CDFs, in addition to quantiles and probabilities.

The final step in this learning process is to evaluate

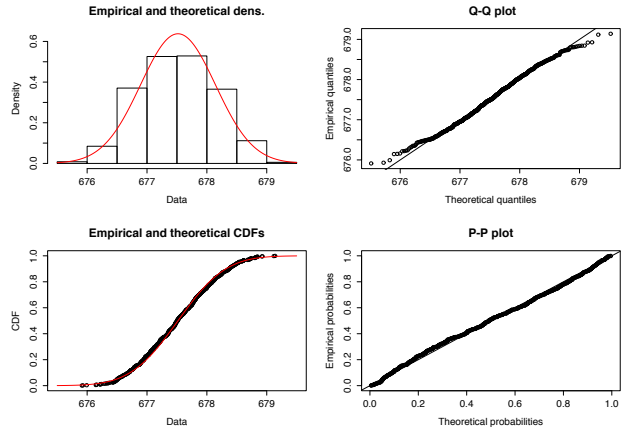


Figure 12: Fitting execution time to a Normal distribution.

the performed fit through a goodness-of-fit test. Anderson-Darling statistic at a significance level of 0.05 was used in this case. It gave a p-value 0.18 which is greater than 0.05 (the significance level). This means that we cannot reject the hypothesis stating that the data is normally distributed with $\mu = 2262.265\mu s$ and $\sigma = 1.28\mu s$.

We applied the same steps for the processes communications time and learned similar distributions. For space constraints, we illustrated the different steps once. It is worthwhile mentioning that, in this case study, only the *Worker* process computation time was considered since we observed that it is the most important. For the other processes, only the communication time was learned.

Once all performance aspects of interest characterized, calibration is performed by annotating the functional BIP model using the learned distributions as illustrated in Section III. Figure 13 shows an example of calibrated BIP component. This corresponds to the *Worker* process in Figure 6. The latter is augmented with timing information that concerns communication, i.e., read and write and computation. Three different probability distributions where learned to this end, namely f_read(), f_write(), and f_exec().

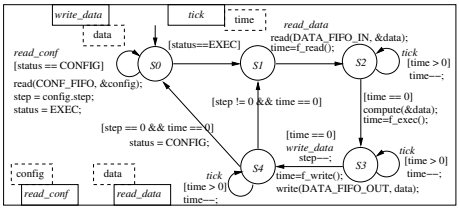


Figure 13: Calibrating the *Worker* with timing information.

*Performance Evaluation:* Before using SMC to check the system-level timing requirements, we wanted to validate the calibrated model with respect to the actual implementation. To this end, we compared the overall execution time of the generated HMAX implementation running on the test-board and the calibrated BIP model. We observed that the time on the model is about 20% lower than what we obtained on the test-board. This result is expected since the calibrated model does not take into account all the implementation delays. For instance, the splitting and joining time were not introduced in the model. Moreover, high-level models are generally more optimistic due to abstraction.

Table I: Probabilities of $\phi_1$ when varying $\Delta$ $(P = 0)$.

| $\Delta(ms)$ | 572.75 | 572.8 | 572.83 | 572.85 | 572.89 | 572.95 |
|---|---|---|---|---|---|---|
| *Prob.* | 0 | 0.28 | 0.57 | 0.75 | 0.98 | 1 |
| *Traces* | 66 | 1513 | 1110 | 488 | 171 | 66 |

Now that the high-level model is correctly calibrated with timing information, it can be safely used for performance evaluation using SMC. We used the SPRT algorithm implemented within the SBIP statistical model checker [21] with confidence parameters $\alpha = \beta = 0.001$ and $\delta = 0.05$. We checked the aforementioned performance requirements $\phi_1$ and $\phi_2$ for different pipelining rate $P = 0, 2$. In this experiment, we used arbitrary fifo sizes: *Main-Splitter*= 10 KB, *Splitter-Worker*= 112 B, *Worker-Joiner*= 336 B, and *Joiner-Main*= 30 KB. In the future, we are going to use SMC to find the best configuration minimizing the overall execution time. Table I shows the probability evolution of $\phi_1$ for different $\Delta$ and the corresponding required SMC traces. One can for instance conclude out of this table that the overall execution time is always lower than $752.95ms$ with probability 1. In Figure 14, we present two results of verifying $\phi_2$ when varying $\Psi$. The curve on the left is obtained with no pipelining $(P = 0)$ while the one on the right is obtained with $P = 2$. The two curves show similar evolutions with a small difference in the bounds. The curve on the right $(P = 2)$ has actually greater values, that is, more variation. We recall that when $P = 0$, all the processes are perfectly synchronized which yields to small variation over line processing time. Using $P > 0$ leads to greater variation since it somehow alter this synchronization. We finally mention that the SMC time was relatively small given the model size (47 BIP components). It took us about 5 hours in average for each curve.

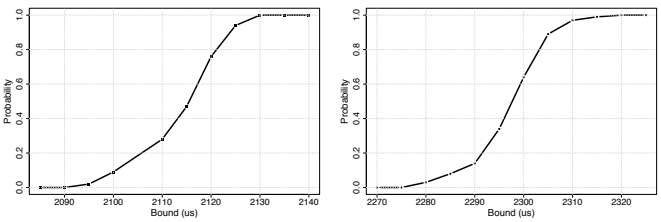


Figure 14: Probabilities of $\phi_2$ as function of $\Psi$ for $P = 0$ (left) and $P = 2$ (right).

## V. Related Work

State-of-art techniques for gathering low-level performance information in early design phases can be classified in three different families. The first uses documentation, e.g., data sheets. The second is based on source/binary/object code, e.g., static analysis, code inspection. The third technique is more accurate and is the most used, albeit the most time and resource consuming. It relies on executable, i.e., high/low-level simulation or execution.

Several frameworks for system-level design uses these techniques together with model calibration to improve the accuracy of high-level models. Model calibration (also referred to as back-annotation) is a well-known and widely-used technique. However, it only received a little attention in the system-level design community. Few existing frameworks consider and implement it differently. In [22], Pimentel et al. use low-level simulation and synthesis to calibrate architecture models in the context of the Sesame simulation framework [23]. The proposed techniques rely on instruction-set simulator (ISS) and automatic synthesis for FPGAs to build latency tables associated with architecture model components. Within their system-level performance evaluation method [24] for the MILAN framework [25], Mohanty and Prasanna propose to calibrate parametrizable abstract models of SoC architecture using interpretive simulation techniques. The obtained measures are characterized as average estimates. In [26], Haid et al. propose an approach for automatic code generation and calibration of compositional performance analysis models. The approach uses high/low-level simulation and data sheets to obtain performance details in order to calibrate the generated models. Giusto et al. [27] propose an improvement of the VCC methodology using back-annotation of high-level behavioral models. Similarly to our work, they perform estimation using a statistical approach but consider a single microprocessor as opposed to our method that works for manycore architecture. In addition, they use linear regression techniques while we use distribution fitting.

The aforementioned frameworks also encompass performance evaluation techniques and rely on established modeling formalisms. For instance, the Artemis workbench [28], which is based on the Sesame environment [23] discussed above, begins with Matlab Simulink representations of the application and a systemC model of the hardware architecture. For performance evaluation, it uses simulation and co-simulation techniques. Frameworks such as [26], [29] use formal methods for system-level performance analysis like SymTA/S [30] or Real Time Calculus [31]. These rely on analytical techniques to determine latencies, worst-case scheduling scenarios, buffer sizes, which requires to build correct abstractions of the application and the architecture. Moreover, they generally produce pessimistic estimations. MetaMoc [32] is also a tool that uses formal methods but is more related to Worst Case Execution Time (WCET) and schedulability analysis for hard real-time embedded software. It is based on UPPAAL [33] and uses model checking combined with static analysis techniques.

Similarly to these methods, we use low-level simulation or concrete execution if possible to gather low-level performance details. Conversely, our proposal relies on a statistical characterization of low-level performance data. Moreover, for performance evaluation, we use SMC, which unlike pure simulation, provides statistical guarantees and is easier to apply than analytical analysis.

## VI. Conclusion

We presented a new approach for modeling and analyzing performance in the context of system-level design. We proposed building high-level performance models through automatic code generation from high-level specifications and statistical inference, which enables probabilistic characterization of low-level performance data. This probabilistic representation accurately captures the data variability since based on concrete execution or low-level simulation. Functional models are then calibrated with the learned probabilities to produce high-level stochastic models encompassing performance details. We use Statistical Model Checking to quantitatively analyze these models with a fixed confidence.

The approach was illustrated and validated on a fragment of a real-life case study for image recognition. We provide tool support, developed within the BIP framework, for its different phases. The first results of this experiment are satisfying. They show that it is practically applicable although, to some extent, difficult. The statistical analysis of data requires some expertise and a deep knowledge of the system. Note that for the statistical analysis part, we only developed the distribution fitting technique which is one among other possible model fitting alternatives. It is worth to recall that distribution fitting is not always possible. For instance, when the independence assumption is not satisfied. This may be a hindrance towards building the performance model. For this reason, we aim to investigate other model fitting techniques such as regression analysis [27] or learning Markov models [34]. In the future, we are also planning to continue exploring the HMAX case study from other perspectives such energy consumption and temperature.

## References


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

[2] T. Hérault, R. Lassaigne, F. Magniette, and S. Peyronnet, "Approximate Probabilistic Model Checking," in *VMCAI*, January 2004, pp. 73–84.

[3] H. L. S. Younes, "Verification and planning for stochastic processes with asynchronous events," Ph.D. dissertation, Carnegie Mellon, 2005.

[4] A. Basu, S. Bensalem, M. Bozga, P. Bourgos, M. Maheshwari, and J. Sifakis, "Component assemblies in the context of manycore," in *FMCO*, 2011.

[5] A. Basu, B. Bensalem, M. Bozga, J. Combaz, M. Jaber, T.-H. Nguyen, and J. Sifakis, "Rigorous Component-Based System Design Using the BIP Framework," *IEEE Software*.

[6] A. Nouri, S. Bensalem, M. Bozga, B. Delahaye, C. Jégourel, and A. Legay, "Statistical model checking QoS properties of systems with SBIP," *Int. Journal on Software Tools for Technology Transfer*, pp. 1–15, 2014.

[7] J. Mutch and D. G. Lowe, "Object class recognition and localization using sparse features with limited receptive fields," *Int. Journal of Computer Vision*, 2008.

[8] D. Melpignano, L. Benini, E. Flamand, B. Jego, T. Lepley, G. Haugou, F. Clermidy, and D. Dutoit, "Platform 2012, a many-core computing accelerator for embedded SoCs: performance evaluation of visual analytics applications," ser. DAC, 2012.

[9] A. Pnueli, "The temporal logic of programs," in *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, ser. SFCS '77. Washington, DC, USA: IEEE Computer Society, 1977, pp. 46–57.

[10] A. Wald, "Sequential tests of statistical hypotheses," *Annals of Mathematical Statistics*, vol. 16(2), pp. 117–186, 1945.

[11] F. Balarin, M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, E. Sentovich, K. Suzuki, and B. Tabbara, Eds., *Hardware-software Co-design of Embedded Systems: The POLIS Approach*. Norwell, MA, USA: Kluwer Academic Publishers, 1997.

[12] B. Bonakdarpour, M. Bozga, M. Jaber, J. Quilbeuf, and J. Sifakis, "A framework for automated distributed implementation of component-based models," *Distributed Computing*, vol. 25, no. 5, pp. 383–409, 2012.

[13] J.-Y. Le Boudec, *Performance Evaluation of Computer and Communication Systems*. EPFL Press, Lausanne, Switzerland, 2010.

[14] D. Vose, *Risk analysis : a quantitative guide*. Wiley, 2008.

[15] G. E. P. Box and D. A. Pierce, "Distribution of Residual Autocorrelations in Autoregressive-Integrated Moving Average Time Series Models," *Journal of the American Statistical Association*, vol. 65, pp. 1509–1526, 1970.

[16] G. M. Ljung and G. E. P. Box, "On a measure of lack of fit in time series models," *Biometrika*, vol. 65, no. 2, pp. 297–303, Aug. 1978.

[17] G. Cowan, *Statistical Data Analysis*. Oxford University Press, Oxford, 1998.

[18] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2014. [Online]. Available: http://www.R-project.org

[19] M. L. Delignette-Muller, R. Pouillot, J.-B. Denis, and C. Dutang, *fitdistrplus:help to fit of a parametric distribution to non-censored or censored data*, 2014, r package ver. 1.0-2.

[20] T. Abdellatif, J. Combaz, and J. Sifakis, "Rigorous implementation of real-time systems - from theory to application," *Mathematical Structures in Computer Science*, vol. 23, pp. 882–914, 2013.

[21] S. Bensalem, M. Bozga, B. Delahaye, C. Jégourel, A. Legay, and A. Nouri, "Statistical model checking QoS properties of systems with SBIP," in *ISoLA (1)*, 2012, pp. 327–341.

[22] A. D. Pimentel, M. Thompson, S. Polstra, and C. Erbas, "Calibration of abstract performance models for system-level design space exploration," *J. Signal Process. Syst.*, vol. 50, no. 2, pp. 99–114.

[23] A. D. Pimentel, C. Erbas, and S. Polstra, "A systematic approach to exploring embedded system architectures at multiple abstraction levels," *IEEE TRANSACTIONS ON COMPUTERS*, vol. 55, no. 2, pp. 99–112, 2006.

[24] S. Mohanty and V. K. Prasanna, "Rapid system-level performance evaluation and optimization for application mapping onto SoC architectures," in *in Proc. of the IEEE Int. ASIC/SOC Conference*, 2002.

[25] A. Bakshi, V. K. Prasanna, and A. Ledeczi, "Milan: A model based integrated simulation framework for design of embedded systems," ser. OM '01, 2001, pp. 82–93.

[26] W. Haid, M. Keller, K. Huang, I. Bacivarov, and L. Thiele, "Generation and calibration of compositional performance analysis models for multi-processor systems," ser. SAMOS'09, pp. 92–99.

[27] P. Giusto, G. Martin, and E. Harcourt, "Reliable estimation of execution time of embedded software," ser. DATE '01, pp. 580–589.

[28] A. D. Pimentel, "The artemis workbench for system-level performance evaluation of embedded systems," *Int. J. Embedded Systems*, 2005.

[29] L. Thiele, I. Bacivarov, W. Haid, and K. Huang, "Mapping applications to tiled multiprocessor embedded systems," in *ACSD*, 2007.

[30] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, "System level performance analysis - the symta/s approach," in *IEEE Proceedings Computers and Digital Techniques*, 2005.

[31] L. Thiele, S. Chakraborty, and M. Naedele, "Real-time calculus for scheduling hard real-time systems," in *ISCA*, vol. 4, 2000, pp. 101 –104 vol.4.

[32] A. E. Dalsgaard, M. C. Olesen, M. Toft, R. R. Hansen, and K. G. Larsen, "METAMOC: Modular execution time analysis using model checking," in *WCET*, 2010, pp. 113–123.

[33] G. Behrmann, A. David, and K. G. Larsen, “A tutorial on uppaal,” in *SFM*, 2004, pp. 200–236.

[34] A. Nouri, B. Raman, M. Bozga, A. Legay, and S. Bensalem, “Fatser statistical model checking by means of abstraction and learning,” in *Runtime Verification*, September 2014.