Performance Engineering for Industrial Embedded Data-Processing Systems*

Martijn Hendriks[1], Jacques Verriet[1], Twan Basten[1,3], Marco Brassé[2], Reinier Dankers[2], René Laan[2], Alexander Lint[2], Hristina Moneva[1], Lou Somers[2,3], and Marc Willekens[1]

[1] Embedded Systems Innovation by TNO, Eindhoven, The Netherlands
[2] Océ Technologies B.V., Venlo, The Netherlands
[3] Eindhoven University of Technology, Eindhoven, The Netherlands

Abstract. Performance is a key aspect of many embedded systems, embedded data processing systems in particular. System performance can typically only be measured in the later stages of system development. To avoid expensive re-work in the final stages of development, it is essential to have accurate performance estimations in the early stages. For this purpose, we present a model-based approach to performance engineering that is integrated with the well-known V-model for system development. Our approach emphasizes model accuracy and is demonstrated using five embedded data-processing cases from the digital printing domain. We show how lightweight models can be used in the early stages of system development to estimate the influence of design changes on system performance.

1 Introduction

Performance metrics such as throughput and latency are key to many (embedded) systems. Typically, however, system-level performance tests only are available late in the development process, because they require an assembled system. It is well known that late changes are more expensive to make than earlier changes [21]. Therefore, if these tests show problems, then repairing these is expensive, if possible at all without major re-design. Having early insight in system-level performance metrics therefore is beneficial: it can decrease time-to-market, reduce development cost and improve quality of the product.

Architectural models such as the 4+1 architectural view model [12] and the CAFCR model [14] usually include performance engineering activities in order to obtain early insight in system-level performance. Nevertheless, these activities are not always done in practice. A recent study [15] identifies a number of major current problems in the wider scope of model-driven engineering. These also apply to model-based performance engineering and include the following issues: (i) limited tool usability, (ii) inconsistencies between artifacts (e.g., between model and realization), (iii) lack of fundamentals / body of knowledge, (iv) lack

* This work is partially supported by the ITEA2 project 11013 PROMES.

of industrial evidence of benefits. Not explicitly mentioned in [15] and more specific for performance engineering is (v) the difficulty of building *sufficiently accurate* predictive models. This often is an implicit but important assumption, because non-accurate models can turn model-based performance engineering into a hazard instead of a benefit. Issues (i), (ii), (iii) and (v) are obstructions that increase the cost of performance engineering, and issue (iv) directly addresses the lack of evidence that performance engineering is beneficial. Our interpretation of this is that in practice it is often felt that the difficulties and cost of applying performance engineering outweigh the potential benefits: there is no business case. Making a business case for performance engineering such as described, e.g., in [22] is hard, because it is very difficult to quantify both the tangible and the non-tangible benefits without extensive empirical studies. Our work tries to solidify the business case in the particular domain of embedded data-processing systems by a description of how we have applied a process, techniques and tools to five industrial cases, and by a discussion of the costs and benefits.

Contribution The paper has two contributions. Firstly, we couple a model-based performance engineering process to the well-known V-model development process [9]. We address model consistency (issue (ii)) and accuracy (issue (v)) by explicit calibration and validation steps, and by assuming evolutionary development. The performance engineering process allows arbitrary modeling tools, and is not part of the critical path in the development process; it can be applied when deemed necessary. There is much work on performance engineering, see, e.g., the survey [3]. Although the process, calibration, and validation ingredients are described by others as well, see, e.g., [16, 18–20], we believe that our description makes an extra step in bringing these ingredients together in the full engineering scope. Secondly, we present how we have used two well-established techniques (discrete-event simulation and regression analysis) with the earlier described process in five industrial cases. The techniques and tools have been selected because they are relatively easy to use (issue (i)). The ideas in these cases add to the body of knowledge (issue (iii)), and their results to the evidence of the benefits of model-based performance engineering (issue (iv)).

Outline We describe the process that we use in Sect. 2. Section 3 describes the techniques and tools that we have applied in the cases, which are consequently described in Sect. 4. Finally, Sect. 5 concludes.

2 Performance Models and the Development Process

Figure 1 shows a view which generalizes our way of working in several industrial cases (see Sect. 4). It relates the performance engineering process to the regular V-model development process [9]. The precondition of this hybrid process is that there is an existing system that is being evolved with a V-model iteration. The definition of existing system and iteration can be very broad. We distinguish the following steps:

1. Requirements analysis for a new system Y, the successor of an existing system X, leads to a number of performance-related questions. For instance,

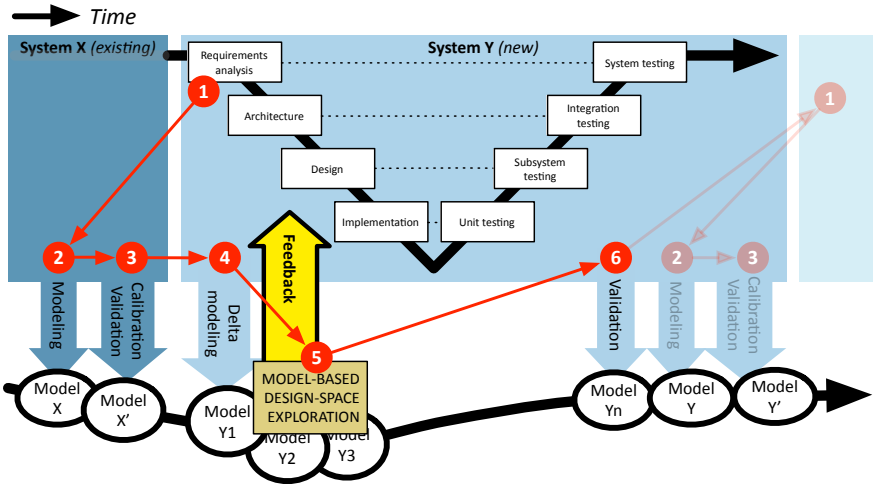

Fig. 1. Performance models in an iterative V-model development process.

system Y might be required to have a higher throughput than X. A typical question then is how much the bottleneck components must be improved in order to achieve the new requirements.

2. An initial model of system X is built based on the initial questions.
3. The model of system X is calibrated and validated with respect to the requirements: we *predict the past*. This is a key step as it builds a certain degree of trust in the model and its predictions.
4. The envisioned changes of system Y compared to system X are incorporated in the calibrated and validated model. This step gives a predictive model, or a set of models (one for each design alternative), for system Y.
5. We use model analysis to explore the design alternatives: we *explore the future*[4]. The analysis results are input for the architecture and design steps.
6. After system Y has been realized, the predictive models can be validated against the actual realization. This retrospective validation builds experience, and allows us to reconcile the model with reality. This completes the iteration and brings us in the position where we might be able to re-use the models for a new V-model iteration.

The value of predictive models is mainly determined by their accuracy, and reasonably accurate models are a necessary precondition for model-based performance engineering. The sketched approach assumes a rather gradual development process with relatively small iterative steps. This enables us to take advantage of the existing system: calibration and validation on the existing system builds trust in the modeling approach and gives an indication of the accuracy of

[4] The phrase "predict the past, explore the future" originates from [16].

the models. Predictions in step 5, however, come from unvalidated models (due to changes introduced in step 4), and therefore always have an unknown accuracy. The development-validation-gap thus bears the risk of significantly wrong model predictions. We distinguish two cases to discuss this problem based on the *cost-to-validation*, which is the cost of the development process that is needed after step 5 to enable step 6. Firstly, we can have the situation with a small cost-to-validation, i.e., the model predictions of step 5 can relatively easily be validated. An example is the optimization of software parameters such as buffer sizes by model-based analysis. Insufficiently accurate model predictions only waste some time in this case, and the validation results can be used for additional calibration. The authors of [4] identify easy validation as one of the success factors of their model of the paper path inside a printer that nowadays is used by industry. Secondly, we can have the situation in which validation of the model prediction can be done only after further development steps (e.g., after several weeks of development and integration), or is not possible at all (e.g., when model predictions lead to the choice to *not* follow a certain design). Insufficiently accurate model predictions can then lead to significant waste of time and engineering resources or a suboptimal design. Thus, using model predictions in the situation with a non-negligible cost-to-validation implies a certain amount of trust in the model. We believe that experience and a set of best practices (that can be domain-specific) can help to mitigate the risks of this situation.

The modeling line in Fig. 1 is loosely coupled to the development process, which is to say that it can be skipped when it is not deemed necessary. We also do not make assumptions about how the models are created; this could be a manual process, or, on the other extreme, a fully automated process that generates performance models from design artifacts.

Although our process does not fix modeling formalisms, we do follow some modeling principles that have proved to be useful. The most important principle that we use in building our performance models, is that of the separation of concerns provided by the Y-chart [2]. This pattern decomposes a system model into an application model, a platform model, and a mapping model. The application model describes the functionality of the system; it typically describes the application's tasks, their computational loads, and their interdependencies. The platform model describes the resources of the computational platform and their capabilities. Platform resources include processing resources such as CPUs and storage resources such as memories, but also bandwidths and energy can be platform resources. The mapping model describes the deployment of the application tasks onto the platform resources.

To successfully apply the modeling process described above, it is essential to define models that accurately describe a system's performance. However, models should also be as simple as possible, because simple models require less effort to develop and to maintain. Simple models typically require little analysis time, which is beneficial for design-space exploration. On the other hand, the models must be sufficiently accurate for the purpose they serve. If a high accuracy is required, then typically a high model complexity is required. For instance, a

cycle-accurate simulation is very accurate, but also very complex. In this paper, we will focus on simpler models that are accurate enough to *predict the past* and *explore the future*.

We distinguish three types of performance models with a varying degree of system knowledge and therefore with a different level of complexity. *Black-box models* are system models that do not use any a priori knowledge of internal behavior of the system. These models typically only consider the system's input and output behavior and are derived using experimental data [5]. *White-box models* are models that have all a priori knowledge about the internal workings of a system. As the structure of a white-box model is known, it is simpler to calibrate them. On the other hand, having system behavior knowledge typically results in more complex models. We distinguish a third model type, which combines both system knowledge and experimental data: *gray-box models* are created using high-level information about a system's internal behavior and corresponding experimental data. In this paper, we only use white-box and gray-box models, as we always have application knowledge and using this knowledge simplifies performance engineering.

3 Techniques and Tools

3.1 Discrete-Event Simulation and Execution Visualization

Discrete-event simulation is a methodology that discretizes the evolution of systems through a sequence of time-stamped events [8]. It allows the user to estimate all kinds of properties such as system throughput and processing latency. Discrete-event simulation is widely applicable, and often discrete-event simulation models are easier to manipulate and experiment with than the system under study. Another key benefit is that discrete-event simulation creates additional insight in the system dynamics.

In [10] we have presented the OctoSim framework, a discrete-event simulation approach for software-intensive embedded systems on a high level of abstraction that is based on a piecewise-linear notion of progress of computational tasks. The Y-chart based approach separates application, platform and mapping, and additionally separates the *execution* of tasks from the *application structure*. This results in models consisting of loosely coupled components with excellent opportunities for re-use. The OctoSim framework is available as a Java library, and can be classified as an embedded domain-specific language. The use of a general-purpose programming language has the advantage that all existing and often highly mature tooling for this language can be used. For instance, the Eclipse Integrated Development Environment (IDE) for Java developers can be used as a modeling environment [6]. This IDE includes a plethora of functionality, e.g., debugging, which in particular is useful during model development. The fact that the OctoSim library is based on Java interfaces makes the modeling language very flexible and powerful.

The OctoSim discrete-event simulation provides a simulation run (or a set of runs) which can be analyzed for key performance metrics. In addition to these

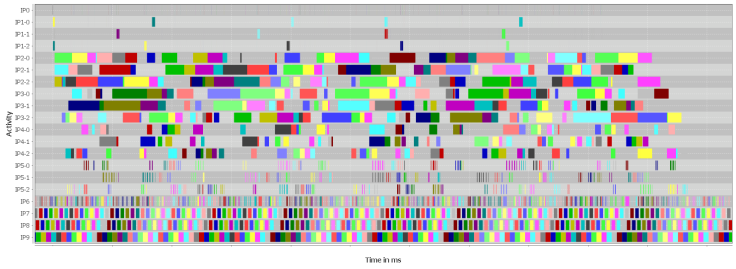

Fig. 2. A sample TRACE view which shows a system's activity over time.

system-level numbers, we have experienced that detailed insight in the dynamics of the system is highly valued for both model validation, and to understand system-level numbers that are not expected. Therefore, we have coupled the TRACE tool, a Gantt chart viewer and analyzer, to our OCTOSIM tool [7]. Figure 2 shows an example of an OCTOSIM simulation run in the TRACE tool. TRACE enables us to zoom into the details of the behavior, and enables analysis methods to, e.g., compute and show differences in behavior or to compute and visualize the critical activities and resources [11]. TRACE is available as an ECLIPSE plug-in and therefore the full discrete-event simulation tool chain is available in a single, mature and widely-used IDE.

3.2 Regression Analysis

Regression analysis is a statistical technique for investigating and modeling the relationship between variables [13]. It involves *response variables* y_i and (independent) *predictor* or *regressor variables* $x_{i,j}$. The goal of regression is finding a regression model, i.e., a model that relates response variables y_i and regressor variables $x_{i,j}$. This is a function f, such that $y_i = f(x_{i,1}, \ldots, x_{i,m}) + \epsilon_i$, where ϵ_i is a zero-mean *error* or *residual*.

There are several types of regression depending on the structure regression function f. The most common type is linear regression, which finds *regression coefficients* $\beta_0, \beta_1, \ldots, \beta_m$, such that $y_i = \beta_0 + \sum_{j=1}^{m} \beta_j \cdot x_{i,j} + \epsilon_i$. Linear regression uses *least-squares estimation* to compute *intercept* β_0 and *slopes* $\beta_1, \ldots, \beta_m$. Besides estimations for the intercept and the slopes, linear regression also provides information on the quality of the found fit. Examples are the *coefficient of determination* R^2, which is the proportion of variation explained by regressors $x_{i,j}$, and an assessment of the significance of the intercept and slopes. Details about these and other supported techniques can be found in the textbook by Montgomery et al. [13].

In this paper, we use regression to create parameterized (white-box or gray-box) models that are used for performance prediction. Using system knowledge, we first determine a model template, i.e., a function f describing the relation between input parameter values and system performance. For linear models, the template equals $y_i = \beta_0 + \sum_{j=1}^{m} \beta_j \cdot x_{i,j}$, where $x_{i,j}$ represents input parameters

that can be controlled and β_i are constants. We use regression to calibrate the identified models. For the example of linear regression, this corresponds to determining the constants β_i using measurements of y_i for different values of input parameters $x_{i,j}$. Section 4 describes two cases, in which regression is used for model calibration; one case uses linear regression, the other non-linear regression.

4 Cases

In this section we describe five cases from industrial practice in which we have applied performance engineering techniques to aid the development process. The design questions span all three parts of the Y-chart: Sects. 4.1, 4.3, and 4.5 describe choices with respect to the application, Sect. 4.2 analyzes platform design alternatives, and Sect. 4.4 considers mapping alternatives.

4.1 Predicting the Effect of Application Changes

Problem Description The first case involves the data paths of a family of wide-format printing systems. These printers can print images of over one meter in width. These images are printed by an ink-jet carriage that moves over the medium, e.g., a paper role. An image is printed in bands, called *swaths*; the height of these swaths equals the number of nozzles in the printer carriage and their width equals the image width. The corresponding data paths have to handle a huge amount of data. They take a bitmap as input and transform this bitmap into sequences of firing moments for the (thousands of) nozzles of the printers' ink-jet carriage. The data paths of the wide-format printers are different for each printer in the product family, but they are all built from a single library of generic image processing steps. Examples of these steps are copying, masking, re-sampling, and transposition.

We would like to quickly and accurately predict the performance of the data path of future wide-format printing systems. For instance, in an early development phase, we would like to assess the influence of different printer configurations on the data path performance. Examples of these configuration changes include a larger number of ink-jet nozzles or different printer dimensions. Similarly, we would like to accurately predict the performance of different sequences of image processing steps.

Performance Engineering As we have access to the code of all image processing steps of an existing prototype, a white-box modeling approach has been used to address this challenge. The modeling step (step 2 in Fig. 1) mainly consisted of code analysis, which was used to determine the (nested) loops in the data path code and the corresponding loop bounds. This loop structure determines a template for the performance model. For instance, if the implementation contains a single loop iterating over the pixels of a swath, then this operation's latency can be expressed as $\beta_0 + \beta_1 \cdot n$, where n is the number of pixels in the swath and β_0 and β_1 are unknown constants. This determines the model structure, which is later used for calibration using (linear) regression.

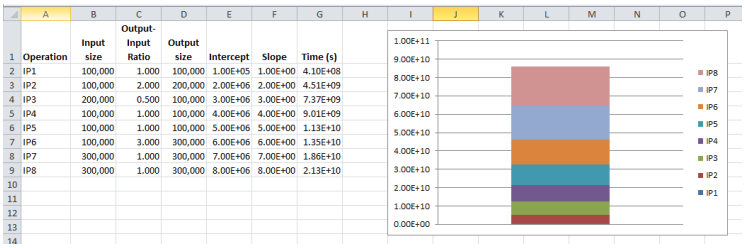

Fig. 3. Simplified Excel performance model.

Calibration and validation (step 3 in Fig. 1) has started with code instrumentation and systematic measurements. Code instrumentation has been applied to record the start and end times of image processing steps using an accurate clock. This additional code disturbs the measurements. However, the measurement overhead is very small compared to the execution times of the image processing steps. Measurements have been performed using different inputs; the inputs are selected using information of the loop bounds found during code analysis. Measurements with varying loop bounds were performed. For instance, if an image processing steps has an expected latency of $\beta_0 + \beta_1 \cdot n$, where n is the number of pixels in the swath and β_0 and β_1 are unknown constants, then the set of measurements should cover different values of n. Finally, linear regression was used to calibrate a performance model for each image processing step separately. The regressors are the loop bounds found using code analysis. For the example of an expected latency of $\beta_0 + \beta_1 \cdot n$, n is used as the only regressor. Linear regression will determine the values of intercept β_0 and slope β_1. We have validated the model by comparing model predictions for a new input (with a different number of pixels in the swath) with measurements. The differences are approximately 1%.

A simplified and anonymized version of the resulting performance model is shown in Fig. 3. This model takes only one regressor, the input size. It is a simple model in Excel with a single input parameter, i.e., the size of the input of image processing step IP1. The input sizes of the subsequent steps are determined by the preceding steps, as for each step it is known how much output it produces for a given input size. By varying the sequence of image processing steps or the size of the input of the first step, different printer configurations can be evaluated and used for exploring the future (steps 4 and 5 in Fig. 1).

Costs & Benefits The procedure used to estimate the performance of wide-format printers is a simple process, which allows a large degree of automation. The following benefits have been observed from using the data path performance model, both during data path development and for design-space exploration.

First, bottleneck identification. The performance model identifies which of the image processing steps are most expensive. This allows the data path designers to focus their attention in optimizing the data path performance on those image processing steps that contribute most to the execution time. Second, hard-

ware scaling analysis. The input size is the main regressor of most of the image processing models. This allows exploration beyond the dimensions of existing wide-format printers. If there are plans for a printer that allows larger input widths or larger ink-jet carriages, then the model provides first predictions of the future data path performance. This gives an early impression whether the data path performance will be sufficient for the new printer configuration. Third, sequence optimization. A typical data path contains a number of re-sampling steps; these change the resolution of the image. Having the optimal sequence of image processing steps is essential in optimizing data path performance; the most expensive steps should be executed on as little data as possible. The predictive model allows all sequences to be specified and analyzed separately; the data path designer does not need to implement and test all sequences individually. Instead, the optimal sequence can be derived using the model and data path designers can implement this sequence. Fourth, detection of unexpected performance behavior. During code analysis, the loop structure of the image processing steps is identified. This is used as input for calibration using linear regression. In principle, each of the loop bounds should appear as a significant regressor in the corresponding model. However, we have unexpectedly identified an operation for which the input size was not considered a significant regressor. This has led to special attention to analyze and correct the unexpected behavior.

4.2 Predicting the Effect of Platform Scaling

Problem Description The second case involves the data path of a high-production cut-sheet printer. Like the wide-format data path, this data path transforms an input bitmap into firing moments. The data path takes the input image, e.g., of A4 size, and divides it into a number of bands. These bands are processed in parallel individually by separate threads of the data path; each thread runs on a dedicated processor core. When there are more bands than processor cores, then bands are processed sequentially. This case involves a platform-scaling question: we would like to estimate the performance of the data path on different hardware platforms without individually purchasing and testing all of them. Using these estimations, discussions are fed to select the computational hardware of the printer being developed with the best balance between cost and performance.

Performance Engineering We applied a gray-box modeling approach in which we use the information that a large part of the application is parallelized over the available processor cores of the platform. Estimating the performance of an application on a (multi-core) computer platform is very challenging, because (i) there are many platform parameters that influence system performance (e.g., the number and type of CPU cores, bus capacities, cache sizes and cache line sizes), and (ii) it is generally not possible to vary only one of the parameters to isolate its effect on performance. To create a simple, yet accurate, predictive performance model, it is essential to identify the parameters that have the highest influence on data path performance. Experiments have, unsurprisingly, identified the number of CPU cores as the most important platform parameter.

Our performance model that predicts application performance on a new platform consists of three parts (step 2 in Fig. 1). Amdahl's law [1], which estimates the maximum possible speed-up of an application running on parallel processors has been used for the first part of our model. It divides the computational load of the application into two parts: a sequential part, which cannot be parallelized, and a parallel part, which can be ideally parallelized. Amdahl's law can be formulated as $T(n) = T(1) \cdot (s + \frac{1}{n} \cdot (1 - s))$, where $T(n)$ is an application's execution time on n processors and $s \in [0, 1]$ is the sequential fraction of the application. Changing the platform, however, will affect both $T(1)$ and s (where we assume that Amdahl's law still applies on the new platform). The second part of our performance model therefore consists of a method to estimate $T(1)$ for an unknown platform. A first attempt to use the CPU clock frequency as a means to relate processors did not provide an accurate model: model predictions deviated greatly from measurements on the available platforms. A more suitable means to relate processors was found in an on-line performance benchmark [17]. This benchmark quantifies the performance of a single core of (multi-core) CPUs. The third and final part of our model consists of a relation between $T(1)$ and s, such that we can estimate s for the new platform from our estimation of $T(1)$ on the new platform that follows from the second part of the model.

Calibration and validation (step 3 in Fig. 1) has been based on measurements on several available homogeneous multi-core platforms. Regression has been used to identify model parameters $T(1)$ and s. The non-linear regression resulted in a good fit, and the sequential fraction s proved to be small; around five percent of the application cannot be parallelized. This sequential fraction involves mainly the parts at the beginning where the bitmaps are divided into bands and at the end where the band results are accumulated. This validates our use of Amdahl's law. Next, we have validated our method to estimate $T(1)$ using the benchmark data and the available platforms. It has proven to be sufficiently accurate for practical usage: the single-threaded data path performance of an unknown platform can be derived from the single-threaded data path performance of a known platform by scaling with the corresponding benchmark values. This has proven accurate within circa 10%. By taking into account additional platform characteristics, an even better accuracy has been achieved. Finally, we have fitted a relation between $T(1)$ and s based on the measurements on the available platforms.

Costs & Benefits Using little effort and simple means, a simple, yet accurate, predictive performance model has been created. The model can be used to estimate the data path performance on many different computational platforms without having to purchase all of them. The model is being used to trade off platform cost and performance to select promising candidate platforms for further investigation (steps 4 and 5 in Fig. 1). Note that when a new platform has been acquired for evaluation, it can be used to further calibrate and validate the model (steps 6 and 3 in Fig. 1) to improve the accuracy.

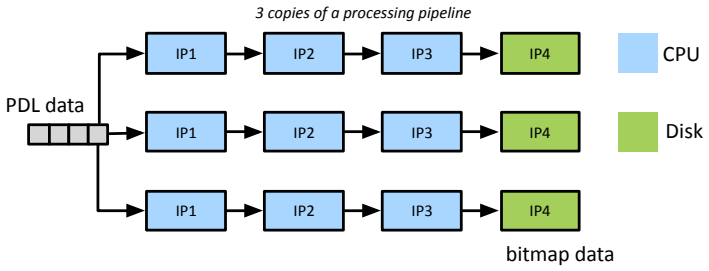

Fig. 4. Three parallel processing pipelines to convert PDL data to bitmap data.

4.3 Optimization of Application Structure for Bitmap Processing

Problem Description We consider a prototype data path for a second type of cut-sheet printer. The data path functionality that is the subject of this case converts Page Description Language (PDL) data (e.g., PostScript) to bitmap images that are suitable for the print engine. This conversion has been implemented in three functional image processing steps IP1 – IP3, and a step, IP4, that writes the bitmap data to a hard disk. Because often there are no data dependencies between PDL items (i.e., pages of the document to print), these steps can be parallelized. Furthermore, the steps allow pipelining. Figure 4 shows a schematic of the situation with three instances of the processing pipeline. The configuration question we face is how many copies of the processing pipeline IP1 – IP4 should be used. There is a trade-off between the memory usage (each pipeline instance statically allocates a significant amount of RAM) and throughput. The latter is hard to estimate based on static calculations because of (i) resource sharing (the steps IP1 – IP3 all are mapped to a 4-core CPU), (ii) pipelining behavior and (iii) the variation in the duration of the processing steps (which heavily depends on the PDL data).

Performance Engineering In the modeling step (step 2 in Fig. 1) we have built a gray-box discrete-event simulation model using the OctoSim tool (see Sect. 3 and [10]). The model consists of a number of dependent tasks (as in Fig. 4), each with a parameter for the nominal execution time. It also includes details about data granularity and buffering. Resource interaction, e.g., when multiple tasks run on the CPU, is modeled by extrapolation of the nominal execution times by the platform model; see [10].

Calibration and validation (step 3 in Fig. 1) has been enabled by code instrumentation (as in Sect. 4.1) and systematic measurements using a small number of representative jobs. This gave us the nominal execution times of the processing steps IP1 – IP4 on the target platform, which have been used for model calibration. In order to validate our OctoSim model, we have measured the throughput of the system for the representative jobs for one, two and three copies of the pipeline. We then have compared these numbers with the predictions of our model (we predict the past). This has shown that the model is quite

accurate given the high level of abstraction: the predictions are all within 10% of the measured values.

Next, we have explored the future (steps 4 – 5 of Fig. 1) by using the model to predict the throughput of a number of fictive jobs and a varying number of pipeline copies. This has shown us that the ratios of the nominal execution times of IP1 – IP3 per PDL data item play an important role. If IP1 – IP3 have a similar duration, then the system does not scale well. In that case IP1, IP2 and IP3 of a single processing pipeline all are active more or less continuously (with different PDL data items). A single pipeline instance therefore already claims three of the four CPU cores. Using workloads derived from practical job sets, we were able to optimize the number of parallel pipelines for the expected use patterns of the printer.

Costs & Benefits We estimate that modeling, calibration and validation, and analysis took approximately one full working week, starting from the basic OctoSim building blocks. The model has allowed us to experiment with different types of input and varying the number of pipeline copies, and this has led to a satisfactory configuration. Being able to avoid experiments on the prototype system has saved a lot of time. Furthermore, the model allowed us to investigate the relation between input characteristics and the throughput in a systematic way, which significantly increased the understanding of the system dynamics. Finding a set of jobs with the required range of properties to test on the prototype would have been very time-consuming, if possible at all.

4.4 Mapping a Computationally Expensive Processing Step

Problem Description In this case we consider a part of a prototype data path that consists of three copies of a pipeline consisting of image processing steps IP5 – IP10. The current prototype implementation does not meet the performance requirements on throughput. Measurements show that the 4-core CPU is heavily loaded. Mapping a computationally expensive step (IP7, which uses three threads) to the GPU might improve the throughput. This, however, is not obvious due to the complex dynamic behavior of the system (pipelining, parallel processing, task interactions on the CPU, etc.). Furthermore, the three copies of the processing pipeline cannot use the GPU in parallel which might create a new bottleneck. The current situation and the design alternative are shown schematically in Fig. 5. Building a prototype of the design alternative would imply that the functionality of IP7 should be re-implemented for the GPU, which would require a lot of engineering effort. Instead of embarking on this directly, we have decided to first employ a model-based analysis.

Performance Engineering We have created, calibrated and validated a gray-box OctoSim discrete-event simulation model for the current situation in a similar way as described in Sect. 4.3 (steps 2 – 3 in Fig. 1). The validation consisted of a manual inspection of the execution trace of the model using the Trace tool, and of a comparison of the measured throughput with the modeled throughput (within 5%). The delta-modeling (step 4 in Fig. 1) adjusted the

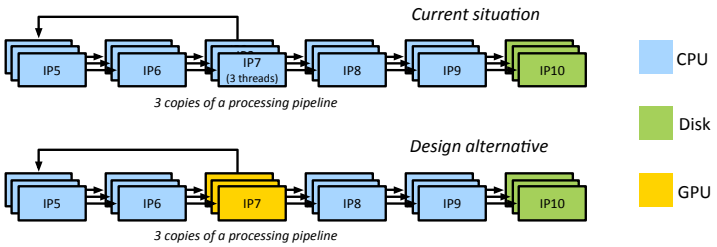

Fig. 5. The current situation and a possible new design which maps IP7 to the GPU.

mapping of IP7 and made some additional minor structural changes. Our design-space exploration (step 5 in Fig. 1) consisted of a systematic analysis of the throughput for a range of durations of IP7, since we have no implementation of IP7 on the GPU and therefore no measurements. This led to the insight that even an efficient implementation of IP7 on the GPU only leads to a small increase in system throughput.

Costs & Benefits An experienced modeler used approximately two working days to create the models from the basic building blocks in the OctoSim library and to analyze them. This includes several meetings with the team that worked on the system to create, calibrate and validate the model. The combination of the model-based analysis results with the estimation of effort for the implementation of IP7 on the GPU led to the decision to not pursue this solution direction. Note that step 6 in Fig. 1 is not possible because of this decision. Performance engineering thus has provided insights that are at the basis of a significant design decision.

4.5 Combining an Application Model with a Real Platform

Problem Description A prototype data path running on a regular desktop PC with a standard operating system suffered a severe performance degradation in some use cases: the throughput could unpredictably drop to less than 10% of the required value. Initial analysis led us to the insight that in these cases the hard disk was the bottleneck. A critical step that reads data from the hard disk became extremely slow. Modeling the behavior of the hard disk under the load that the data path imposed on it, however, would require detailed knowledge of OS and hard disk internals. This information simply was not available to us.

Performance Engineering Instead of creating a model of the application, the mapping and the platform, we created a model of the application and mapping only that we could execute on the real platform (step 2 in Fig. 1). In other words, we created a Java program that mimics the load that the data path imposes on the hard disk. The program has three threads that each write a number of files to the disk at a certain rate, and it has one thread that reads those files from the disk at a certain rate. Configuration parameters of the model are the

file size, the rates, and the delay for the reader (i.e., the reader starts to read the first file after a certain number of files have been written to the disk). The JAVA program can be executed on the physical platform to analyze various scenarios from a performance point of view. Calibration of the application model consisted of configuring the file size, the write and read rates, and the reader delay. These values were extracted from the problematic use case. Manual inspection resulted in the conclusion that the test program exhibited behavior similar to the behavior of the real data path: we predicted the past (step 3 of Fig. 1). Analysis led us to the conclusion that the OS gives reading a lower priority than writing, which has the effect that in high load scenarios reading almost completely stops. We used this qualitatively validated model to experiment with two mechanisms to control the concurrency of the writers and reader in order to make the hard disk performance more predictable (we did not succeed in influencing the OS scheduling in a more direct way; step 4 in Fig. 1). The model predicted that a semaphore that prevents concurrent API calls to the file system from the writer and reader threads solves the problematic behavior: we have explored the future (step 5 in Fig. 1).

Costs & Benefits Creation of the test program cost little effort; JAVA provides ample means for I/O and concurrency. The experiments that the test program allowed us to do would have been tedious using the actual system. The reason is that we then would have needed an appropriate set of jobs that imposes the required load on the disk. It is non-trivial to obtain such a set. The model-based experiments have provided more insight in the dynamic behavior of the hard disk and the implications for the data path performance. Furthermore, they have provided support for the envisioned semaphore solution which requires non-trivial implementation efforts.

5 Conclusion

We have introduced a model-based performance engineering approach that is coupled to the V-model. We have discussed the role of accuracy of model predictions and trust issues that can play a role, especially if the cost-to-validation is significant. Furthermore, we have applied model-based performance engineering with lightweight techniques and tools to five industrial cases from the digital printing domain. Y-chart separation of concerns (application, platform, mapping) and a high level of abstraction ensure simple but accurate models that enable design-space exploration. We have made several observations from these cases.

First, building the models, especially the discrete-event simulation models which require an explicit Y-chart application graph, is a useful exercise in itself. It stimulates *performance thinking* in the team, and documents and clarifies the design in a way that is often lacking. These models (or informal drawings representing the models) are good means for communication and facilitate knowledge transfer between team members. A next step in our process is validation of the constructed performance models to build trust and to quantify predictive accu-

racy. Usually, we take the system as the leading artifact to which the model must conform, but the roles could also be reversed. An example is given in Sect. 4.1 in which the system exhibits behavior that differs from the model prediction. In this case, we regard the regression model as leading and expect a software bug in the system that causes the difference.

Second, as already observed in [3, 20], automation is an important factor for the acceptance of performance engineering. In the five cases that we have described above, the foundational parts of the gray- and white-box models have been relatively easy to obtain, i.e., the regression formulas (Sect. 4.1), Amdahl's law (Sect. 4.2), the application graphs (Sects. 4.3 and 4.4), and the application and mapping (Sect. 4.5). The data that is needed to calibrate and validate the models, however, was much harder to obtain (except for the case in Sect. 4.5). We believe that a systematic way of storing fine-grained performance-related system events will alleviate this. Furthermore, this would enable systematic access to information that can directly be used by architects and engineers to identify and diagnose performance problems, and to get more insight in the system dynamics. For instance, in the cases described in Sects. 4.1 and 4.3 we have added performance logging to the system that enabled us to immediately visualize a Gantt chart of the system's execution after a test run (Fig. 2 is an example of this). Architects and engineers acknowledge that this kind of visualization is very useful. Furthermore, the performance logging output is directly applicable for calibration and validation of performance models.

Third, there are various models for implementing performance engineering. Two extremes are (i) to hire a third party to do performance engineering, and (ii) to change the way of working of the development team to do performance engineering internally. Both have their strengths and weaknesses and both impose different requirements on the tools, processes and, last but not least, the people involved. The five cases have used the former model, and our conclusion is that the cost has been low and we believe that the benefits have been substantial.

To summarize our conclusions, the V-model based performance engineering method, where models are calibrated using existing products or prototypes and then used to explore design alternatives for new products, works in the presented cases from the digital printing domain. Although some of the modeling techniques that we have applied may be regarded as domain-specific, we believe that the process in which they are applied, is not. Other domains which also employ an evolutionary (in contrast to revolutionary) approach with respect to development can use the process of Fig. 1, but may need to apply different, possibly domain-specific, modeling and analysis techniques.

References

1. Amdahl, G.M.: Validity of the single processor approach to achieving large scale computing capabilities. In: AFIPS spring joint computer conference (1967)
2. Balarin, F. et al.: Hardware-Software Co-design of Embedded Systems: The POLIS Approach. Kluwer (1997)

3. Balsamo, S., di Marco, A., Inverardi, P., Simeoni, M.: Model-based performance prediction in software development: a survey. Software Engineering, IEEE Transactions on 30(5) (2004)
4. Beckers, J.M.J., Muller, G.J., Heemels, W.P.H., Bukkems, B.H.M.: Effective industrial modeling for high-tech systems: The example of happy flow. INCOSE International Symposium 17(1) (2007)
5. Bohlin, T.P.: Practical Grey-box Process Identification: Theory and Applications. Springer, Berlin (2015)
6. Eclipse Foundation: Eclipse website (2015), `http://www.eclipse.org/`
7. Embedded Systems Innovation by TNO: Trace website (2015), `http://trace.esi.nl`
8. Fishman, G.S.: Discrete-Event Simulation – Modeling, Programming and Analysis. Springer-Verlag New York (2001)
9. Forsberg, K., Mooz, H.: The relationship of system engineering to the project cycle. INCOSE International Symposium 1(1) (1991)
10. Hendriks, M. et al.: A blueprint for system-level performance modeling of software-intensive embedded systems. International Journal on Software Tools for Technology Transfer (2014)
11. Hendriks, M. et al.: Analyzing execution traces – critical-path analysis and distance analysis. Submitted to STTT (2015)
12. Kruchten, P.B.: The 4+1 view model of architecture. IEEE Software 12(6) (1995)
13. Montgomery, D.C., Peck, E.A., Vining, G.G.: Introduction to linear regression analysis. John Wiley & Sons, Inc., New York, third edn. (2001)
14. Muller, G.J.: CAFCR: A Multi-view Method for Embedded Systems Architecting; Balancing Genericity and Specificity. Ph.D. thesis, Delft University of Technology (2004)
15. Mussbacher, G. et al.: The relevance of model-driven engineering thirty years from now. In: Model-Driven Engineering Languages and Systems, Lecture Notes in Computer Science, vol. 8767. Springer (2014)
16. Parappurath, V.V., Voeten, J.P.M., Kotterink, K.C.: Calibration error bound estimation in performance modeling. In: Euromicro Conference on Digital System Design (2013)
17. PassMark Software: CPU Mark - Single Thread Performance (2015), `https://www.cpubenchmark.net/singleThread.html`
18. Pimentel, A.D., Thompson, M., Polstra, S., Erbas, C.: Calibration of abstract performance models for system-level design space exploration. Journal of Signal Processing Systems 50(2) (2008)
19. Smith, C.U., Williams, L.G.: Software performance engineering. In: UML for Real. Springer US (2003)
20. Voeten, J. et al.: Predicting timing performance of advanced mechatronics control systems. In: IEEE 35th Annual Computer Software and Applications Conference Workshops (COMPSACW) (2011)
21. Westland, J.C.: The cost of errors in software development: evidence from industry. Journal of Systems and Software 62 (2002)
22. Williams, L.G., Smith, C.U.: Making the business case for software performance engineering. In: 29th International Computer Measurement Group Conference. pp. 349–358. Computer Measurement Group (2003)