Quipu: A Statistical Model for Predicting Hardware Resources

ROEL MEEUWS, S. ARASH OSTADZADEH, CARLO GALUZZI, VLAD MIHAI SIMA, RAZVAN NANE and KOEN BERTELS, Delft University of Technology

There has been a steady increase in the utilization of heterogeneous architectures to tackle the growing need for computing performance and low-power systems. The execution of computation-intensive functions on specialized hardware enables to achieve substantial speedups and power savings. However, with a large legacy code base and software engineering experts, it is not at all obvious how to easily utilize these new architectures. As a result, there is a need for comprehensive tool support to bridge the knowledge gap of many engineers as well as to retarget legacy code. In this article, we present the *Quipu* modeling approach, which consists of a set of tools and a modeling methodology that can generate hardware estimation models, which provide valuable information for developers. This information helps to focus their efforts, to partition their application, and to select the right heterogeneous components. We present *Quipu*'s capability to generate domain-specific models, that are up to several times more accurate within their particular domain (error: 4.6%) as compared to domain-agnostic models (error: 23%). Finally, we show how *Quipu* can generate models for a new toolchain and platform within a few days.

Categories and Subject Descriptors: C.1.3 [**Processor Architectures**]: Other Architecture Styles—*Heterogeneous (Hybrid) systems*; C.4 [**Performance of Systems**]: Modeling Techniques

General Terms: Measurement, Performance, Design

Additional Key Words and Phrases: Reconfigurable architectures, modeling, estimation, statistics, software complexity metrics, system analysis and design

ACM Reference Format:
Meeuws, R., Ostadzadeh, S. A., Galuzzi, C., Sima, V. M., Nane, R. and Bertels, K. 2013. Quipu: A statistical model for predicting hardware resources. *ACM Trans. Reconfig. Technol. Syst.* 6, 1, Article 3 (May 2013), 25 pages.
DOI: http://dx.doi.org/10.1145/2457443.2457446

1. INTRODUCTION

With the ever growing need for more computing performance and lower power consumption, manufacturers have traditionally relied on technology scaling. However, with the end of Moore's law in sight [Rupp and Selberherr 2011], there has been a steady increase in the utilization of parallel and heterogeneous architectures. There are already many examples of architectures, which incorporate GPUs, ASICs, FPGAs, and DSPs to accelerate applications [Bertels et al. 2010; Chen et al. 2007; nVidia Corp. 2011]. By executing computation-intensive functions on such specialized architectures, it is possible to achieve substantial and power-efficient performance improvements without the need of technology scaling. In this respect, reconfigurable architectures are gaining in popularity, as they allow for substantial application

This research is partially supported by the Artemisia iFEST project (grant 100203), the Artemisia SMECY project (grant 100230), and the FP7 Reflect project (grant 248976).
Authors' address: R. Meeuws, S. A. Ostadzadeh, C. Galuzzi, V. M. Sima, R. Nane, and K. Bertels, Computer Science and Engineering Department, Delft University of Technology; email: r.j.meeuws@tudelft.nl.

DOI: http://dx.doi.org/10.1145/2457443.2457446

speedups at low power costs. At the same time, they retain the necessary flexibility to adapt to changing application requirements. This flexibility becomes even more visible when the reconfigurable fabric is integrated in a microprocessor architecture to easily incorporate and execute specialized accelerators. Recent architectures such as the Xilinx Zynq [Xilinx 2011] and the Altera Cyclone V [Altera 2011] already offer this kind of facility.

Developing applications for these platforms requires both traditional software engineering expertise and specific knowledge of programming reconfigurable fabrics. This forces companies to move from pure software or hardware development to a combined hardware/software codesign approach. However, with a limited expertise in hardware/software codesign, companies struggle to harness the potential of these new architectures. Some companies utilize SystemC to tackle this problem, but many others rely on large bases of legacy code in High Level Languages (HLLs), such as C, which prevent them from taking this step. As a result, there is a clear demand for comprehensive tool support to bridge the knowledge gap of engineers and to retarget existing code to such platforms. Part of that demand is being addressed by C-to-HDL compilers, such as Catapult C [Morris 2004], ROCCC [Villarreal et al. 2010], or DWARV [Yankova et al. 2007]. Notwithstanding, developers have to address many other issues in a short time frame in order to meet time-to-market pressure. These include, for example, the identification of resource-intensive parts of the application, the evaluation of different architectures and mapping options, and the estimation of project costs. It is in these aspects that many toolchains are lacking the necessary tool support.

In order to handle these issues, developers need information on different factors, such as the power consumption, the speedup, the communication requirements, or the hardware resource consumption of the intensive parts of an application on different processing elements. The designer may determine the necessary information by creating, for each kernel,[1] all possible implementations. However, the design space can be huge and evaluating all design alternatives can be exceedingly time-consuming. Especially in the presence of reconfigurable hardware, where many different kernels need to be synthesized, searching this design space may take up to several months. Even more, during the development process, the application may still change many times and this process would have to be repeated accordingly.

In order to solve this problem, we require tools that can quickly characterize these important features when only a software implementation is available. This way, the time needed to evaluate different heterogeneous processing elements can be reduced by several orders of magnitude. The Q^2 profiling framework [Bertels et al. 2011; Ostadzadeh et al. 2012] is one such set of tools. The main goal of this framework is to enable efficient application mapping by quantifying these important hardware features. Q^2 provides the necessary information to maximize the potential performance increase, while taking into account communication bottlenecks and resource constraints. In order to provide resource estimates for various toolchains and platforms, Q^2 provides a generic approach for building prediction models, called the *Quipu* Modeling Approach [Meeuws et al. 2006, 2007, 2008, 2011, 2012]. *Quipu* is a quantitative prediction modeling approach for early Design Space Exploration (DSE). *Quipu* models are able to predict the important hardware aspects of kernels to be mapped to reconfigurable components. They take a HLL description (C code) as input and estimate area, interconnect, static power, clock period, and other FPGA-related measures for a particular combination of a platform and a toolchain. The *Quipu* modeling approach is not restricted to any platform or toolchain and appropriate *Quipu*

[1]In this article, we define a *kernel* as a single function in a HLL description.

models can be generated in different contexts. By using linear prediction models based on Software Complexity Metrics (SCMs), the time required by Quipu prediction models to determine estimates becomes several orders of magnitude smaller than the time-consuming process of hardware synthesis required to obtain the final results. In particular, we achieve a speed of 87.4 predictions per second, as opposed to the much more costly process of hardware synthesis. Consequently, developers can quickly identify the resource-intensive parts of their application, evaluate the effect of changes on the cost of the final design, or select the right processing elements for their application. This results in a time saving of hours or even days per design iteration. Furthermore, resource estimates have a crucial role in optimizations such as loop-unrolling, automatic parallelization, or recursive variable expansion, because of the limited resources that are available.

In this article, we present a high level quantitative prediction modeling scheme that generates prediction models for different toolchains, different platforms, and different application domains. *Quipu* generates models that accurately capture the relation between hardware and software metrics, based on linear regression, neural networks, and other statistical techniques. The main contributions presented in this article are the following:

— a robust statistical modeling methodology based on a large set of 324 kernels from 66 real applications from different application domains;
— a modeling methodology, which allows the generation of prediction models targeting different application domains; these domain-specific models exhibit an increased accuracy compared to domain-agnostic models;
— fully operational instances of *Quipu* prediction models targeting four different combinations of toolchains and platforms, including ASICs, with prediction errors ranging from 20% upto 39%;
— a comprehensive description of the generation of *Quipu* prediction models for a specific combination of a platform and a toolchain in a few days;
— a benchmark for the C-to-HDL generation capabilities for different C-to-HDL compilers using the *Quipu* kernel library.

The remainder of this article is structured as follows. In Section 2, we review the existing work related to our approach and we establish the added value of the *Quipu* modeling approach. In Section 3, we give an overview of the theory and the components of our approach. We evaluate the *Quipu* modeling approach in Section 4. Specifically, we evaluate the error behavior of our models for four different toolchains and three separate application domains. Furthermore, we show how *Quipu* models can be retargeted and utilized for a particular combination of a toolchain and a platform. Finally, Section 5 concludes the article and proposes directions for future work.

2. RELATED WORK

Many approaches for hardware performance estimation have been proposed over the last years. Some schemes aim to drive the final phases of hardware synthesis, such as mapping, and place and route. More recently, the focus has moved to making predictions from HLL descriptions, such as ANSI-C or MATLAB. All these approaches, however, have their particular shortcomings and restrictions. Some approaches are tightly coupled to specific platforms and toolchains or only support Hardware Description Languages (HDLs), whereas others focus on specific parts of the design or only generate models for a particular design. In this section, we summarize the main approaches in the field of hardware estimation. We investigate what part of a design they estimate, if they are dependent on a particular tool or platform, what hardware

Table I. Overview of the Main (Hardware) Resource Estimation Approaches

	Reference	Object	Target	Predictions	Input
Other	[Enzler et al. 2000]	Entire design	Generic	Area, Frequency	DFG (RTL)
	[Lakshmi et al. 2011]	Entire design	Xilinx	Power	FPGA netlist
	[Schumacher et al. 2008]	Entire design	Xilinx	Area, Delay, Power	(V)HDL
	[Nayak et al. 2002]	Entire design	MATCH	Area	(V)HDL
	[Brandolese et al. 2004]	Entire design	SystemC	Area	SystemC
	[Bilavarn et al. 2006]	Entire design	Design-Trotter	Area	HCDFG
	[Deng et al. 2008]	IP core specific	TANOR	Area, Power	HLL (MATLAB)
	[Chuong et al. 2009]	Controller	Trimaran	Area, Delay	HLL (C)
LR/HLL(C)	[So et al. 2003]	Loop nests	DEFACTO	Area	HLL (C)
	[Holzer and Rupp 2005]	Entire design	SPARK	Delay	HLL (C)
	[Degryse et al. 2008]	Loop controller	CLoogVHDL	Area, Frequency	HLL (C)
	[Kulkarni et al. 2006]	Entire design	SA-C	Area	HLL (C)
	[Cilardo et al. 2010]	Entire design	Generic[a]	Area	HLL (C)
	Quipu	**Entire design**	**Generic**	**Area, Latency**	**HLL (C)**

[a] Only Impulse-C demonstrated.

criteria they predict, and what input specification they require. A summary of the main characteristics of each approach is presented in Table I.

For example, Chuong et al. [2009] presented an area-time estimation scheme for the control of a design only, disregarding the datapath. The reported error was 3.8% for flip-flops and 10.3% for slices. The prediction errors were validated using a set of only 10 kernels. In addition, this work did not provide predictions for the whole design and strictly targeted the Trimaran compiler.

Industry has also come forward with efforts to provide hardware estimates in the early stages of the design process. As an example, Schumacher et al. [2008, 2011] presented a resource estimation algorithm from a HDL description. This method mimics the actual synthesis toolchain. The reported errors for flip-flops and slices were 14.2% and 21.9%, respectively. The authors validated their model using a set of 90 VHDL descriptions.

Enzler et al. [2000] presented an estimation scheme targeting tasks that lack extensive control structures. It is based on a set of custom formulas for area and latency. The authors attempted to validate their approach using only 6 kernels. Errors of approximately 12% for area and 29% for frequency were reported.

Deng et al. [2008] presented a methodology for area and power prediction. Similarly to our work, they based their approach on Linear Regression (LR). The methodology generates estimation models that are tailored to characterize different parameters in specific IP cores. In that sense, these models can not be used for different designs or kernels. The authors validated their models with a dataset of just 12 configurations from only 2 IP cores. They reported an error of 8.2% for the number of slices and 7.0% for power.

Similarly, Lakshmi et al. [2011] presented a statistical power modeling approach that provides a generic model to be used as IP-Core macro-model. The prediction models are based on detailed resource usage information and on port activity numbers. The authors used a set of 13 IP cores to build their model. At least 6 of those cores were used as training set and at least one was used as validation set. They reported an error of 4%. Within the domain of SystemC, Brandolese et al. [2004] presented another estimation approach using statistical methods. This approach is based on the design parameters of the SystemC and VHDL descriptions. It was generated from a dataset

of 20 designs using classic LR. Nevertheless, the results were validated with only 5 designs. The reported error for the number of LUTs was 36.8%.

Similarly, Nayak et al. [2002] presented an area estimator to enable automatic DSE within the MATCH compiler [Banerjee et al. 2000], which is based on a classic LR taking into account the number of operations of each type, their bit-widths, and the number of registers as determined from VHDL generated from MATLAB. The authors failed to mention how their estimation formula was determined. In contrast to our approach, they validated their estimator with only 7 kernels, reporting an estimation error of 16%.

In Bilavarn et al. [2006], an estimation methodology targeting HLLs in the context of the Design-Trotter project was presented. The estimation scheme is based on a specific architectural template, where each component is estimated separately. A library of functional components was utilized to characterize different nodes in the Hierarchical Control and Data Flow Graph (HCDFG). The authors reported an error of 20% for area estimates using only 22 kernels from the DSP domain.

Up to this point, all discussed related approaches contained custom expressions or algorithms to derive hardware estimates. In addition, these approaches were all performed in the context of specific platforms or toolchains. In our work, instead, we focus on using statistical analysis to obtain the necessary hardware prediction models for different combinations of tools and platforms. Furthermore, although Nayak et al. [2002], Brandolese et al. [2004], Bilavarn et al. [2006], Deng et al. [2008], and Chuong et al. [2009] targeted HLLs, they did not support ANSI-C. Therefore, in the following, we narrow our focus on several approaches that deal with both statistics and ANSI-C.

Holzer and Rupp [2005] presented a custom model for execution time estimation based on the SPARK C-to-VHDL compiler [Gupta et al. 2003]. They briefly mentioned SCMs, measures that characterize software descriptions, although they only use a critical path estimation scheme based on Control Flow Graphs (CFGs). They did not present a statistical analysis of their results. Based on only 9 kernels from 2 applications, they reported an error between 39.3% and 44.4%. Although this work targeted HLL descriptions, it was dependent on SPARK and presented only a very limited validation set, contrary to our work.

Kulkarni et al. [2006] presented an approach for FPGA area estimation based on building prediction models for each type of DFG node. These prediction models were generated by performing LR on a set of DFG nodes with changing characteristics, resulting in a set of model coefficients for that node. The authors reported an error of 5.3% for area based on only 4 kernels from the image processing domain. The estimation approach proposed by Kulkarni et al. is tailored to the SA-C intrinsics and, as such, can not be easily recalibrated for a different combination of tools and platforms.

An example approach that focuses on a specific kind of design was presented in Degryse et al. [2008]. Based on the Polyhedral model, the authors presented a prediction scheme that targets loop controllers. The authors determined the effect of the quantity of statements and the loop nesting depth on the utilized hardware generation methodology. They reported an error of 7.14% using a validation set of only 12 kernels for their LR model.

So et al. [2003] presented an estimation strategy for their DEFACTO C-to-VHDL compiler, which helps to determine the unroll factor. By using a LR model based on High Level Synthesis (HLS) to predict the post-place and route area, they circumvented the expensive low-level synthesis passes. The approach aims to reduce the time to evaluate different design alternatives. The model was evaluated using a set of 209

alternatives generated from only 5 multimedia kernels, but no prediction error was reported.

As listed in Table VI, each of these works has claims on the quality of their models in terms of error based on small validation sets containing 3 to 12 observations. However, if the goal is to report errors that are not biased to a small dataset, a larger set of validation data, such as the one used in Quipu, becomes vital. Let us suppose, for example, we validate a model using a set of only 12 observations. It is very unlikely that these observations can represent the entire spectrum of kernels. For one, with such a limited set of kernels, the possibility of *cherry-picking* the validation set increases. Secondly, an anomalous kernel that is not well modeled by a certain model can adversely affect the validation error, when a small set of kernels is used. This is especially the case for kernels from different domains. For approaches that utilize larger datasets for validation, we refer the reader to, for example, the ones in Monostori et al. [2005] or Palermo et al. [2009], targeting FIR filters and multiprocessor systems, respectively. However, as these two works are not in the context of hardware estimation for reconfigurable components, we do not further discuss them here.

Finally, in Cilardo et al. [2010], we find a strategy that is *seemingly* very close to our approach. In correspondence with our earlier work [Meeuws 2007; Meeuws et al. 2007, 2008], they used a kernel library, a HLL-to-HDL compiler, and a metrication tool, with the addition of some specific counts of operators for each bit-width. Additionally, they also used LR to obtain prediction models, as in our approach. They validated their results with a set of 3 kernels and reported an error of 31% for area. The main differences with the *Quipu* modeling approach presented in this article, apart from the lower accuracy, are the following.

— Cilardo et al. [2010] utilized a kernel library of approximately 200 synthetic kernels to generate their prediction models, while we employ a library of 324 real kernels from 66 applications, of which up to 266 were generated into HDL. This, in turn, dramatically increases the applicability of our models for real applications.
— We validate our model using 10-fold cross-validation on 266 HDL descriptions that were generated from our kernel library in order to obtain a higher accuracy for the reported error, in contrast to the *very small* validation set utilized by Cilardo et al. [2010].
— the *Quipu* modeling approach is able to generate domain-specific models by using real kernels from different domains.

A careful analysis of all these works shows that our approach uniquely addresses the problem of early generic quantitative hardware prediction using statistical methods targeting ANSI-C. Additionally, the validation of our approach uses a substantial real-life dataset to provide acceptable accuracy. To the best of our knowledge, no other approach exists in this particular niche.

In this article, we present the *Quipu* modeling approach for the prediction of hardware resource consumption targeting reconfigurable components. This method targets HLLs such as C, in order to drive early DSE and to provide models for different reconfigurable platforms and toolchains. Contrary to existing approaches, we validate our approach with up to 266 real kernels from our library using models produced by our modeling approach. We validate the general applicability of our approach by providing models for three other combinations of tools and platforms and we show how *Quipu* can be retargeted to one of these toolchains within a few days. Furthermore, we provide domain-specific models that exhibit a higher accuracy compared to domain-agnostic models, because kernels in the same domain tend to be similar to each other. Such similarity has been observed before with regard to dynamic behavior in, for instance, Eeckhout et al. [2002] and Cammarota et al. [2011], but we will show this

also holds true with regard to static elements, such as hardware generated by HLS, in Section 4.5.

3. THE *QUIPU* MODELING APPROACH

The work presented in this article is in the context of *Quipu* [Meeuws 2007; Meeuws et al. 2007, 2008, 2011], a quantitative prediction modeling approach for early DSE. This statistical modeling approach generates prediction models that are able to predict the important hardware aspects of kernels to be mapped to reconfigurable components. This is especially useful in the context of early DSE, where a huge design space needs to be reduced as fast as possible. The models take an ANSI-C description as input and estimate various FPGA-related measures, such as area, frequency, or latency. Despite the fact that *Quipu* estimates are less accurate compared to the actual synthesis results or low level estimates, these estimates provide all the necessary insight into hardware costs at an early stage without the need for time-consuming synthesis or compilation. As a result, *Quipu* estimates can be used effectively within iterative algorithms for partitioning and mapping, where estimates need to be made for many design alternatives in a short period of time. These estimates are also valuable in determining sensible optimization parameters, such as the loop-unroll factor, where area constraints play an important role.

Apart from the clear advantages of fast and early estimations, our approach is able to generate models for different tools and platforms, which provides a high level of retargetability. This is important for companies that want to retain a high level of flexibility in choosing design alternatives, while retaining a comparable level of prediction quality. To clearly show this advantage, this article presents *Quipu* prediction models for four different combinations of platforms and toolchains. Furthermore, we present an extensive case study on how to retarget our *Quipu* modeling approach to a new toolchain and platform in just a few days.

In the following, we describe the different parts of the *Quipu* modeling approach. More precisely, in Section 3.1, we introduce the Q^2 profiling framework, which makes heavy use of our approach and is utilized for validating our work. In Section 3.2, we present the use cases of our approach and describe the methodology for calibrating models for a new toolchain and/or platform. Afterwards, we describe the components of the modeling methodology in detail. In Section 3.3, we describe SCMs as the main criteria in our approach. Subsequently, in Section 3.4, we present the necessary modeling theory employed by *Quipu*. In Section 3.5, we discuss different ways to evaluate our prediction models. Finally, in Section 3.6, we describe the Kernel Library and tools that implement our approach.

3.1. Q^2 Profiling Framework

The *Quipu* modeling approach is an essential part of the Q^2 profiling framework [Bertels et al. 2011; Ostadzadeh et al. 2012], which is depicted in Figure 1(a). The main focus of this framework is to provide essential profiling information in order to drive efficient mapping of applications onto heterogeneous reconfigurable systems. Q^2 is part of the *Delft Workbench* (*DWB*) [Bertels et al. 2006], a semi-automatic tool platform for integrated HW/SW codesign, targeting heterogeneous and reconfigurable computing systems. The two main concerns of the profiling framework are to reveal the data communication that occurs inside the application and to estimate reconfigurable resource consumption for each part of the application. Ultimately, the profiling information helps to efficiently partition the application into hardware and software. *Quipu* mainly addresses the second concern by providing the necessary hardware resource estimates early in the design process.

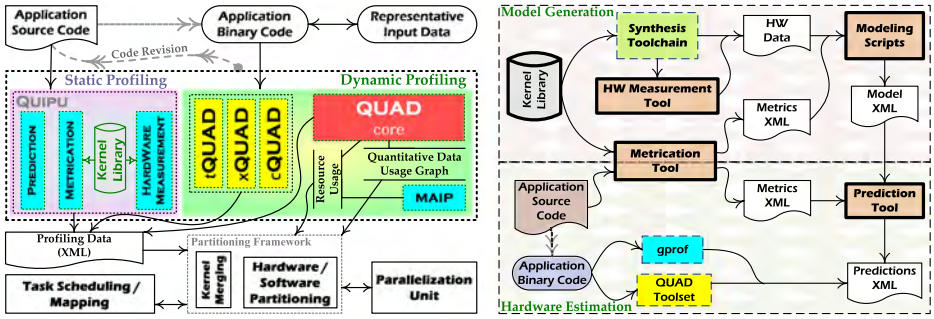


Fig. 1. (a) An overview of the Q^2 profiling framework and (b) an overview of the *Quipu* modeling approach.

3.2. Modeling Methodology

In Figure 1(b), an overview of the *Quipu* modeling approach is depicted. The figure is divided into two parts that correspond to the two main use cases of the *Quipu* modeling approach. These two use cases have different objectives, although they are related to each other.

(1) *Model Generation.* This use case is mainly utilized when *Quipu* models need to be generated for a certain toolchain and/or platform; for example, when a new version of the synthesis toolchain is released. This process can be time-consuming, but it is executed only once for a particular combination of a tool and a platform. A complete walkthrough of the generation of a fully operational *Quipu* model is presented in Section 4.2.
(2) *Hardware Estimation*. This use case is employed to provide fast and early estimates during partitioning or even during algorithm development. As such, it is utilized far more often than *Model Generation*. In Section 4.6, we show how *Quipu* prediction models can be used in a real setting.

In *Model Generation*, a *Quipu* model can be calibrated using the output data from each specific combination of a toolchain and a platform. As many toolchains have a variety of optimization and effort level options, it may be necessary to generate different sets of calibration data for some of the main options. The modeling methodology for the creation of new *Quipu* models consists of the following steps.

Step 1. Kernel library customization. For a new toolchain, the HDL generation scripts in the kernel library need to be adjusted accordingly. Furthermore, it may be necessary to provide the kernels in lists, or to mark them in a different way, depending on the targeted toolchain.

Step 2. HDL generation. After the scripts are in place, the HDL should be generated for each kernel. It may happen that some kernels are incompatible with the targeted tool. In that case, the developer may choose to make the necessary changes or to ignore the incompatible kernels.

Step 3. HDL synthesis. Using the generated HDL files, the synthesis toolchain should generate the hardware for each kernel. For this purpose, the synthesis scripts need to be adjusted for the targeted toolchain. Some HDL files may not be synthesizable and, as such, may be discarded. Instead, the developer might go back to *Step 2* to resolve this issue.

Table II. Overview of the SCMs Currently Employed by the *Quipu* Modeling Approach

Category	Some metrics	Number
Basic Blocks	Number of basic blocks, max. and avg. number of operators, statements, loads, and loads from the heap.	9
Execution estimates	Loop iterations are accounted for, where possible, in a separate number of operators, and number of statements.	2
Memory	Number of loads and stores, number of loads and stores to the heap. The total amount of bits is counted separately.	8
Bitwise	Number of bitlevel operations (XOR, AND, etc.). The total amount of bits is counted separately.	6
Floating Point	Number of additions/subtractions, divisions, multiplications, modulo's, square roots, trigonometric operations, etc. The total amount of bits is counted separately.	18
Integer	Number of additions/subtractions, divisions, multiplications, modulo's, etc. The total amount of bits is counted separately.	20
Variables etc.	Number of parameters, constants, variables, globals, pointers. The total amount of bits is counted separately.	9
Nesting	Nesting level, cumulative and average nesting level [Meeuws et al. 2007], and the number of loops.	4
Other	Number of statements, function calls, logical operators, type conversions, bits in type conversions, and the presence of a return value.	6
Halstead	Number of (unique) operators and operands [Halstead 1977].	4
From literature	AICC [Harrison 1992], scope number and scope ratio [Harrison and Magel 1981], McCabe's Cyclomatic Number [McCabe 1976], Oviedo's Def-Use pairs [Oviedo 1980], Elshoff's Data Flow [Elshoff 1984].	6
Total		92

Step 4. Data extraction. With a sufficiently large set of synthesized kernels, the *Quipu* Metrication and Hardware Measurement tools should now gather the necessary SCMs and hardware performance data. It may be necessary to adjust the Hardware Measurement tool for the new toolchain.

Step 5. Statistical modeling. After the data becomes available, the semi-automatic modeling scripts can be employed to generate the final prediction models. Although the scripts are able to automatically generate appropriate models in most cases, manual intervention can further improve the quality of these models.

3.3. Software Complexity Metrics

As our approach is based on statistical analysis, it is essential to obtain an independent dataset of measurements in order to generate realistic and accurate prediction models. As our starting point is ANSI-C, it is essential to quantify the characteristic aspects of the software description at hand. Previously, in Meeuws [2007] and Meeuws et al. [2007], we have introduced SCMs as suitable measures characterizing software descriptions. SCMs are indicators of different aspects of the source code under consideration. Currently, we use a set of 92 different SCMs, as listed in Table II. Most of the SCMs in our model are simple counts of different operations available in ANSI-C. In addition, we also implemented several SCMs related to Software Measurement, such as the Cyclomatic complexity, Elshoff's data complexity, and others. A drawback of using SCMs for statistical modeling is the inherent multicollinearity, which degrades the modeling process, because it can seem that important variables are not significant [Munson 2002]. We describe some solutions to this problem in Section 3.4.3. For more details on the SCMs used in our approach, we refer the interested reader to the work we presented in Meeuws [2012, Chap. 3].

3.4. Regression Analysis

The statistical techniques that are employed by *Quipu* are an essential part of our modeling approach. In the following, we introduce some of the key aspects of our analysis,

starting with some basic modeling terminology and going on to the different regression techniques that are employed.

3.4.1. Model Definition. Given the SCMs mentioned in Section 3.3, let us define the prediction model that we need to determine. As a first step, we consider the following relation between hardware and software:

$$y_{HW} = g(\vec{\mathbf{x}}_{\mathbf{SCM}}) + \epsilon. \tag{1}$$

This is the theoretical optimal model relating some hardware metric y_{HW} to the SCMs $\vec{\mathbf{x}}_{\mathbf{SCM}}$[2] with the ideal relation $g(\cdot)$ and some error ϵ inherent to the problem at hand. In practice, an optimal model can not be found. Instead, any modeling scheme is an approximation to some extent. Therefore, we model the relation $g(\cdot)$ with an approximated relation $\hat{g}(\cdot)$[3]. This results in the introduction of some error $\hat{\epsilon}$ inherent to our approximation scheme. The approximation $\hat{g}(\cdot)$ can be, for example, an ad-hoc model, a LR model, or an Artificial Neural Network (ANN). In case of LR techniques, $\hat{g}(\cdot)$ is a linear equation. We can express this approximation as follows:

$$\hat{y}_{HW} = \hat{g}(\vec{\mathbf{x}}_{\mathbf{SCM}}) + \hat{\epsilon} = \hat{\vec{\mathbf{a}}}\vec{\mathbf{x}}_{\mathbf{SCM}} + \hat{b} + \hat{\epsilon}, \tag{2}$$

where $\hat{\vec{\mathbf{a}}}$ is a vector of coefficients $\hat{a}_i$ corresponding to the element x_i of the set of SCMs $\vec{\mathbf{x}}_{\mathbf{SCM}}$ obtained for a certain kernel, which correspond to the hardware metric $\hat{y}_{HW}$, and $\hat{b}$ is the offset of the linear model. Note that these variables are stochastic variables. This means that reporting a simple percentage error is not enough. To be more precise, the characterization of the error distribution must be addressed as well.

In this respect, LR can be seen as a technique used to solve a set of linear equations. Traditionally, this linear system has been represented as follows:

$$\vec{\mathbf{y}} = \mathbf{X}\vec{\beta} + \vec{\epsilon}, \tag{3}$$

where, $\vec{\mathbf{y}}$ is the *observation vector*, $\mathbf{X}$ is the *design matrix*, $\vec{\beta}$ is the *parameter vector*, and $\vec{\epsilon}$ is the *residual error vector*. The observation vector corresponds to the hardware measures, which are called dependent variables. The design matrix holds the so-called independent variables, which consist of the SCMs described earlier. The regression analysis comprises the determination of the model parameters in the parameter vector $\vec{\beta}$. This will also yield the residual error. An exact solution for the model parameters will most probably not be found. As such, we determine the residual error vector $\vec{\epsilon}$, by estimating the model parameter vector $\vec{\beta}$ and calculating the difference between $\vec{\mathbf{y}}$ and $\mathbf{X}\vec{\beta}$. There are various techniques to estimate these model parameters. Some of the techniques used in our approach are discussed in the following sections.

3.4.2. Generalized Linear Model (GLM). The assumption that a certain dataset is normally distributed does not hold in many cases. As such, there is a need to support different data distributions. A well-known approach to support more general datasets is the GLM. This is an approach that has two main features that help in regressing nonnormal data: support for the exponential family of distributions, and the notion of a link function. There are many distributions supported, such as the normal distribution, the Poisson distribution, and many more. In addition to relaxing the requirement of normally distributed errors, the GLM also introduces a link function, which describes how the response variable and the independent variables are related. To a certain extent, this is comparable to the transformation of the response variable.

[2] $\vec{x}$ is the standard notation for a vector.

[3] $\hat{g}(\cdot)$ represents an approximation of the relation $g(\cdot)$.

As an example, consider *Logistic Regression* (LogR), where a binary output variable is modeled. This is accomplished by actually regressing the odds for a `true` or `false` value, instead of regressing their integer values 1 and 0, respectively. Such a variable is characterized by the binomial distribution. Let us assume that y_i is binomially distributed as $B(n_i, p_i)$, that is, there exists a bound set of values n_i. Then, LogR models the logistic transformation (logit$(\cdot)$) of the probability p_i for y_i to be one as follows:

$$\text{logit}(p_i) = \log\left(\frac{p_i}{1-p_i}\right) = \hat{g}(\vec{\mathbf{x}}_{\text{SCM}}) + \hat{\epsilon}. \tag{4}$$

This logistic function of the probability of encountering a value of `true` for y_i is assumed to be linearly dependent on the SCM. By using the inverse logit, we can predict the probability of each possible value. In this example, GLM can be employed using the binomial distribution with the logit link function.

3.4.3. Collinearity. The collinearity between the different SCMs in our model poses a problem. Due to the collinearity, some SCMs measure more or less the same aspect of the code. This is problematic for regression analysis as certain aspects are now overrepresented. As a result, other potentially important variables can be marked as nonsignificant, degrading the accuracy of the model. We have utilized several well-known techniques to address this problem so far. These include Principal Component Analysis (PCA) [Meeuws et al. 2007], Partial Least Squares Regression (PLSR) [Meeuws et al. 2008], and Stepwise Model Selection (SMS) [Meeuws et al. 2011]. Apart from these automatic approaches, it is also possible to provide a manually selected subset of the SCMs.

In PCA, an *orthogonal set of principal components* minimize the covariance among the different dependent variables. In this way, the collinearity problem is reduced. PLSR is an adaptation of this technique, where the components are selected in such a way that they explain as much variance in the response variable as possible. Another common approach to reduce the number of predictors (SCMs) is SMS. In this work, we utilize the Bidirectional elimination approach employed in [Venables and Ripley 2002]. Starting from a preliminary model, this method successively adds and removes SCMs step-by-step. At each step, the significance of each SCM, given the current model instance, is calculated and the most significant variable is added to the model. After that, any variable that can be removed without increasing the error is removed. This procedure continues as long as there are possible steps that can improve some quality criterion.

3.4.4. Nonlinearity. During our analysis, we found out that in addition to the collinearity problem, many SCMs do not have a clear linear relation with the predicted hardware characteristics. Consequently, it is useful to transform the SCMs, where possible, to better relate them to the hardware characteristics, or to use nonlinear regression techniques. The *Box-Cox power transformation* [Box and Cox 1964] is a widely used data transformation. This transformation preprocesses the data in order to reduce the variance of the dataset, as well as to make the sample distribution more similar to the normal distribution. The transformation is defined as follows:

$$x_i^{(\lambda)} = \begin{cases} \frac{x_i^{\lambda}-1}{\lambda} & \text{for } \lambda \neq 0 \\ \log(x_i) & \text{for } \lambda = 0 \end{cases}, \tag{5}$$

where x_i are the observations of the independent variables, λ is the estimator used to determine the transformation, and $x_i^{(\lambda)}$ are the transformed observations. By using

a Box-Cox estimator in the *R statistical computing environment* [Faraway 2006], we determined that λ is close to zero for many predictors. Therefore, we considered log-transformations for all problematic SCMs in our model selection procedure. The transformed SCMs are considered in all different modeling techniques employed in this article.

Apart from data transformations, there are several techniques that tackle nonlinearity. The most notable technique employed by the *Quipu* modeling approach is the use of Artificial Neural Networks (ANNs) [Callan 1998]. An ANN is composed of a set of nodes that are arranged in *layers*. Each layer uses the outputs from the previous layers as inputs. Each node is a weighted linear combination of inputs. The value of this combination is transformed using an activation function passed on to the next layer:

$$\text{net}_j = f(\sum_{i=0}^{n-1} x_i w_{ij}), \tag{6}$$

where net_j is the output of node j, $f(\cdot)$ is the activation function, x_i ($i \in 0, \ldots, n-1$) are the outputs of n nodes in the previous layers, and w_{ij} is the weight for the input from node i to node j. We use the ANN training package *nnet* from the *R* statistical computing environment. This package trains networks with a single hidden layer using the Broyden-Fletcher-Goldfarb-Shanno (BFGS) training algorithm [Nocedal and Wright 2000]. During the development of our approach, other modeling techniques, such as random forests, ridge regression, and multivariate adaptive regression splines, have also been considered. However, these techniques did not consistently outperform neural networks when using our kernel library, but did require a substantial amount of additional computation time. To this purpose, we limit ourselves, in this article, to the use of GLM, PCR, PLSR, SMS, Box-Cox, and ANNs.

3.4.5. Sparse Data and Other Issues. The dataset of SCM and hardware characteristics may contain many zero observations. For example, the number of floating point operations is zero for all integer-only kernels. When we consider *sparse* data in the independent variables, linear regression may be adversely affected by the disbalance in the data, in extreme cases. Because the few nonzero observations in such extreme cases do not provide sufficient information to have a meaningful effect in regression, an easy solution is to omit those independent variables from the model. We utilize a zero-threshold that lets *Quipu* omit those regressors that have less than a certain amount of nonzero entries.

In the case of sparsity in the dependent variable, the problem is different as we cannot simply remove this variable. As an example, consider the number of DSP blocks in a Xilinx FPGA, which will not be present in many designs. [Welsh et al. 1996] proposed a multilevel approach to tackle this problem. Similarly, in *Quipu*, first, a binomial model is generated that predicts the chance of a nonzero response. Then a second model provides the expected response value. We discussed this in more detail in Meeuws et al. [2011] and Meeuws [2012].

Apart from sparse data, it is important to determine the presence of any *outliers*. Outliers are data points that are significantly different from the other data points. It is not a good practice to remove outliers from datasets to produce better results, unless there is a clear general explanation why these outliers exist. In this article, we have omitted specific outliers in the case of Catapult-C, where global variables were not detected, which resulted in unwanted and incorrect optimizations in the obtained results.

3.5. Model Evaluation

Cross-Validation (CV) is essential for validating the predictive quality of statistical models. In this process, a part of the data is used for model calibration and another part is used for validation. The simplest form of CV is *Holdout Cross-Validation* [Kohavi 1995], where the dataset is divided into two subsets: one for training and one for validation. In case only a small set of data points is available, this method has the disadvantage that the training set may not be representative for the calibration set. This leads to a huge variance in the reported error depending on the data points that are selected for each set.

Another CV method is *K-fold Cross-Validation* [Kohavi 1995]. In this method, the data is split in K subsets. Each subset is used as a validation set once, while the remaining $K-1$ subsets are used collectively as training set. In our work, we use the 10-Fold CV method, as suggested in Kohavi [1995] and Sheiner and Beal [1981].

In order to express the error of a prediction model, most approaches calculate the *Mean Absolute Percentage Error* (MAPE) as follows:

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right|. \tag{7}$$

As we will see in Section 4, for example in Figure 3, individual percentage errors tend to be larger when closer to zero. This observation is also known in other fields of research [Guang et al. 1995]. In order to explain this, let us assume a constant variance of the errors, as is the case for normally distributed data. In this case, if we consider $\hat{y}$ to have a constant error component ϵ_c, we see that the percentage errors tend to follow an inverse relation:

$$\left|\frac{y_i - \hat{y}_i}{y_i}\right| = \left|\frac{y_i - (y_i + \epsilon_c)}{y_i}\right| = \left|\frac{\epsilon_c}{y_i}\right|. \tag{8}$$

As a result, the MAPE will be adversely influenced by datasets with more relatively small kernels. A more robust summary statistic is the *Rooted Mean Squared Error*, which is not affected by large relative errors for small kernels. Indeed, the most common error summary statistic used in this type of CV is the *tenfold CV Relative Rooted Mean Squared Error of prediction* (RMSEp%):

$$\text{RMSEp\%} = \frac{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}}{\frac{1}{n}\sum_{i=1}^{n} y_i}, \tag{9}$$

where y_i are the observed values and $\hat{y}_i$ are the predicted values. This summary statistic captures the overall predictive performance of our model. Nevertheless, in order to provide a clear picture of the error behavior of our models, we also provide visual characterizations of the error using histograms and plots of the relative error.

3.6. The Kernel Library and Tools

As shown in Figure 1(b), *Quipu* consists of a set of tools and a kernel library, which we describe in the following.

Kernel Library. In order to generate sufficient data points for the modeling process, *Quipu* gathers SCMs and hardware performance indicators from its extensive kernel library. This is a database of 324 kernels from a wide variety of application domains, contrary to many existing techniques, which use sets of tens of kernels at most. This library is the main reason why *Quipu* can produce generally applicable models. As we will see in Section 4.5, this high number of kernels allows for the generation of

Table III.

Overview of the kernel library, the number of kernels and the performance in the generation of synthesizable HDL for four C-to-HDL compilers. For a more elaborate description of the library, please refer to Section 5.2 and Appendix A in [Meeuws 2012].

Domain	Kernels	Floating-Point	DWARV		Catapult-C		LegUp		C-to-Verilog	
			HDL[a]	synth.[b]	HDL	synth.	HDL	synth.	HDL	synth.
Bio-informatics	8	-	6	6	6	6	7	7	1	1
Compression	16	-	13	13	9	9	14	14	8	8
Cryptography	80	-	78	77	71	65	67	65	31	31
Data Processing	14	-	12	12	13	12	14	14	7	7
DSP	24	10	24	18	23	22	15	14	10	10
ECC	16	-	16	16	15	12	14	14	5	5
Mathematics	70	32	69	54	57	57	33	33	26	26
Multimedia	65	23	65	59	61	60	40	40	31	31
Physics	21	14	14	6	10	7	7	7	2	2
Unclassified	10	2	9	5	6	6	6	5	4	4
Total	324	81	308	266	271	256	226	213	125	125

[a]The number of kernels for which HDL was produced.

[b]The number of HDL files that were synthesizable.

domain-specific models as well. For example, the 80 cryptography kernels or the 70 mathematical kernels can be the basis of domain-specific models.

The *Quipu* modeling approach contains several scripts that traverse the library. These scripts contain the necessary hooks, where target tools can be inserted. Furthermore, similar scripts are provided for the synthesis of the generated HDL descriptions. As described in step 1 to 3 of Section 3.2, these scripts need to be adjusted for a particular combination of a toolchain and a platform, so that hardware can be generated for each kernel. An overview of the kernels in this library is provided in Table III. This table lists both the number of kernels for each application domain and the HDL generation capabilities of the target C-to-HDL tools that were employed in this work. These capabilities are measured in the number of kernels that were successfully translated to HDL and synthesized. As a result, our kernel library can also be utilized for benchmarking purposes.

Metrication Tool. The determination of all the SCMs described in Section 3.3 has been implemented in the *Quipu Metrication tool*, which produces an XML file containing SCM measurements for each kernel. This tool is written as an engine in the CoSy compiler system [ACE b.v. 2003], a comprehensive compiler that contains a large set of optimizations and is easily extensible by writing engines that can be plugged into the framework. Because our metrication engine operates on the IR of this compiler, it is possible to use existing CoSy optimization engines to match the optimizations that exist in the target tools. This adds additional flexibility to the *Quipu* modeling approach. Although this tool is essential to the our approach, the SCMs may also be useful within a Software Measurement framework that helps drive management of software development processes. In contrast to approaches such as Oliveira et al. [2010], where project, system, and process metrics are used, *Quipu* SCMs would relate the project effort estimation to the characteristics of the source code itself. As such, *Quipu* could be beneficial for example in estimating the effort of implementing an existing or partially implemented source code description to a target platform.

Hardware Measurement Tool. In order to gather the necessary hardware performance indicators, we have developed the *Quipu Hardware Measurement tool*. This tool consists of scripts that parse the output of different synthesis toolchains to gather

the necessary data. For some toolchains, this is as simple as finding the right data in the report files, but, for others, this is not trivial. For example, in order to count the number of wires in Xilinx designs, we created an XDL parsing tool that extracts this and other relevant data from the XDL file.

Modeling Scripts and Prediction Tool. The gathered SCMs and hardware measurements are utilized by a set of modeling scripts that automatically evaluates the different modeling techniques described in Section 3.4. The output model XML file can be used in the *Hardware Estimation*, where, based on SCM inputs, the *Quipu Prediction tool* provides estimates of any required hardware aspects. All intermediate files in the *Hardware Estimation* are saved in XML format for easy integration. Additionally, the results of execution and memory profiling tools, for example, might be integrated as depicted in Figure 1(b).

4. *QUIPU* PREDICTION: EXPERIMENTAL EVALUATION

In this section, we first present a case study in retargeting the *Quipu* modeling approach to the combination of a new toolchain and a new platform. After that, we present the evaluation of prediction models for four different combinations of toolchains and platforms, and for different domains.

4.1. Experimental Setup

In the context of our experiments, we used the following four combinations of C-to-HDL compilers, synthesis toolchains, and platforms.

— DWARV/*Xilinx*. This combination consists of the DWARV C-to-VHDL compiler [Yankova et al. 2007] from the *DWB*, the Xilinx ISE 13.3 synthesis tools, and the Virtex 5 FX-200T FPGA. This FPGA consists of 30,720 slices. The HDL generation took about 2 minutes. We have synthesized the designs for speedup using the following optimization settings: `-opt_mode Speed -opt_level 2`. The designs were fully mapped and routed.
— *Catapult-C/Synopsys*. This combination uses the Catapult-C High Level Synthesis Compiler University Edition [Morris 2004] together with the Synopsys Design Compiler version F-2011 targeting 45nm ASICs using the NanGate FreePDK45 Generic Open Cell Library [Nangate Inc.]. Catapult-C was run with standard optimization levels targeting a 200MHz clock speed without any kernel-specific settings. HDL Generation took around 8 hours.
— *LegUp/Synopsys*. This combination consists of the LegUp C-to-Verilog compiler [Canis et al. 2011] from the University of Toronto and the same Synopsys setup as the Catapult-C/Synopsys combination.. The LegUp compiler was executed using the process described in the documentation with the notable change of the clang optimization flag to `-O2`. HDL generation took around 8 hours. We used medium effort levels for any synthesis optimizations and employed full placement and routing.
— *C-to-Verilog/Altera*. This combination consists of the C-to-Verilog compiler from the Haifa University [Ben-Asher and Rotem 2008, 2010], the Altera Quartus 10.0 synthesis tools, and the Stratix IV EP4SE820 FPGA. This FPGA consists of 325,220 Adaptive Logic Modules (ALMs). The HDL generation took around 45 minutes. The designs were synthesized using default effort levels and were fully mapped and routed.

All designs were synthesized on an 8-core 2.67GHz Intel Xeon X5550 machine with 32GB of ram running a 64-bit version of CentOS 5.7 Linux using around 6 parallel syntheses. More parallel sessions would lead to memory problems. The individual tools were executed with multithreading, where available.

The SCMs were determined using our *Quipu* Metrication tool. If not otherwise specified, the SCMs were determined utilizing the `-O1` optimization flag of the cosy compiler. This process takes 6.5s to complete for the 324 kernels in the library using no optimizations, 15.1s using optimization level 1, and 34.1s using optimization level 2. It should be noted that SCMs for an additional set of 244 functions not marked as kernels were also determined as a side-effect, yielding an average prediction speed of 87.4 kernels per second. This number increases when predictions for multiple platforms are required, as the source code does not have to be processed again.

For modeling, we used the *Quipu* modeling scripts without any manual tweaking, unless otherwise specified. The modeling process was executed on the same 8-core machine mentioned before. Upto 6 parallel LR models were generated in parallel. ANN training used 14 parallel threads to utilize the hyper-threading capability of the cores.

4.2. Retargeting *Quipu* to LegUp/Synopsys: A Walkthrough.

To show the generality of the *Quipu* modeling approach, we describe here the process of retargeting the *Quipu* models to the combination of a new toolchain and a new platform. For this purpose, we consider the LegUp C-to-Verilog compiler together with the Synopsys Design Compiler targeting the NanGate Open Cell Library. In order to generate models for this new combination of tools, we followed the procedure described in the next paragraphs. For different combinations of toolchains and platforms, an equivalent approach can be followed, possibly skipping some steps.

P*reparation: Understanding the new toolchain.* In order to retarget *Quipu*, it is essential to first understand the new tools and platform. In the case of LegUp/Synopsys, it was relatively straightforward to install and start generating Verilog and synthesizing designs. We believe an experienced user can perform this task in one or two days.

S*tep 1: Kernel library customization.* As we need hardware results for all kernels in the kernel library, we changed the scripts to generate Verilog automatically, starting from the manual steps learned during the preparation stage. The main difficulty was to properly isolate the functions in the library, as the LegUp compiler compiles complete files to a single Verilog file. Using the *llvm − extract* command in the LLVM compiler, we were able to achieve the required behavior. This step was performed in one day.

S*tep 2: HDL generation.* After the appropriate scripts were in place, the generation of Verilog was performed in less than an hour. The LegUp compiler compiled 226 kernels.

S*tep 3: HDL synthesis.* We determined the name of the top-level designs in the Verilog files and used our experience from the preparation stage to write a TCL script for the Synopsys design compiler. We were able to start the synthesis of the kernel library in less than an hour. Afterwards, the synthesis of the generated Verilog took 2 hours.

S*tep 4: Data extraction.* With all kernels synthesized, we extracted the SCMs from the ANSI-C sources and hardware data from the synthesis results. Extracting the SCMs from the ANSI-C source code is especially important if code changes have been made or in case the *Quipu* Metrication tool should take into account specific optimizations. It is important to perform a sanity check on the hardware data, as it can be the case that faulty HDL code resulted in zero-sized designs or in other problematic data. This process took 17s to complete for the 324 kernels in the library using the `-O1` flag.

S*tep 5: Statistical modeling.* Based on the extracted data, we performed the semi-automatic statistical modeling procedure in the *Quipu* modeling approach. First, a *quick* modeling is performed, where ANNs are omitted. This takes a few minutes. We

repeated this procedure using several SCMs sets for different optimization levels. Furthermore, we evaluated several values for the zero-threshold parameter, which omits SCMs that have a certain percentage of zeroes in the dataset. Using the best obtained threshold value, we performed the full modeling process, which took around 2 hours for the estimation model for the total area reported by Synopsys. Although, the results were sufficient, the set of SCMs can be further refined by manually investigating the data to find the SCMs that demonstrate a clear correlation with the total area. For this purpose, we may visually inspect scatterplots for each SCM against the total area. However, we did not manually refine the model in this way in this article. Finally, we executed the modeling procedure using this refined set to obtain the final *Quipu* prediction model. The whole modeling process took one day.

The process just described shows how the *Quipu* modeling approach can be retargeted to a new platform and toolchain within a few days. In case of poor compilers, or extensive compiler optimizations that are not accounted for by the *Quipu* Metrication tool, this process can take a few additional days, in order to adjust the kernel library code or to implement the necessary optimizations in the *Quipu* Metrication tool.

4.3. C-to-HDL Compiler Benchmarking

As listed in Table III, the evaluated C-to-HDL compilers are able to translate different numbers of ANSI-C inputs to HDL descriptions. Furthermore, some of these descriptions are not synthesizable. More precisely, we observe that DWARV is able to generate synthesizable VHDL for 82.1% of the kernels, Catapult-C for 79.0%, LegUp for 65.7%,[4] and, finally, C-to-Verilog performs the worst and can generate synthesizable VHDL only for 38.6% of the kernels. In this respect, we consider our kernel library to also be a useful benchmark in the evaluation of different C-to-HDL compilers. Especially, it allows to compare the capabilites of these compilers in accepting ANSI-C constructs, while still generating syntactically correct HDL code.

4.4. Domain-Agnostic Modeling

In Figures 2 and 3, we plotted the prediction quality and the error distribution for each *Quipu* model generated for one of the four target toolchains, respectively. Figures 2(a)–(d) depict the prediction quality of the domain-agnostic models by relating the predicted area metric to the actual area metric. The line $y = x$ denotes the case where the model has perfect predictions. Each dot indicates the result for one kernel in the library. The models in these figures are domain-agnostic, which means that all the available domains were taken into account to generate the model. In Figures 3(a)–(d), we see the error distributions for these models according to the observed values and according to the probability density. An error value of 1 denotes an error of 100%. It can be seen that the three domain-agnostic models exhibit very similar behaviour in these plots. This is more evident from Table IV, where we see that the errors for the domain-agnostic models range from 20.3% to 39.5%.[5] This table further includes the evaluation criteria from Section 3.5 and the details on what type of models have been selected by our modeling scripts as described in Section 3.4. We have also included in this table the error results from Cilardo et al. [2010], which targeted Xilinx FPGAs. This model performs worse compared to the *Quipu* model

[4]LegUp does not support floating point operations and, as such, the performance is degraded. Integer only performance was 87.7%.

[5]We note that the error for the C-to-Verilog/Altera case is higher than reported in Meeuws et al. [2011]. This is partly because of the extra number of kernels used in generating the model, that is, 125 instead of 118. Additionally, the website used to generate the Verilog code seems to have changed the code generation.

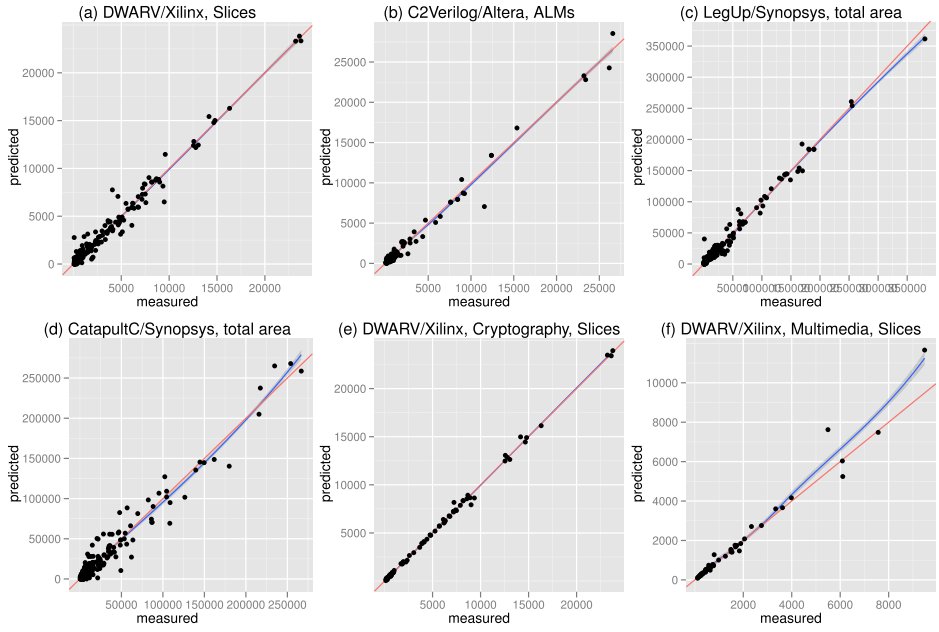


Fig. 2. The prediction performance of different *Quipu* models generated for different tools and platforms.

Table IV.

Overview of the model performance of several *Quipu* prediction models for different toolchains, platforms, and domains.

	Model	Measure	RMSEp%	MAPE	Kernels	Model type
Domain-agnostic	[Cilardo et al. 2010]	slices	-	31%	3	OLSR
	Dwarv/Xilinx (XST)[a]	slices	29.9%	42.1%	266	ANN (3-11-1[b])
	Dwarv/Xilinx (SCM)	slices	23.8%	52.6%	266	GLM (normal)
	Altera/C-to-Verilog	ALMs	23.9%	36.2%	125	ANN (25-1-1)
	Legup/Synopsys	total area	20.3%	61.8%	213	ANN (26-7-1)
	Catapult-C/Synopsys	total area	39.5%	56.6%	256	ANN (39-9-1)

	Model	Measure	RMSEp% (agnostic)[c]	RMSEp% (specific)[d]	MAPE	Kernels	Model type
Domain-specific	Cryptography	slices	6.7%	2.4%	5.5%	74	GLM (poisson)
	Multimedia	slices	36.1%	29.2%	8.6%	59	GLM (poisson)
	No Control Flow	latency	-	9.8%	11.1%	57	GLM (gamma)

[a] Statistical model based on XST high level estimates.
[b] Number of nodes in the input–, hidden–, and output layer.
[c] Same model as Dwarv/Xilinx (SCM).
[d] RMSEp% for the domain-specific model.

targeting the Xilinx platform. Furthermore, although Cilardo et al. use a set of 200 synthetic kernels in their approach, the validation is based on only 3 real kernels.

As an additional comparison, we generated a linear model based on the high level estimates provided by the Xilinx XST 13.3 high level synthesis tool. This model exhibits errors that are above the error reported for our DWARV/Xilinx model. It is interesting that our high-level estimates even outperform models that use the Xilinx XST estimates as parameters when we consider the RMSEp%. A possible explanation is that XST is independent of specific devices,[6] and does not take into account mapping, placement, and routing. Instead, *Quipu* models are generated using

[6]Although XST is independent of specific devices, it is not independent of device families.

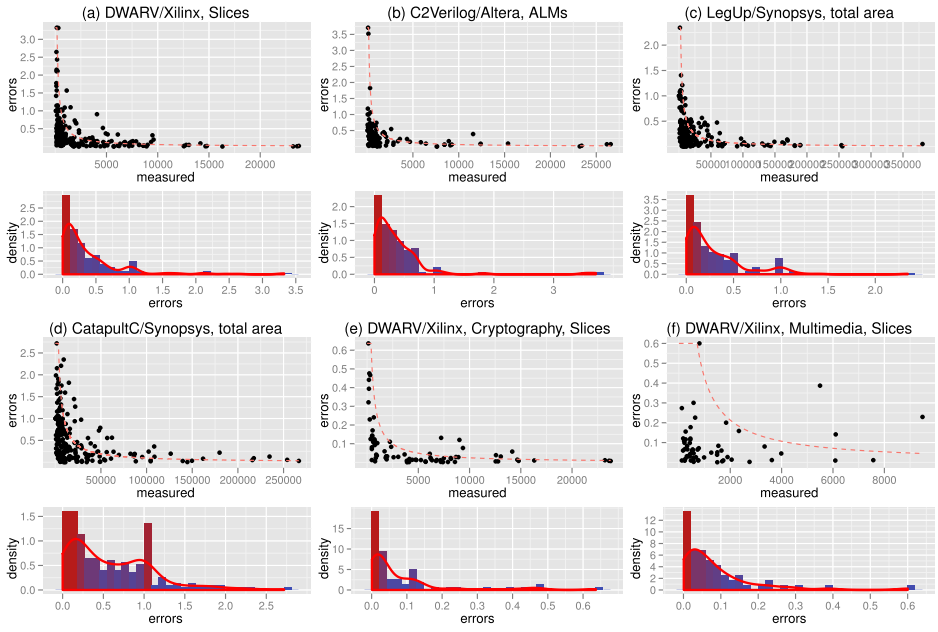


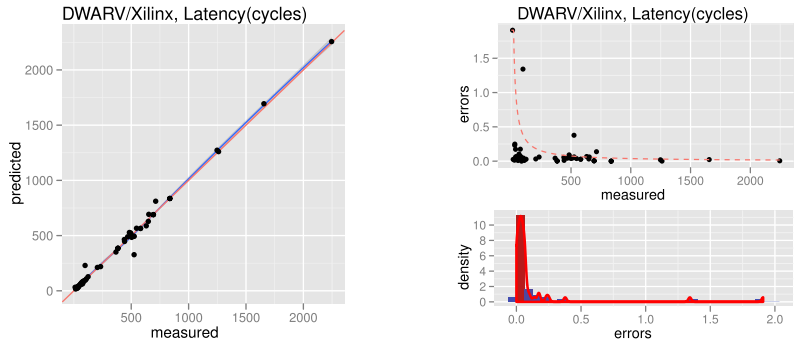
Fig. 3. The error distribution of different *Quipu* models generated for different tools and platforms.

the results after placement and routing has been performed, making it possible to capture the effect of these design steps.

Notice that the errors increase as the size of the kernel gets closer to zero. This holds true for all four cases. The reason for this behavior is that there seems to be a fixed component in the variance, which becomes more significant when the component is smaller. When we consider that the used regression techniques aim to minimize the absolute error, this error will be normally distributed when the number of data points becomes large. Therefore, it makes sense for the percentage errors to have an inverse relation with the measured values. We plotted the standard deviation σ as a percentage of the measured values in the plot for comparison, indicated by the dashed red line. The main outliers visible in the graph also exist in the set of smaller kernels, several of these outliers also contain mostly bitwise operators and constant shifts. We point out that our models are not accurate for very small kernels, especially if they mostly consist of bitwise operators and constant shifts.

Another important observation is that the histograms and probability density trends, in Figure 3, show that the errors are not *normally distributed*. In fact, the distributions are skewed towards zero. This means that traditional error statistics do not capture wholly the error behavior of the model. Instead, these histograms better characterize the error behavior of our models. Another effect of these nonnormal error distributions is that statistical significance tests should not be used as they assume the normal distribution.

In contrast with the other cases, the Catapult-C model exhibits a somewhat larger error. The main reason for this behavior is the level of optimizations that is performed by this compiler. More specifically, Catapult-C tries to allocate memory buffers for any input pointers. Starting from a buffer size of 1024 elements for each pointer, it subsequently tries to reduce this size. However, many kernels in the database have unbounded loops, so pointer sizes are not always known. These kernels are then



(a) The actual number of cycles vs. the predicted number of cycles.

(b) The prediction errors vs. the predicted number of cycles.

Fig. 4. The prediction performance of the number of cycles by *Quipu* models generated for the DWARV/Xilinx/Virtex 5 case, specifically for kernels without Control Flow.

fixed at the arbitrary number of 1024 elements. Because the source code does not have a relation with this externally provided threshold, the prediction performance is degraded. Indeed, when we generated a model for the 113 kernels that were not fixed to 1024 elements, we obtained an RMSEp% of 23.9%. An interesting feature of the error histogram in Figure 3 for Catapult-C is the high number of kernels with an error around 1.0 (or 100%). This relates to a relatively high number of negative predictions by the underlying model, which are truncated to 0 during prediction. A prediction of 0 always relates to an error of 100%.

4.5. Domain-Specific Modeling

In order to show the domain-specific modeling of our approach, we have generated three domain-specific models: one for Cryptography, one for Multimedia, and one without control constructs, as depicted in Figures 2(e)–(f), 3(e)–(f), and 4(a)–(b). As can be seen in Table IV, the errors of the first two models are well below the error for the domain-agnostic model, if we compare to the error of the domain-agnostic model for the domain subset. In the case of Cryptography, the error is reduced from 6.7% to only 2.4% and, in the case of Multimedia, it is reduced from 36.1% to 29.2%. Observe that the error for the Multimedia domain using the domain-agnostic model is even larger than the error for the whole dataset.

The small errors for these domains can be explained by the specific characteristics of these two domains. The Cryptography domain consists of highly regular code that contains repeated patterns of additions, shifts, bitwise operators, and array accesses. In addition, this domain contains no floating point operations. Furthermore, as the kernels in this domain are very large, the small constant effect that adversely influences the error for small kernels does not become a problem. The Multimedia domain, in contrast, contains many image, video, and sound algorithms. These algorithms often process data in blocks, which implies a certain level of regularity among the kernels in this domain.

Apart from generating domain-specific models based on the application domain of each kernel, the SCMs in the *Quipu* modeling approach allow other subset models to be generated as well. For example, we have generated a model for kernels without branches, as depicted in Figure 4. This particular model predicts the latency in number of cycles for each kernel with an error of only 9.8%.

Table V. Results of the Q^2 Profiling Framework and the Corresponding Partitioning for the Virtex-5 FX-200T

Application	Number of kernels		Area (slices)			Comm. Red.[a]	Speedup
	Total	Mapped	Prediction	Actual	Error(%)		
MELP	59	4	6534	6043	8.1%	57.1%	1.30x
CED	12	4	5381	7307	26.4%	22.7%	2.92x
N-Body	10	4	11730	8209	30.0%	42.9%	n/a

[a]Estimation Communication Reduction as reported by QUAD.

4.6. Model Utilization

In order to show how *Quipu* prediction models can be utilized in a real scenario, we present some results of the Q^2 profiling framework in Table V. This table provides a summary of the results for three well-known applications. The first application is the *Mixed Excitation Linear Prediction* (MELP) *vocoder* [Supplee 1997], used mainly for secure voice in radio devices. *MELP* uses extensive look-up tables and models of the human voice to extract and regenerate speech and, as such, it is a computation-intensive application. *Canny Edge Detection* (CED) [Canny 1986] is a well-known edge detection algorithm, which outperforms many other edge detection methods. *N-Body* [Hut et al. 1995] is a widely used technique to investigate the evolution of particles in various fields of science, such as physics or astronomy. All three of these applications are computation-intensive and can benefit from mapping parts of the application into reconfigurable fabrics.

The goal of the Q^2 profiling framework is to provide the necessary profiling information in order to efficiently partition the application into hardware and software. Based on hot-spots in the program, a preliminary partitioning can be proposed by moving the top number of hot-spots to the hardware. However, with limited resources available, it is important to predict the area consumption, so that we can evaluate different partitionings. The Virtex 5 model utilized for the predictions in Table V indicates, for example, that all proposed mappings would fit on a Virtex 5 FX-200T, but the proposed mapping for *N-Body* would not fit on a Virtex 5 FX-30T, which contains only 5120 slices. We see that Q^2 was able to propose a partitioning that yields in communication reductions and speedups. This partitioning was determined, partly, using the *Quipu* prediction models. The accuracy of the predictions for each of the applications is also somewhat different. The small error of the MELP application, seems due to the many relatively small kernels that were mapped, while the large error of *N-Body* is most probably related to the heavy use of floating point kernels, which are underrepresented in our Kernel Library.

4.7. Comparison with Other Approaches

As mentioned in Section 2, one of the main issues of existing prediction approaches is the confidence that can be attributed to the reported error. In Table VI, we have listed the error rates and the sizes of the validation sets for the approaches described in Section 2. From this table, it is evident that apart from Schumacher et al. [2008] and *Quipu*, none of the other approaches use a sizeable validation set to substantiate their claims. Even though, approaches such as [Cilardo et al. 2010] feature a set of up to 200 kernels, these are not used for validation purposes, but for model generation. In contrast, we utilize a library of 324 kernels to generate our models and, subsequently, utilize up to 266 of those kernels for validation purposes by using tenfold cross-validation. These 266 kernels were the maximum number of kernels that could be translated to synthesizable HDL using the C-to-HDL compilers at our disposal. Tenfold cross-validation splits the data in 10 sets which are used separately

Table VI. Overview of the Performance and Validation Quality of the Existing Estimation Approaches Including *Quipu*

No regression or no HLL (C)			Regression and HLL (C)		
Reference	Error[a]	Size[b]	Reference	Error[a]	Size[b]
[Enzler et al. 2000]	12%	6	[So et al. 2003]	-	5
[Lakshmi et al. 2011]	4%	< 13	[Holzer and Rupp 2005]	39.3%	9
[Schumacher et al. 2008]	21.9%	90	[Degryse et al. 2008]	7.14%	12
[Nayak et al. 2002]	16%	7	[Kulkarni et al. 2006]	5.3%	4
[Brandolese et al. 2004]	36.8%	5	[Cilardo et al. 2010]	31.3%	3
[Bilavarn et al. 2006]	22%	20	***Quipu***	**23.8%**	**266**
[Deng et al. 2008]	8.2%	12			
[Chuong et al. 2009]	10.3%	10			

[a]Error in the number of slices or comparable area metric.
[b]The number of kernels in the validation set.

as validation sets, while the corresponding remaining sets are used to generate the prediction model used in validation.

Interestingly, the only other approach that features a substantial validation set, Schumacher et al. [2008] also present an error rate of around 21.9%. However, that approach targets VHDL and not ANSI-C and it is limited to the Xilinx toolchains and platforms. An important question is how the other approaches perform with larger validation sets. In addition, most related works did not specify the exact method that was used to determine the error. We assume that in most cases MAPE is used for reporting the error.

5. CONCLUSIONS

In this article, we have presented the extensive *Quipu* quantitative hardware prediction modeling approach that targets the early stages of HW/SW codesign. This approach consists of a set of tools, a library of kernels, and a modeling methodology, which together provide valuable information for identifying resource-intensive parts of an application. This way, it helps with the evaluation of different mapping options, or for guiding developers in merging and splitting, among other things. We have shown how this approach is able to generate adequate prediction models for several hardware measures targeting a number of different platforms in the reconfigurable and ASIC domains. Furthermore, we have shown how our approach can semi-automatically generate models for a new toolchain and/or platform within a few days. This is essential when new architectures and tools become available and need to be quickly integrated in the design process in order to keep the time-to-market as low as possible.

In contrast to most of the existing approaches, we provide a substantial validation set of 324 kernels from well-known applications from many different application domains. This provides the much needed confidence about the reported error of our generated models. Furthermore, *Quipu* enables us to generate prediction models for specific application domains, which show an increased accuracy. With errors between 4.6% and 39.5%, measured using the RMSEp%, our models are useful in the early stages of HW/SW codesign. In these phases, it is essential to quickly obtain sufficient insight into the hardware characteristics in order to, for example, prune the design space, focus the effort within manual hardware design, or generate preliminary partitionings and cost estimates.

Although our approach provides many essential hardware estimates, there is still a clear need for estimates of more dynamic characteristics, such as speedup and dynamic power, especially in the early stages of the design, when it is not feasible

to generate and simulate hardware for multiple platforms. To this purpose, we are currently working on incorporating dynamic SCMs and run-time prediction, which enable these kinds of estimations.

REFERENCES

ACE B. V. 2003. CoSy compilers, overview of construction and operation.

Altera. 2011. SoC FPGA ARM Cortex-A9 MPCore processor advance information brief.

Banerjee, P., Shenoy, N., et al. 2000. A MATLAB compiler for distributed, heterogeneous, reconfigurable computing systems. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'00)*. 39.

Ben-Asher, Y. and Rotem, N. 2008. Synthesis for variable pipelined function units. In *Proceedings of the International Symposium on Systems-on-Chip (SOC'08)*. 1–4.

Ben-Asher, Y. and Rotem, N. 2010. Automatic memory partitioning: Increasing memory parallelism via data structure partitioning. In *Proceedings of the International Workshop on Hardware/Software Codesign (CODES'10)*. 155–162.

Bertels, K. Vassiliadis, S., Panainte, E. M., Yankova, Y., Galuzzi, G., Chaves, D., and Kuzmanov, G. 2006. Developing applications for polymorphic processors: The Delft Workbench. Tech. rep. CE-TR-2006-XX.

Bertels, K., Simna, V.-M., et al. 2010. HArtes: Hardware-software codesign for heterogeneous multicore platforms. *IEEE Micro 30*, 5, 88–97.

Bertels, K., Ostadzadeh, S. A., and Meeuws, R. J. 2011. Advanced profiling of applications for heterogeneous multi-core platforms. In *Proceedings of the International Conference on Engineering of Reconfigurable Systems & Algorithms (ERSA'11)*. 171–183.

Bilavarn, S., Gogniat, G., Philippe, J-L., and Bossuet, L. 2006. Design space pruning through early estimations of area/delay tradeoffs for FPGA implementations. *IEEE Trans. Comput. Aided Des. Integr. Circ. Syst. 25*, 10, 1950–1968.

Box, G. E. P. and Cox, D. R. 1964. An Analysis of Transformations. *J. R. Stat. Soc. Series B 26*, 2, 211–252.

Brandolese, C., Fornaciari, W., and Salice, F. 2004. An area estimation methodology for FPGA based designs at SystemC-level. In *Proceedings of the IEEE/ACM Design Automation Conference (DAC'04)*. 129–132.

Callan, R. 1998. *Essence of Neural Networks*. Prentice Hall, Upper Saddle River, NJ.

Cammarota, R., Kejariwal, A., D'Alberto, P., Panigrahi, S., Veidenbaum, A. V., and Nicolau, A. 2011. Pruning hardware evaluation space via correlation-driven application similarity analysis. In *Proceedings of the 8th ACM International Conference on Computing Frontiers (CF'11)*. 4:1–4:10.

Canis, A., Choi, J., Aldham, M., Zhang, V., Kammoona, A., Anderson, J., Brown, S., and Czajkowski, T. 2011. LegUp: High-level synthesis for FPGA-based processor/accelerator systems. In *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA'11)*. 33–36.

Canny, J. 1986. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell. 8*, 679–698.

Chen, T., Raghavan, R., Dale, J. N., and Iwata, E. 2007. Cell broadband engine architecture and its first implementation: a performance view. *IBM J. Res. Dev. 51*, 559–572.

Chuong, L. M., Lam, S.-K., and Srikanthan, T. 2009. Area-time estimation of controller for porting C-Based functions onto FPGA. In *Proceedings of the IEEE/IFIP International Symposium on Rapid System Prototyping (RSP'09)*. 145–151.

Cilardo, A., Durante, P., Lofiego, C., and Mazzeo, A. 2010. Early prediction of hardware complexity in HLL-to-HDL translation. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL'10)*. 483–488.

Degryse, T., Devos, H., and Stroobandt, D. 2008. FPGA resource estimation for loop controllers. In *Proceedings of the 6th Workshop on Optimizations for DSP and Embedded Systems (ODES'08)*. 9–15.

Deng, L., Sobti, K., and Chakrabarti, C. 2008. Accurate models for estimating area and power of FPGA implementations. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP'08)*. 1417–1420.

Eeckhout, L., Vandierendonck, H., and Bosschere, K. D. 2002. Workload design: Selecting representative program-input pairs. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*. 83–94.

Elshoff, J. L. 1984. Characteristic program complexity measures. In *Proceedings of the International Conference on Software Engineering (ICSE'84)*. 288–293.

Enzler, R., Jeger, T., Cottet, D., and Tröster, G. 2000. High-level area and performance estimation of hardware building blocks on FPGAs. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL'00)*. 525–534.

Faraway, J. 2006. *Extending the Linear Model with R*. CRC Press.

Guang, W., Baraldo, M., and Furlanut, M. 1995. Calculating percentage prediction error: A user's note. *Pharm. Res. 32*, 4, 241–248.

Gupta, S., Dutt, N., Gupta, R., and Nicolau, A. 2003. Spark: A high-level synthesis framework for applying parallelizing compiler transformations. In *Proceedings of the 16th International Conference on VLSI Design*.

Halstead, M. H. 1977. *Elements of Software Science*. Computer Science Library.

Harrison, W. 1992. An entropy-based measure of software complexity. *IEEE Trans. Software Eng. 18*, 11, 1025–1029.

Harrison, W. and Magel, K. 1981. A topological analysis of the complexity of computer programs with less than three binary branches. *SIGPLAN Not. 16*, 4, 51–63.

Holzer, M. and Rupp, M. 2005. Static estimation of execution times for hardware accelerators in system-on-chips. In *Proceedings of the International Symposium on Systems-on-Chip (SOC'05)*. 62–65.

Hut, P., Makino, J., and McMillan, S. 1995. Building a better leapfrog. *Astrophys. J., Part 2. Lett. 443*, L93–L96.

Kohavi, R. 1995. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'95)*. Vol. 2, 1137–1143.

Kulkarni, D., Najjar, W. A., Rinker, R., and Kurdahi, F. J. 2006. Compile-time area estimation for LUT-based FPGAs. *ACM Trans. Des. Autom. Electron. Syst. 11*, 1, 104–122.

Lakshminarayana, A., Shukla, S., and Kumar, S. 2011. High level power estimation models for FPGAs. In *Proceedings of the IEEE Symposium on VLSI*. 7–12.

McCabe, T. J. 1976. A complexity measure. *IEEE Trans. Softw. Eng.*, 308–320.

Meeuws, R. J. 2007. A quantitative model for hardware/software partitioning. M.S. thesis, Delft University of Technology, Delft, The Netherlands.

Meeuws, R. J. 2012. Quantitative hardware prediction modeling for hardware/software co-design. Ph.D. thesis.

Meeuws, R. J., Galuzzi, C., and Bertels, K. 2011. High level quantitative hardware prediction modeling using statistical methods. In *Proceedings of the International Conference on Embedded Computer Systems: Architectures, Models, and Simulations (SAMOS'11)*. 140–149.

Meeuws, R., Sigdel, K., Yankova, Y., and Bertels, K. 2008. High level quantitative interconnect estimation for early design space exploration. In *Proceedings of the 17th International Conference on Field Programmable Logic and Applications (FPL'08)*. 317–320.

Meeuws, R. J., Yankova, Y. D., and Bertels, K. L. M. 2006. Towards a quantitative model for hardware/software partitioning. Tech rep., part of Rcosy DES.6392 project.

Meeuws, R. J., Yankova, Y. D., Bertels, K., Gaydadjiev, G. N., and Vassiliadis, S. 2007. A quantitative prediction model for hardware/software partitioning. In *Proceedings of the 17th International Conference on Field Programmable Logic and Applications (FPL'07)*. 317–320.

Monostori, Á., Frühauf, H. H., and Kókai, G. 2005. Quick estimation of resources of FPGAs and ASICs using neural networks. In *Proceedings of Lernen, Wissensentdeckung und Adaptivität (LWA'05)*. 210–215.

Morris, K. 2004. Catapult C: Mentor announces architectural synthesis. eejournalnet.

Munson, J. C. 2002. *Software Engineering Measurement*. CRC Press, Inc., Boca Raton, FL.

NanGate Inc. NanGate FreePDK45 Generic Open Cell Lib. v1.3.

Nayak, A., Haldar, M., Choudhary, A., and Banerjee, P. 2002. Accurate area and delay estimators for FPGAs. In *Proceedings of the Conference and Exhibition on Design, Automation and Test in Europe (DATE'02)*. 862.

Nocedal, J. and Wright, S. J. 2000. *Numerical Optimization*. Springer.

nVidia Corp. 2011. *Tegra 2 Technical Reference Manual*.

Oliveira, A. L., Braga, P. L., Lima, R. M. F., and Cornelio, M. L. 2010. GA-based method for feature selection and parameters optimization for machine learning regression applied to software effort estimation. *Inf. Softw. Tech. 52*, 11, 1155–1166.

Ostadzadeh, S. A., Meeuws, R. J., Ashraf, I., Galuzzi, C., and Bertels, K. L. M. 2012. The Q2 profiling framework: Driving application mapping for heterogeneous reconfigurable platforms. In *Proceedings of the International Workshop on Reconfigurable Computing: Architectures, Tools and Applications (ARC'12)*. 76–88.

Oviedo, E. I. 1980. Control flow, data flow, and program complexity. In *Proceedings of the Annual International Computer Software and Applications Conference (COMPSAC'80)*. 146–152.

Palermo, G., Silvano, C., and Zaccaria, V. 2009. ReSPIR: A response surface-based pareto iterative refinement for application-specific design space exploration. *IEEE Trans. Comput. Aided Des. Integr. Circ. Syst. 28*, 12, 1816–1829.

Rupp, K. and Selberherr, S. 2011. The economic limit to Moore 's law. *IEEE Trans. Semicond. Manuf. 24*, 1, 1–4.

Schumacher, P. and Jha, P. K. 2008. Fast and accurate resource estimation of RTL-based designs targeting FPGAS. In *Proceedings of the 17th International Conference on Field Programmable Logic and Applications (FPL'08)*. 59–64.

Schumacher, P. R., Miller, I. D., Parlour, D. B., Janneck, J. W., and Jha, P. K. 2011. Method of estimating resource requirements for a circuit design. Patent. US 7979835.

Sheiner, L. B. and Beal, S. L. 1981. Some suggestions for measuring predictive performance. *J. Pharmacokinetics Pharmacodynamics 9*, 503–512. 10.1007/BF01060893.

So, B., Diniz, P. C., and Hall, M. W. 2003. Using estimates from behavioral synthesis tools in compiler-directed design space exploration. In *Proceedings of the IEEE/ACM Design Automation Conference (DAC'03)*. ACM, 514–519.

Supplee, L. M. E. A. 1997. MELP: The new federal standard at 2400 bps. In *Proceedings of the IEEE International Conference on Acoustics Speech and Signal Processing*. 1591–1594.

Venables, W. and Ripley, B. 2002. *Modern Applied Statistics with S.* 4th Ed. Statistics and Computing, Springer.

Villarreal, J., Park, A., Najjar, W., and Halstead, R. 2010. Designing modular hardware accelerators in C with ROCCC 2.0. In *Proceedings of the 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM'10)*. 127–134.

Welsh, A., Cunningham, R., Donnelly, C., and Lindenmayer, D. 1996. Modelling the abundance of rare species: Statistical models for counts with extra zeros. *Ecol. Modell. 88*, 1–3, 297–308.

Xilinx. 2011. Zynq-7000 extensible processing platform summary.

Yankova, Y., Kuzmanov, G., Bertels, K., Gaydadjiev, G., Lu, Y., and Vassiliadis, S. 2007. DWARV: Delftworkbench automated reconfigurable VHDL generator. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL'07)*. 697–701.

Received March 2012; revised September 2012; accepted November 2012