Quantitative Prediction for Early Design Space Exploration in Delft WorkBench: An Outlook

Roel Meeuws, Kamana Sigdel , Yana Yankova , Koen Bertels
Computer Engineering, EEMCS
Delft University of Technology
{r.j.meeuws,k.sigdel,y.d.yankova,k.l.m.bertels}@tudelft.nl

Abstract— **In this paper, we discuss Quipu our multi-dimensional quantitative prediction model for hardware-software partitioning. The proposed model is based on linear regression between software metrics determined on a dataset of 127 kernels and measures from their corresponding hardware designs. These software metrics capture the complexity of the C language description. The hardware designs are determined using the DWARV C-to-VHDL translator [1].**

Currently, Quipu exhibits a relatively large error compared to lower level approaches, however the Quipu model can make fast and early predictions and is applicable to a wide variety of applications. For the moment, we have only considered prediction of area measures, like the number of slices or flip-flops. The main steps to improve Quipu are the following: 1) re-evaluation of the selected software metrics. 2) use of a lower level representation of the C code. 3) extension of the set of kernels. 4) extension of the modeled hardware parameters. In other words, a consolidated model can provide more, and more accurate information.

We conclude that fast and early prediction of hardware characteristics is important, but our approach was not accurate enough in the past. While a somewhat larger error is acceptable in the early stages of design, we need to improve our Quipu model. Furthermore, for Quipu to be applicable, it must predict additional hardware measures for a wider range of application domains.

Keywords—**Reconfigurable architectures, Modeling, Estimation, Statistics, Software metrics, System analysis and design**

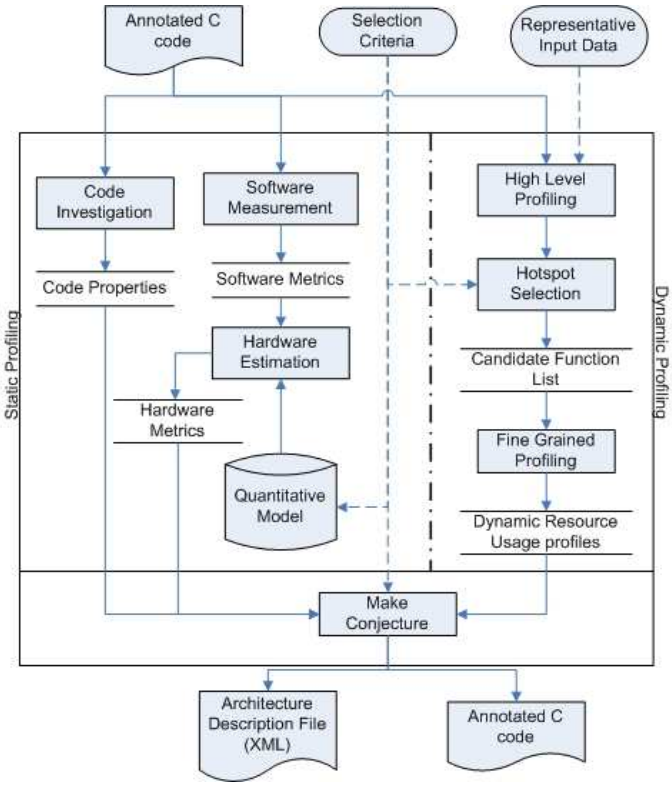

Fig. 1
Work flow of code profiling in Delft Workbench

I. Introduction

With the advent of Heterogeneous and Reconfigurable Computing, a need for a new generation of design support tools targeting these emerging paradigms has arisen. Where traditional tools focus on either hardware or software, as engineers are either software or hardware engineers, these new tools target both hardware and software in order to assist the future configware engineers [2].

The Delft Workbench[3] is such a tool platform supporting integrated hardware-software co-design, starting from profiling and partitioning to synthesis and compilation. An important step in any development process is Design Space Exploration, not in the least because decisions in this phase have a significant impact on the course of the project. During DSE in heterogeneous system design, engineers need to partition an application over different computational components, like CPUs, FPGAs, ASICs, or DSPs. This process, called hardware-software partitioning, requires evaluating the cost of implementing a task on each component. Because, DSE and HW-SW Partitioning iteratively evaluate a very large design space, fast cost evaluation is essential. Therefore, Delft Workbench uses code profiling to predict hardware characteristics and identify hot-spots, as depicted in Fig. 1. Using the profiling information we can prune the design

space and guide the partitioning process. Estimates for hardware resource consumption, for example, can be used to omit functions that are too large to fit on an FPGA, or too small to exploit any degree of parallelism.

For this purpose, we developed Quipu[1], a Multi-Dimensional Quantitative Prediction Model for Early Design Space Exploration as part of the Delft Workbench reconfigurable platform. This prediction model provides resource estimates for implementing tasks on hardware from a high level functional specification in C. The model is based on the hypothesis that software and hardware complexity are related. In [4] we have shown that this hypothesis holds true in case of area estimation. Quipu captures this relation by statistical modeling techniques.

In this paper, we evaluate the performance and applicability of our Quipu model and discuss several areas of improvement. First, we describe the current model in Section III and briefly evaluate previous results. Then, we discuss in detail the different areas of improvement in Section IV. And finally, we conclude in Section V

II. Related Research

Other prediction for early design space exploration can be found in e.g. [5], where a constant time incremental estimation approach targeted at iterative hardware/software partitioning is presented. This approach updates estimates at each partitioning step. In [6], the authors estimate area by using linear models at each DFG node. Each type of node uses its own type of linear model. In [7], another early estimation approach was presented. The authors present three custom prediction models for determining hardware implementation cost from VHDL specifications. However, they don't present a sound statistical analysis of their results. Furthermore, they use a small set of (60) observations and produce a large (not quantified) error. However, these approaches do not operate on C-level specifications as our model does.

In [8], we do find an early estimation scheme from C code. The authors present a custom model for execution time estimation based on the SPARK C-to-VHDL compiler. They briefly mention software metrics, but only use a kind of critical path estimation scheme based on CFGs. They also do not present a sound statistical analysis of their results. Based on 9 kernels from 2 applications they report an error between 39.3% and 44.4%.

[1] A Quipu, is a calculating and recording device use in the ancient Inca Empire made from multiple strings and knots.

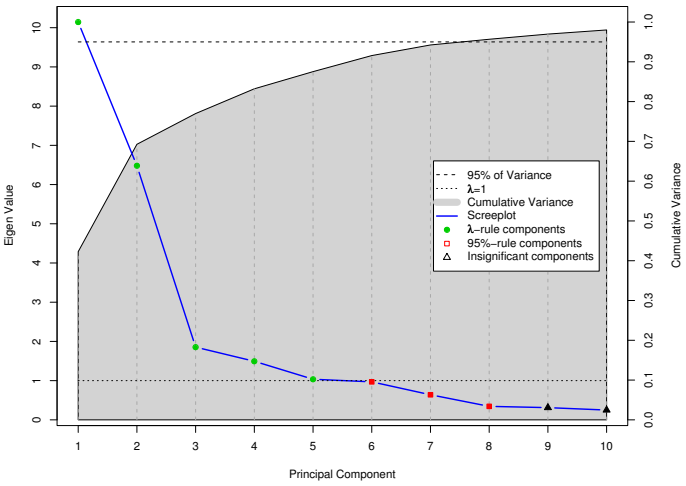

Fig. 2
Summary of PCA with Scree-plot and Variance, showing 12 PCs.

III. Quipu: A Multi-Dimensional Quantitative Prediction Model

Our Quantitative Model, Quipu, is based on Multi-Dimensional Linear Regression using Software Complexity Metrics. SCMs capture different aspects of software complexity from computer programs and functions, like program size or control intensity. Already, SCMs are used in software development processes to predict e.g. development time or the number of errors. Therefore, one advantage of using SCMs is that several independent variables may already be available. Additionally, most metrics are inherently simple and can be determined in a relatively short time.

Previously, we gathered 24 software metrics for Quipu. For a detailed description of these metrics refer to [4]. Together, these metrics capture a wide range of code characteristics. We have shown that some of these metrics correlate with hardware [4], [9]. However, there is also a certain level of correlation among the SCMs. This phenomenon is called interdependence or multicollinearity. For example, the Average Path Length, Maximum Path Length, and Statements all measure program length. Because classical linear regression requires independent variables, we transformed the set of metrics by applying Principal Component Analysis (PCA), yielding 8 principal components. In Fig. 2, we see the summary of this

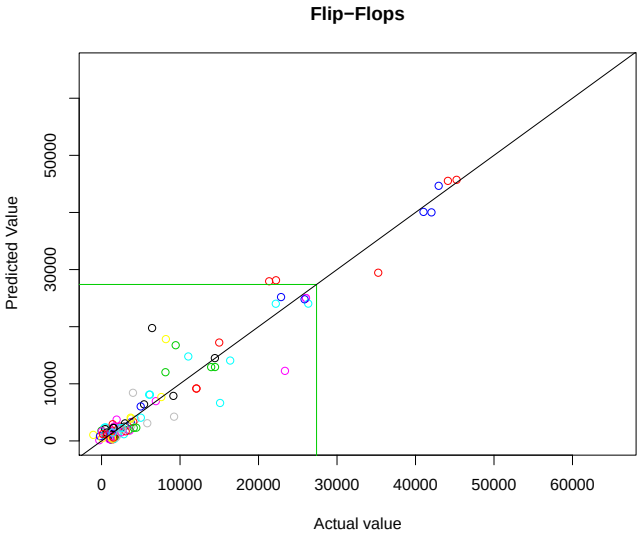

Fig. 3
Predicted vs. Actual number of flip-flops with indication or Virtex II Pro flip-lops (square) and ideal prediction (diagonal)

analysis. There are several ways to choose the number of relevant principal components (PC). For this paper we chose the number of PCs that represent 95% of the variance in the dataset.

The Delft Workbench targets the MOLEN reconfigurable platform[10], a combination of conventional and reconfigurable processors. The MOLEN platform as we used it, contains a Xilinx Virtex-II Pro. Therefore, we used the following area cost measures, which are basic elements of Xilinx FPGAs.

- **Flip-Flops** - D-type Flip-Flops.
- **Look-Up Tables (LUTs)** - 4-input LUTs.
- **Slices** - A basic element consisting of 2 LUTs, 2 D-Type Flip-Flops, and some extra elements like multiplexers and carry chains.
- **Multipliers** - 18bit multipliers

Furthermore, we added the number of states as a criterion for Finite State Machine (FSM) size. The hardware measures were obtained from automatically generating VHDL with the DWARV C-to-VHDL compiler[1]. The used VHDL compiler and synthesizer with their respective optimization options affects the behavior and quality of the Quipu model. Currently, Quipu is based on a specific set of tools. However, in the future we envision an automatic model generator for other tool sets.

Currently, Quipu utilizes a set of 135 C–kernels that have been translated into VHDL with the DWARV C–to–VHDL translator as the basis for linear regression. We were able to make predictions for the number of flip-flops with an average expected error of 66.6%, as listed in Table I. The predictive quality of Quipu's model for flip-flops is depicted in Fig. 3.

Model	Prediction Error	R^2
Slices	89.5%	0.717
Flip-Flops	66.6%	0.920
LUTs	128.6%	0.628
States	79.5%	0.795

TABLE I
Error Analysis of the Quipu model.

A. Evaluation

The current version of our model has some strong and weak points that we need to evaluate. First, the strong points of our model are:

- **Early Estimation**

Our approach works from a high level specification by using software complexity metrics, making early predictions possible.

- **Fast Estimation**

Our approach can make estimations very fast. For example, area estimations for 135 kernels took 7.5 s on a 2.4 GHz Athlon64 processor.

- **One Model**

Our model can estimate several aspects, without the need for different estimation techniques. For example, when a new optimization is introduced, the model can easily be extended to predict the area requirements of this optimization.

These three characteristics make the model especially suited for design space exploration. In the first place, early estimation is essential for any DSE on a high level specification. Secondly, fast estimation minimizes the overhead of DSE and partitioning due to their highly iterative nature. And finally, a single easily extensible model minimizes the costs of considering new optimizations and components during DSE.

However, there are also two major shortcomings that need to be addressed for our model to be viable:

- **Large Prediction Error**

The Quipu model still results in large error margins on the scale of 66.6%-128.6%. While this margin might be acceptable for a proof-of-concept model, it is unacceptable for any real world situation.

- **Narrow Scope**

Our model is currently tailored to the DWARV C-to-VHDL compiler, i.e. it has the same code restrictions and disregards resource constraints.

Although some margin of error is acceptable at the high level that our model operates at, the current expected error margins give problems. Especially when an application has a small set of kernels, which all exhibit a large error, the model can not be used to guide DSE. Furthermore, in order to consider different hardware generation schemes, Quipu should loosen it's restrictions on the source code.

IV. Areas of Improvement

Given the shortcomings mentioned in Section III-A, we need to establish how the Quipu model can be improved. In previous work, we have identified several areas of improvement and in this section we describe these in more detail.

A. Software Metrics

As Quipu is based on capturing software complexity, the quality of the different software complexity metrics has a substantial influence on the quality of the model. Previously, we used 24 SCMs, without precisely evaluating their individual qualities. We identify four possible improvements to our use of SCMs.

- **Additional Metrics**

One obvious improvement to our previous model is the addition of new metrics. Already we have identified 24 additional metrics concerning basic blocks, operations, and fractals[11]. Also, Quipu currently doesn't support floating points. For future versions of the Quipu model, floating point metrics are essential.

- **Selection of metrics**

The statistical methods that we use do not automatically discard any metric that may corrupt the Quipu model. It is therefore important to look at methods to choose a subset of metrics that is relevant to the model. One possibility is looking at Partial Least Squares Regression[12], a method that finds linear components in the independent variables that are also relevant to the dependent variables.

- **Metric Transformations**

Not every metric necessarily relates to hardware cost in a linear fashion. It is therefore important to individually observe this relation for each software metric. As an example, the number of paths in a kernel grows exponentially with the number of subsequent loops, while hardware area is expected to grow in a more (sub)linear fashion. In this case we can apply a *log*-transformation.

- **Advanced Linear Modeling Techniques**

Traditional linear modeling techniques, assume that independent variables are continuous ratio scales. However, this assumption does not hold for many metrics. More precise, some metrics are discrete measures or even categorical by nature. Using advanced linear modeling techniques, like Probit, Logit, or Poisson Regression[13], these kinds of measures can be modeled more correctly.

B. Kernel Representation

Although capturing software complexity from a high level description like C makes early prediction possible, information on optimizations is not available. Furthermore, restricting prediction to applications written in C prohibits designers to use other high level languages more suitable to a problem. For example, a mathematical routine might be written in Fortran, or a computer game in an object oriented language like C++.

In order to solve both these problems, measurements of software complexity could be done on a commonly used intermediate representation. For example, we could use the GIMPLE intermediate representation used in the newest generation of the GNU Compiler Collection (GCC)[14]. Because GCC has many language front-ends, we would be able to estimate hardware costs for various languages. Furthermore, optimization passes are often defined on the intermediate representation, which allows us to estimate after optimizations are performed, instead of estimating the actual optimizations. The drawback, however, is that the speed of estimation will deteriorate. It is therefore important to choose those optimizations that also improve hardware generation.

C. Kernel Library

As mentioned, Quipu is currently based on a kernel library of 135 kernels. Table II shows the different application domains that are present in the library, as well as several characterizations of the code present in those domains. We can deduce several points of improvement from this overview.

- **Size of the Library**

First and foremost, we can increase the accuracy of the Quipu model by building our model from a larger set of kernels. Especially, when the number of independent variables increases, the number of observations need to grow accordingly. We can illustrate this need

Domain	Kernels	Bit-Based	Streaming	Account-Keeping[a]	Control-Intensive	Avg. State-ments
Compression	2	x		x	x	21
Cryptography	56	x	x		x[b]	169
DSP	5	x	x		x[b]	28
ECC	6	x	x		x	110
Mathematics	19					22
Multimedia	32	x[b]	x		x	54
General	15			x[b]	x	35
Total	135					98

[a] Non-constant space complexity.
[b] Only some instances in that domain express this characteristic.

TABLE II
OVERVIEW OF THE COLLECTION OF FUNCTIONS.

by looking at the model quality indicator *Adjusted* R^2. This value captures the variance in the data explained by the model adjusted for the number of variables and observations,

$$R_A^2 = 1 - (1 - R^2)c_{adjust}, c_{adjust} = \frac{N-1}{N-k-1} \quad (1)$$

where N is the number of observations and k is the number of independent variables. The more c_{adjust} deviates from 1.0 the less the quality will be from the ideal. For our model this factor is 1.61 lowering R^2 from 92.0% to 87.1%. In order keep the deterioration within 5% or 10%, we need at least 1008 or 518 observations respectively.

- **Balance in Application Domains**

Just increasing the number of observations might not be a good idea. Already the table shows the kernel library has more kernels in some application domains than others. As the use of language features and design patterns may differ per application domain, overrepresenting one domain may make predictions for this domain better, while deteriorating predictions for another one. Therefore, a kernel library with a balanced distribution of kernels over application domains is preferable.

- **Balance in Code Size**

Analogously, the small kernels are overrepresented in the current Quipu model. As an illustration, the number of operators in the C code ranges from 0–12000 per kernel, while 100 (or almost 80%) of the kernels have less than 1000 operators. This is in violation of the Homoscedasticity assumption, required for linear regression.

- **Relax Code Restrictions**

Outside these observed points of improvement, the code restrictions in the library can be relaxed. This facilitates building models for different tool chains with different restrictions. The drawback for loosening the restrictions is that the impact of code rewriting affects Quipu's quality.

D. Predicted Hardware

Currently, Quipu predicts several area cost measures, like Slices, Flip-Flops, or LUTs. However, there are other hardware characteristics essential for DSE:

- **Delay / Speed-up**

An important question for selecting kernels for hardware implementation is: *how much speed-up can we attain?*. Delay estimation would therefore be a valuable addition to our model. However, there are some difficulties in establishing the delay of a circuit; in case of highly data dependent kernels the delay may vary. Therefore, we envision a delay model that uses observations from dynamic profiling with representative input data as independent variables. Furthermore, the delay measurements require not only synthesis but also simulation with the representative input data.

- **Interconnect**

Another factor to be considered is the number of resources dedicated to interconnection. Especially on FPGAs the number of available wires is limited. Kernels that use a huge amount of interconnect can inhibit the use of other kernels, although enough computational components might be available. For accurate interconnect estimates, observations must be determined after the place and route phase in synthesis.

- **Throughput**

Delay is not the only hardware performance indicator; in streaming applications, for example, the throughput is the main point of interest. As with delay estimation, dynamic profiling and simulation are necessary to make accurate predictions. Problems may arise when part of a kernel's input and output data are accessed via pointers, which can make data dependency size estimation difficult.

- **Power**

Another important hardware cost measure in modern computing systems is power dissipation. Where in high performance computing, servers, and desktop computing heat is a growing problem, mobile applications have a limited power supply available. As with delay, power requires dynamic profiling information and simulation, and additionally a power model is re-

quired during simulation.

E. Other Issues

Apart from improving the quality and applicability of the Quipu model, there are also several more general elements that need attention in the future.

- **Integration**

Integrating Quipu in existing tool chains like Delft Workbench is essential to make semi-automatic DSE possible. Quipu should provide output in formats required by the tool-chain. Both integration with other DSE tools and tools from subsequent passes should be considered. Also feed back from later passes can be used to improve Quipu's prediction.

- **Modeling Tool**

Because Quipu is targeted at a specific tool-chains, an automatic modeling tool resulting in a tailored model can make our estimation scheme available to other tool chains.

- **Quality Assessment**

Apart from error analysis, it is important to test the Quipu model in real DSE situations. For example, how does it compare with other approaches when driving hardware-software partitioning.

V. Conclusion

We presented our Multi-Dimensional Quantitative Model for Early Design Space Exploration called Quipu. We conclude that fast and early prediction of hardware characteristics is important, but our approach was not accurate enough in the past. While a somewhat larger error is acceptable in the early stages of design, we need to improve on the current Quipu model. Furthermore, for Quipu to be useful in a production tool chain, it must predict additional hardware measures for a wider range of application domains.

Acknowledgment

Delft WorkBench is sponsored by the hArtes (IST-035143), the MORPHEUS (IST-027342), and RCOSY (DES-6392) projects.

References

[1] Y. D. Yankova, K. Bertels, S. Vassiliadis, R. J. Meeuws, and A. Virginia, "Automated hdl generation: Comparative evaluation," in *Proceedings of International Symposium on Circuits and Systems (ISCAS2007)*, May 2007.

[2] J. Becker and R. Hartenstein, "Configware and morphware going mainstream," *J. Syst. Archit.*, vol. 49, no. 4-6, pp. 127–142, 2003.

[3] Delft workbench. Delft, The Netherlands. [Online]. Available: http://ce.et.tudelft.nl/DWB/

[4] R. J. Meeuws, "A quantitative model for hardware/software partitioning," Master's thesis, Delft, Netherlands, May 2007.

[5] F. Vahid and D. D. Gajski, "Incremental hardware estimation during hardware/software functional partitioning," in *Readings in hardware/software co-design*. Norwell, MA, USA: Kluwer Academic Publishers, 2002, pp. 516–521.

[6] D. Kulkarni, W. A. Najjar, R. Rinker, and F. J. Kurdahi, "Fast area estimation to support compiler optimizations in fpga-based reconfigurable systems," in *FCCM '02: Proceedings of the 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*. Washington, DC, USA: IEEE Computer Society, 2002, p. 239.

[7] W. Fornaciari, F. Salice, and D. P. Scarpazza, "Early estimation of the size of vhdl projects," in *Proceedings of the first IEEE/ACM/IFIP Intl. Conf. on Hardware/Software Codesign and System Synthesis*. Milano, Italy: Politecnico di Milano, 2003, pp. 207–212.

[8] M. Holzer and M. Rupp, "Static estimation of execution times for hardware accelerators in system-on-chips," in *System-on-Chip, 2005. Proceedings. 2005 International Symposium on*, 2005, pp. 62–65.

[9] R. J. Meeuws, Y. D. Yankova, K. Bertels, G. N. Gaydadjiev, and S. Vassiliadis, "A quantitative prediction model for hardware/software partitioning," in *Proceedings of 17th International Conference on Field Programmable Logic and Applications (FPL'07)*, August 2007, p. 5.

[10] E. M. Panainte, "The molen polymorphic processor," *IEEE Trans. Comput.*, vol. 53, no. 11, pp. 1363–1375, 2004, fellow-Stamatis Vassiliadis and Member-Stephan Wong and Member-Georgi Gaydadjiev and Member-Koen Bertels and Student Member-Georgi Kuzmanov.

[11] P. Kokol and J. Brest, "Fractal structure of random programs," *SIGPLAN Notices*, vol. 33, no. 6, pp. 33–38, 1998.

[12] A. Höskuldsson, "Pls regression methods," *Journal of Chemometrics*, vol. 2, no. 3, pp. 211–228, 1988.

[13] J. S. Long, *Regression Models for Categorical and Limited Dependent Variables (Advanced Quantitative Techniques in the Social Sciences)*. Sage Publications, 1997.

[14] J. Merrill, "Generic and gimple: A new tree representation for entire functions," in *Proceedings of the 2003 GCC Summit*. Red Hat, Inc., 2003. [Online]. Available: http://dept-info.labri.fr/~fleury/Stage_Master2-06/papers/GENERIC.ps