A QUANTITATIVE PREDICTION MODEL FOR HARDWARE/SOFTWARE PARTITIONING

Roel Meeuws, Yana Yankova, Koen Bertels, Georgi Gaydadjiev, Stamatis Vassiliadis

Computer Engineering
Delft University of Technology
Delft, Netherlands
email: {r.j.meeuws, y.d.yankova, k.l.m.bertels, g.n.gaydadjiev, s.vassiliadis}@tudelft.nl

ABSTRACT

An important step in Heterogeneous System Development is Hardware/Software Partitioning. This process involves exploring a huge design space. By using profiling to select hot-spots and estimate area and delay we can prune the design space considerably. We present a Quantitative Model that makes early predictions to prune the design space and support the partitioning process. The model is based on Software Complexity Metrics, which capture important aspects of functions as control intensity, data intensity, and code size. To remedy interdependence among software metrics, we performed a Principal Component Analysis. The hardware characteristics were determined by automatically generating VHDL from C using the DWARV C-to-VHDL compiler. Linear regression on these data generated our model. The model error differs per hardware characteristic. We show that for flip-flops the mean error is 69%. In conclusion, our quantitative model makes fast and sufficiently accurate area predictions in support of early Hardware/Software Partitioning.

1. INTRODUCTION

In the last few decades computing systems have become increasingly heterogeneous. Apart from general purpose processors these systems contain such elements as ASICs, DSPs, and FPGAs. Although, these systems offer several advantages over conventional computing systems - e.g. providing application specific hardware, reconfigurability, and a wider selection of COTS components - they lack the design tools to support the development process.

Delft Workbench, a tool platform in support of integrated hardware/software co-design targeted at these systems, aims to fill this gap starting from profiling and partitioning to synthesis and compilation. Some major difficulties of hardware/software co-design in the early stages of development are the absence of hardware characteristics and the size of the design space. 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.

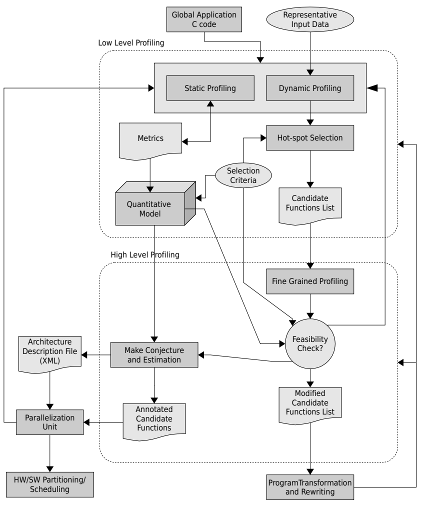
Global Application C code
Representative Input Data
Low Level Profiling
Static Profiling
Dynamic Profiling
Metrics
Hot-spot Selection
Selection Criteria
Quantitative Model
Candidate Functions List
High Level Profiling
Fine Grained Profiling
Feasibility Check?
Architecture Description File (XML)
Make Conjecture and Estimation
Modified Candidate Functions List
Parallelization Unit
Annotated Candidate Functions
HW/SW Partitioning/ Scheduling
ProgramTransformation and Rewriting

Fig. 1. Work flow of code profiling in Delft Workbench

In the past, high level estimation was often based on synthesis-like schemes as allocation and binding, and scheduling (e.g.[1]). The main drawback of this approach was the high computational complexity of these algorithms. Furthermore, C–level specifications do not easily lend themselves to synthesis algorithms, although some schemes targeted C-derivatives like SA-C[2].

In contrast, we introduce here a High Level Estimation scheme to support Hardware/Software Partitioning based on a quantitative model, as presented in Fig. 1. The basic idea is to build a model through linear regression on software complexity metrics (SCM), that predicts characteristics as area and delay. Using SCMs allows fast predictions even during the early stages of development. For a preparatory study of such a model, we refer the reader to [3].

The paper is organized as follows. In Section 2, we briefly discuss relevant research. Then, in Section 3, we present the criteria of our model, followed by the experimental setup in Section 4. In Section 5, we present the results of the statistical analysis. Finally, we present our conclusions in Section 6.

2. RELATED RESEARCH

Early prediction approaches have been presented before. In [4] we find a constant time incremental estimation approach targeted at iterative hardware/software partitioning, where at each partitioning step the estimates are updated. More similar to our approach, [2] estimates area by using linear regression models per DFG node. Different DFG nodes are characterized by different linear models. However, these approaches do not operate on C-level specifications as our model does.

3. CRITERIA OF EVALUATION

As mentioned earlier, we envision a model based on SCMs. These metrics capture certain aspects of computer programs and functions as 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 advantages of using SCMs is that several independent variables may already be available. Additionally, when metrics are not available, most metrics are inherently simple and can be determined in a relatively short time.

In the context of our work we used 24 SCMs, as listed in Table 1. The metrics in the table come from their corresponding publications, where mentioned. Some have been modified, some are introduced for the first time.

- **Cumulative Nesting Depth** - Because the maximum nesting depth characterizes only a part of a function, we introduce the Cumulative and Average Nesting Depth.The former is the sum of all nesting depths over all basic blocks:

$$\mathtt{CUMNEST} = \sum_{i=0}^{i=n_b} \mathtt{depth}(\mathtt{b_i}) \qquad (1)$$

  where n_b is the number of basic blocks and b_i is basic block i.

- **Average Nesting Depth** - This metric averages the nesting depth of each basic block in the function:

$$\mathtt{AVGNEST} = \mathtt{CUMNEST}/n_b \qquad (2)$$

- **(Modified) Basili-Hutchens Complexity** - This metric is based on structured and unstructured code, where the latter gets a larger penalty. Because in our expectation loops imply more hardware than if-statements, we defined if-statements to be structured and loops to be unstructured control constructs.

- **Loop Complexity** - This new metric captures the fact that nested loops often loop over the same code and would therefore represent less increase in hardware as consecutive loops. The following recursive definition of the Loop Complexity is therefore multiplicative for nesting and additive for sequencing of loops:

$$\begin{aligned} &1)\mathtt{LOOPCOM}(F(\emptyset)) &&= 1 \qquad (3)\\ &2)\mathtt{LOOPCOM}(F_1;\cdots;F_n) &&= \sum_{i=1}^{n} \mathtt{LOOPCOM}(F_i)\\ &3)\mathtt{LOOPCOM}(F(F_1,\cdots,F_n)) &&= 1.1\mathtt{LOOPCOM}(F_1;\cdots;F_n)\end{aligned}$$

  where 1) represents a loop that doesn't contain another loop, 2) represents consecutive loops, and 3) represents nested loops.

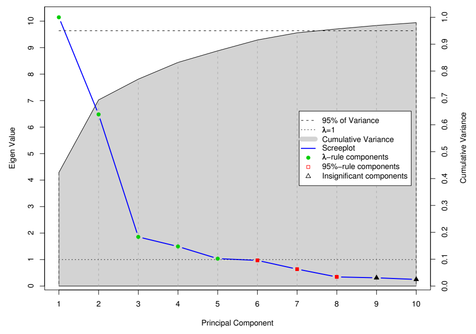

Fig. 2. Summary of PCA with Scree-plot and Variance, showing 12 PCs.

Together, these metrics capture a wide range of code characteristics. In Section 5, we will see that some of them correlate with hardware. However, SCMs also correlate with each other. This phenomenon is called interdependence or multicollinearity. For example, the Average Path Length, Maximum Path Length, and Statements all measure program length. Because linear regression requires independent variables, we transformed the set of metrics by applying Principal Component Analysis (PCA). In Fig. 2, we see the summary of this 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.

In principle, the scope of our model is to predict area and delay on the MOLEN platform [15]. Because this is a reconfigurable platform, delay also comprises reconfiguration delays. Nevertheless, in this paper we will restrict to results for our area prediction model. Delay was omitted for the time being, because there was not enough time to simulate 126 kernels.

The MOLEN platform as we used it, contains a Xilinx Virtex-II Pro. Therefore, we will use the following area characteristics, which are basic elements in Xilinx FPGAs.

Metric	PC 1: Length	PC 2: Control	PC 3: Nesting	PC 4: Diversity	PC 5: Data	PC 6: Volume	PC 7: Loops	PC 8: Paths
AICC[5]	0.011	−0.008	-0.194	**-0.821**	−0.017	0.001	0.021	0.129
Avg. Nesting Depth	−0.043	−0.025	**0.633**	−0.009	0.009	−0.006	−0.047	0.061
Avg. Path Length	**0.370**	0.014	−0.005	−0.016	−0.010	−0.012	−0.020	−0.067
Basili-Hutchens[6]	**0.364**	−0.058	0.015	0.007	−0.019	−0.001	0.049	0.033
Cum. Nesting Depth	0.008	**-0.373**	−0.017	−0.013	0.202	0.005	0.029	0.224
Cyclomatic(McCabe)[7]	0.031	**-0.411**	0.028	−0.001	−0.030	0.013	0.015	-0.128
Gong and Schmidt[8]	0.029	**-0.410**	0.040	−0.001	−0.030	0.013	0.016	-0.121
Loads	0.001	0.007	0.008	0.005	−0.010	**0.569**	0.013	−0.024
Loop Complexity	−0.024	−0.064	−0.034	−0.005	−0.011	−0.017	**0.957**	0.020
Max. Nesting Depth	−0.042	−0.042	**0.605**	−0.014	−0.001	−0.001	0.035	−0.004
Max. Path Length	**0.370**	0.026	−0.045	−0.012	−0.007	−0.001	−0.010	−0.015
NPATH[9]	−0.046	-0.131	−0.077	−0.024	0.009	−0.020	−0.017	**-0.900**
Oviedo def-use pairs[10]	0.212	0.278	0.028	0.050	**0.368**	0.094	0.122	-0.182
Piwowarski[11]	0.023	**-0.378**	0.128	0.028	−0.022	0.020	0.139	0.032
Prather's mu[12]	0.345	−0.018	0.213	0.049	0.015	−0.001	−0.014	−0.009
Scope Number[13]	**0.372**	−0.020	−0.020	−0.002	−0.017	−0.006	−0.006	0.015
Statements	**0.352**	−0.049	−0.042	−0.010	0.027	0.005	−0.026	0.056
Stores	**0.372**	0.023	−0.028	−0.005	0.002	0.002	0.014	−0.001
Tai def-use pairs[14]	0.010	**-0.517**	-0.170	0.007	0.046	−0.013	-0.193	0.106
Variable Declarations	-0.129	−0.006	−0.071	0.007	**0.697**	0.017	0.009	−0.006
Operands	0.025	−0.003	−0.011	−0.001	0.021	**0.534**	−0.005	0.009
Operators	−0.046	−0.037	−0.002	−0.009	−0.029	**0.613**	−0.015	0.025
Unique Operands	0.025	−0.057	0.063	−0.029	**0.573**	−0.060	−0.055	0.052
Unique Operators	0.012	0.031	0.284	**-0.564**	0.032	0.006	−0.020	-0.172
Eigenvalue	10.145	6.479	1.854	1.494	1.036	0.969	0.640	0.344
Variance	0.423	0.270	0.077	0.062	0.043	0.040	0.027	0.014
Cum. Variance	0.423	0.693	0.770	0.832	0.875	0.916	0.942	0.957

Table 1. Results of Principal Components Analysis with VARIMAX Rotation of the 24 Software Metrics considered.

Domain	Kernels	Bit-Based	Streaming	Account-Keeping[a]	Control-Intensive
Compression	2	x		x	x
Cryptography	56	x	x		x[b]
DSP	5	x	x		x[b]
ECC	6	x	x		x
Mathematics	19				
Multimedia	32	x[b]	x		x
General	15			x[b]	x
Total	135				

[a]Non-constant space complexity.

[b]Only some instances in that domain express this characteristic.

Table 2. Overview of the collection of functions.

- **Flip-Flops** - These are D-type Flip-Flops.
- **Look-Up Tables (LUTs)** - These are 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** - These are 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 generated VHDL. The used VHDL compiler and synthesizer with their respective optimization options affects the behavior and quality of the model. Currently, the model is based on a specific set of tools. However, in the future we envision an automatic model generator for other tool sets.

4. EXPERIMENTAL SETUP

To test our hypothesis that SCMs correlate with hardware characteristics and build a model that can predict those characteristics, we have followed the following procedures.

First, a set of 135 C-functions was collected domains, as summarized in Table 2. By using source code from existing applications from many different areas in computing, we aim to build a model that is generally applicable.

For the purpose of gathering SCMs from this set of functions we built a metrication tool based on the Elsa/Elkhound compiler front-end from UC Berkeley, which then generated the dataset of metrics from the gathered functions. On a 2.4GHz AMD Athlon64 this process took 7.5 seconds in total. Then a PCA was performed, which yielded the eight components in Table 1. The data were then transformed into PC scores for use in the linear regression analysis.

We then used the DWARV [16] C-to-VHDL compiler to translate all 135 kernels to VHDL. The VHDL code was then synthesized using the Xilinx ISE Synthesizer. Six kernels did not compile and two more kernels were not synthesizable. Thus we collected 127 observations for linear regression. Finally, we performed a linear regression analysis using the area metrics as dependent variables and the PC scores as independent variables.

5. EXPERIMENTAL RESULTS

In this section we present the different area models based on SCMs and evaluate their performance. For this purpose, we will first present several graphs that illustrate the predictive capabilities of the models and then we will assess the quality of the models using traditional performance indicators.

In Fig. 3, we find a clear relation between the predicted and the actual number of slices. There are however two outliers in the top-left corner of the plot. One out-lier makes heavy use of expression lists, which our metrication tool does not account for at the moment. The other one makes heavy use of constant expressions, which makes Halstead's

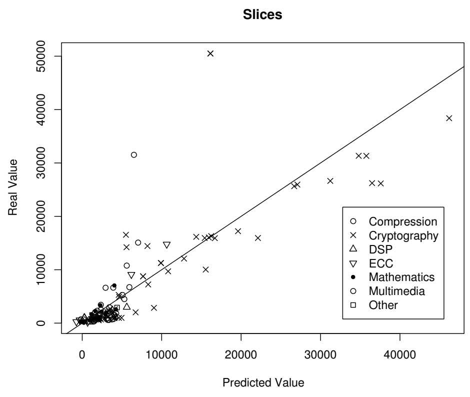

Fig. 3. Actual vs. predicted no. of slices per app. domain.

metrics unrepresentative. Another observation we make is that some application domains occupy specific areas in the graph. Especially, the Cryptography functions dominate the upper right part of the graph. A possible implication of this behavior is that different application domains may need different models.

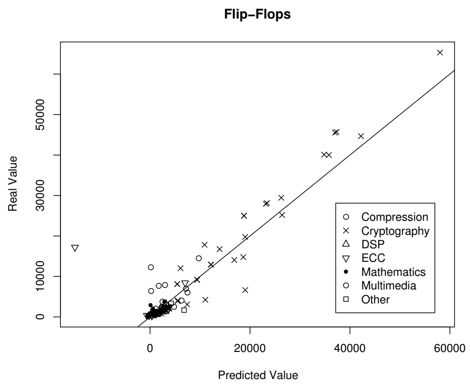

Fig. 4. Actual vs. predicted no. of flip-flops per app. domain.

In Fig. 4, we also observe a clear relation between the predicted and actual number of flip-flops. The out-lier from the ECC domain on the left side is a routine that performs the butterflies in a Viterbi codec. It is not clear why the model mispredicts this particular point.

As can be observed in Fig. 5, the behavior of the model for LUTs is quite similar to the one for slices. Apparently, slice usage is dictated by the number of used LUTs, instead of flip-flops. The outliers correspond to those of the model for slices.

From Fig. 6 we deduce our current linear model does not predict the number of multipliers in any way. One of the reasons is that the number of multiplications in the code was not yet included as an SCM. Another reason is that many zeroes occur in the data for this dependent variable. Therefore, a traditional linear model might not be a good choice.

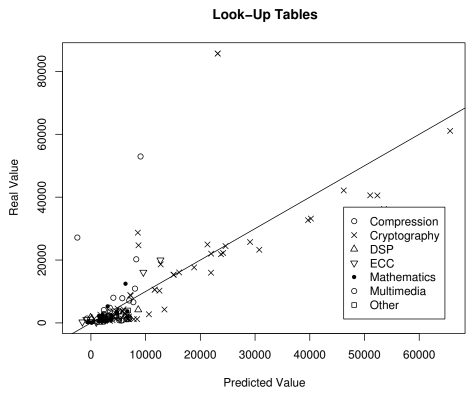

Fig. 5. Actual vs. predicted no. of luts per app. domain.

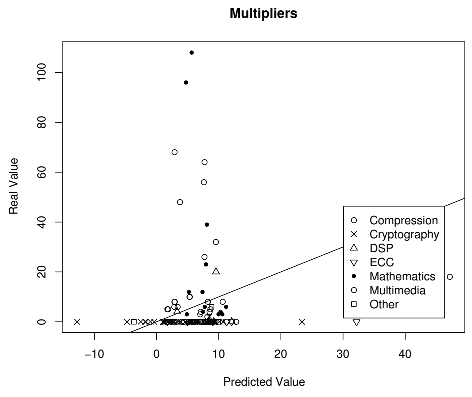

Fig. 6. Actual vs. predicted no. of multipliers per app. domain.

Instead, we may need to employ a General Linear Model approach in the future.

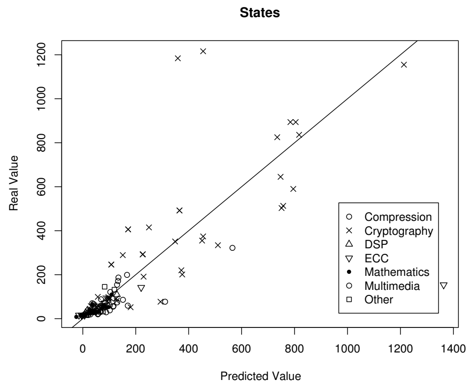

Fig. 7. Actual vs. predicted no. of states per app. domain.

For the number of states we see again a similar picture, depicted in Fig. 7. There are two outliers - i.e. cryptography functions - in the top-left corner, that are mispredicted because of their very large loop-bodies. In the bottom-left corner we observe the exact same out-lier as in the flip-flop

Variable	**R2!**	RMSE$\%_{\texttt{fit}}$	RMSE$\%_{\texttt{PRESS}}$	p-value
Slices	0.717	85.0%	88.93%	< 2.2e-16
Flip-Flops	0.920	47.2%	69.2%	< 2.2e-16
LUT!s	0.628	101.8%	108.4%	< 2.2e-16
Multipliers	0.047	287.5%	296.5%	0.6732
Period	0.095	29.8%	30.1%	0.1458
States	0.795	70.2%	101.7%	< 2.2e-16

Table 3. Performance indicators for the several dependent variables in the linear model for **DWARV!**.

model.

In Table 3, we find indicators of the quality of our models. The table shows the ratio of variance explained by the models (R^2), the expected error in the original dataset ($RMSE_{fit}$) and on new data ($RMSE_{PRESS}$), and the chance that the model does not explain anything (p-value). Apart from the multipliers-model the models appear to perform reasonably well. The Flip-Flop model is the best of these models.

The reported error margins between 69.2% and 108.4% are less accurate than other schemes like in [2] (10%) and [4] (7%). On the other hand our model operates at C-level and is very fast (7.5s for 135 kernels). Furthermore, the prediction plots show that error occurs mostly at the small kernels.

6. CONCLUSIONS

We have seen that a linear model based on SCMs can estimate the number of slices, flip-flops, look-up tables, and states within reasonable bounds in the early stages of development. Furthermore, such a model can make predictions in a relatively short time, as required in a hardware/software partitioning context. Although there is a considerable error involved, the model is precise enough to prune the design by removing functions that are too large to fit on an FPGA or too small to exploit any degree of parallelism.

In the future, we will incorporate more metrics in our model and use more advanced modelling techniques like Generalized Linear Regression. Furthermore, specific models for different platforms and optimizations could be considered.

7. ACKNOWLEDGEMENTS

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

8. REFERENCES

[1] A. Sharma and R. Jain, "Estimating architectural resources and performance for high-level synthesis applications," in *DAC '93: Proceedings of the 30th international conference on Design automation*. New York, NY, USA: ACM Press, 1993, pp. 355–360.

[2] 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.

[3] R. J. Meeuws, Y. D. Yankova, and K. Bertels, "Towards a quantitative model for hardware/software partitioning," Delft University of Technology, Delft, Netherlands, Delft, Netherlands, Tech. Rep. RCOSY DES.6392, April 2006.

[4] 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.

[5] W. Harrison, "An entropy-based measure of software complexity," *IEEE Trans. Softw. Eng.*, vol. 18, no. 11, pp. 1025–1029, 1992.

[6] V. R. Basili and D. H. Hutchens, "An empirical study of a syntactic complexity family." *IEEE Transactions on Software Engineering*, vol. 9, no. 6, pp. 664–672, 1983.

[7] T. J. McCabe, "A complexity measure," in *ICSE '76: Proceedings of the 2nd international conference on Software engineering*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1976, p. 407.

[8] H. Gong and M. Schmidt, "A complexity measure based on selection and nesting," *SIGMETRICS Perform. Eval. Rev.*, vol. 13, no. 1, pp. 14–19, 1985.

[9] B. A. Nejmeh, "Npath: a measure of execution path complexity and its applications," *Commun. ACM*, vol. 31, no. 2, pp. 188–200, 1988.

[10] E. I. Oviedo, "Control flow, data flow, and program complexity," in *COMPSAC'80: Proceedings of the Fourth International Computer Software and Applications Conference*.

[11] P. Piwowarski, "A nesting level complexity measure," *SIGPLAN Not.*, vol. 17, no. 9, pp. 44–50, 1982.

[12] R. E. Prather, "An Axiomatic Theory of Software Complexity Measure," *The Computer Journal*, vol. 27, no. 4, pp. 340–347, 1984.

[13] W. Harrison and K. Magel, "A topological analysis of the complexity of computer programs with less than three binary branches," *SIGPLAN Not.*, vol. 16, no. 4, pp. 51–63, 1981.

[14] K.-C. Tai, "A program complexity metric based on data flow information in control graphs," in *ICSE '84: Proceedings of the 7th international conference on Software engineering*. Piscataway, NJ, USA: IEEE Press, 1984, pp. 239–248.

[15] E. Moscu Panainte, S. Vassiliadis, S. Wong, G. Gaydadjiev, K. Bertels, and G. Kuzmanov, "The molen polymorphic processor," *IEEE Trans. Comput.*, vol. 53, no. 11, pp. 1363–1375, 2004.

[16] 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.