CC4CS: an Off-the-Shelf Unifying Statement-Level Performance Metric for HW/SW Technologies

Vittoriano Muttillo
Università degli Studi dell'Aquila
vittoriano.muttillo@graduate.univaq.it

Giacomo Valente
Università degli Studi dell'Aquila
giacomo.valente@univaq.it

Luigi Pomante
Università degli Studi dell'Aquila
luigi.pomante@univaq.it

Vincenzo Stoico
Università degli Studi dell'Aquila
vincenzo.stoico@graduate.univaq.it

Fausto D'Antonio
Università degli Studi dell'Aquila
fausto.dantonio@graduate.univaq.it

Fabio Salice
Politecnico di Milano
fabio.salice@polimi.it

ABSTRACT

Outlining the general characteristics of embedded systems is an arduous task. In fact, the design of such kind of systems is heavily influenced by functional and non-functional requirements, and it is based on quite complex design flows. So, there is often the need to adopt a HW/SW co-design methodology able to support the designers during high-level phases so that they can perform early analysis before dealing with low-level ones. Such a methodology, to be effective, should consider also performance estimation and ESL HW/SW timing co-simulation. The goal of this paper is to introduce a novel and fast performance metric able to speed-up the early analysis and design space exploration to identify the more promising architectures for different application domains. In particular, the paper presents a framework to evaluate such a metric and to perform some preliminary analysis to evaluate its meaningfulness when exploited in the HW/SW domain.

ACM Reference Format:
Vittoriano Muttillo, Giacomo Valente, Luigi Pomante, Vincenzo Stoico, Fausto D'Antonio, and Fabio Salice. CC4CS: An Off-the-Shelf Unifying Statement-level Performance Metric for HW/SW Technologies. In *ICPE'18 Companion: ACM/SPEC International Conference on Performance Engineering, April 9-13, 2018, Berlin, Germany, USA.* ACM, NY, NY, USA, 4 pages. DOI: http://doi.org/10.1145/3185768.3186291

1 INTRODUCTION

In the last thirty years there has been an exponential increase of the exploitation of embedded systems in everyday life. Due to their HW/SW heterogeneity and the critical impact of non-functional constraints, the adoption of a HW/SW co-design methodology is a key factor for a successful development. In such a context, early performance estimation and HW/SW timing co-simulation are always crucial steps. In such steps, the availability of performance metrics suitable for both HW and SW technologies and with a low computational complexity are fundamental in the early design stages. It is worth noting that a low computational complexity usually induces a reduction in the accuracy of the metric; however, this accuracy decay provides the undeniable advantage of allowing a preliminary evaluation of the architectural spaces and the identification of a solutions subspace.

One of the most known assembly-level metrics for SW performance estimation are MIPS (*Millions of Instructions per Second*), CPI (*Clock Cycles per Instructions*) or IPC (*Instructions per Clock Cycles*) [1]. Among them, at least MIPS can be considered as off-the-shelf, since it is normally available on data-sheets even if it is not so clear how standard is the evaluation process and what are its statistical characterization (e.g. precision and accuracy). Unfortunately, although MIPS, CPI and IPC can be considered an efficient way to compare different microprocessors with the same ISA (*Instruction Set Architectures*), they are ineffective when different ISA have to be compared. Finally, these metrics are SW oriented (assembly-based) and, due to this, they are not applicable in the HW domain. So, for all the reasons above discussed, MIPS, CPI and IPC cannot be considered a useful evaluation means in the field of Electronic System-Level (ESL) HW/SW co-design methodologies.

In such a context, the goals of this work are to analyze the usefulness and the meaningfulness of an innovative performance metric that is concurrently "Off the Shelf", "HW/SW Unifying", and "Statement Level". In fact, to overcome existing metrics limitations, the idea is to consider one related to ***Clock Cycles for C Statement*** (CC4CS), i.e. the number of clock cycles needed to a specific processor technology to execute a *generic C statement.* So, it is would be at statement-level of abstraction and, thanks to even more improved *High-Level Synthesis* (HLA) tools that are able to synthesize C functions, it would be targeted to both SW and HW processor technologies (i.e. *HW/SW unifying*): processors built to execute a given ISA (*General Purpose Processors*, GPP; *Application Specific Processors*, ASP) and processors built to directly (i.e. NO ISA involved) execute applicative functions (*Single/Specific Purpose Processors*, SPP). So, such a metric would be an ideal one for the very early steps of an ESL HW/SW Co-Design Methodology but also for the comparison of SW implementation performances. However, some critical issues soon arise when thinking with more attention to CC4CS. First of all, the concept of *generic C statement* is ambiguous, since a C statement is not a-priori limited in complexity and can give rise to very different HW/SW implementations. Second, to evaluate it in a standard, repeatable, fast and low-cost way, they are needed an evaluation framework and a meaningful set of benchmark functions. The first point can be addressed by considering as "generic C statements" the most common way a programmer writes them (so it is better to talk about *common C statements*), and this consideration should drive the selection of the adopted benchmark. An encouraging precedent can be considered the work done for the definition of the very first (and successful) COCOMO model [2] where, by analyzing a very huge set of source codes, a relationship by the number of *Lines of Code* (LOC) and the SW development cost has been identified, independently by the complexity of each line. The second point, other than identifying a relevant benchmark, can be addressed by designing a proper framework for CC4CS evaluation. So, this work mainly focuses on the development of such a framework and, by means of a simple benchmark, tries to evaluate usefulness and meaningfulness of CC4CS to understand if further effort must be invested in such a direction. For this, Sections II and III define the metric, the framework, and the adopted benchmark, while Sections IV and V evaluate CC4CS both in the SW and HW domains also analyzing possible exploitation opportunities. Finally, Section VI tries to understand if, on the base of the obtained results, such a metric could have a future or not.

2 DEFINITION OF CC4CS

The proposed metric is related to C programming language statements, so it is called CC4CS (*Clock Cycles for C Statement*). The choice of the C language is motivated by the following three reasons: it is the most used language for embedded SW development; it is very similar to *SystemC* [3] (especially when focusing on *SystemC Synthesizable Subset*), one of the most used specification languages for HW/SW co-design; the most diffused HLS (*High Level Synthesis*) tools are able to realize SPPs that implements an algorithm specified in C/SystemC language. So, CC4CS is defined as follow:

Def. *For a given processor X, CC4CS(X) is the number of clock cycles needed by processor X to execute a common C statement*

A first clarification is due with respect to the concept of "common C statement". It could be generally intended as "something that ends with a semicolon" (other views are possible too, e.g. Table 6.1 in [4]) but, to avoid ambiguity, this work adopts an empirical approach: it refers to the way a common profiling tool as gcov [5] performs the C statements identification when profiling their execution. Another clarification is related to the fact that such a metric will be for sure influenced by the used compiler or HLS tool. Some ways to manage this issue could be: to specify also the used tools (possibly giving rise to a set of CC4CS for each processor); to report the average of the results obtained by using the most diffused tools; to report only the results related to the most diffused one. At this point, it is quite clear that CC4CS, as defined above, will be influenced by several factors and that a CC4CS-based estimation will be affected by relevant errors. However, these are acceptable by keeping in mind the following aspects: it is a straightforward way to have an off-the-shelf metric; it can be applied to each processor technology (i.e. GPP, ASP and SPP); it is intended to be used for very early performance analysis in SW and HW/SW domains. Anyway, as described in the next sections, CC4CS can be also characterized by a set of values related to *Min, Max, Average*, and *Standard Deviation* (or by a statistical distribution). In this way, it is possible to perform different analysis depending on the final goal.

3 CC4CS EVALUATION FRAMEWORK

Starting from the definition provided before, to evaluate CC4CS for a given processor technology there is the need for a methodology supported by tools to allow fast and repeatable operations. In fact, as already said, considering a single C function, CC4CS is the ratio between the number of clock cycles required by the target processor technology to execute the function and the number of executed C statements:

CC4CS = #Required_Clock_Cycles / #Executed_C_Statements.

So, to make the metric meaningful for a given processor it is needed, at least, to: define a set of relevant C functions to be used as benchmark for all the processor technologies; for each benchmark function to identify a way to stimulate (i.e. execute) it by means of relevant input data sets; to identify a tool to perform profiling in order to count the number of executed C statements for each input; to identify tools to compile/synthesize the C function for the target processor and to simulate its execution in order to obtain the number of clock cycles needed for the on-target execution. Naturally, such steps must be applied for each different processor technology. However, it is worth noting that it is an offline one-shot task since CC4CS, once evaluated, would be available "for free" for next projects. So, to support CC4CS evaluation, a proper framework has been developed. Additionally, such a framework is also able to evaluate statistics on the metric. A simple benchmark composed of 10 well-known algorithms (i.e. C leaf functions) has been realized. The functions of the benchmark are the following ones: *Quicksort, Mergesort, Matrix Multiplication, Kruskal, Floyd-Warshall, Dijkstra, Breadth First Search, Depth First Search, Banker's Algorithm, A**. The source code is available on [11]. The following paragraphs describes the main features of the generic framework.

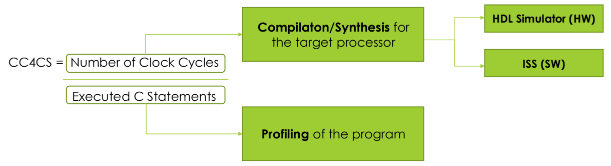

Figure 1: CC4CS Evaluation Framework.

3.1 Input Generation

To evaluate CC4CS, a module that (semi)automatically generates inputs for the benchmark functions has been created. In particular, for each function they have been randomly generated 1000 input data sets. Moreover, for each function, different data types have been considered (i.e. *int8, int16, int32,* and *float*) to analyze the results with respect to the internal architecture of the considered processor. Each input data set is stored in a header file to be included in the function at compile time.

3.2 Profiling on the Host Architecture

After the inputs generation phase, a procedure to count the number of executed C statements is needed. This value is obtained by performing a profiling of the benchmark functions by means of the *gcov* [5] profiler for each generated input. To obtain the total number of executed C statements for each function, a sum of the single profiling numbers has been performed. It is worth noting that such a profiling is performed one-shot on the host platform since it is independent of the target processor technologies.

3.3 Profiling on the Target Processor

The last data needed to calculate the CC4CS metric is the number of clock cycles needed by the target processor technology to execute each function in the benchmark.

Depending on the processor technology there is the need for an *Instruction Set Simulator* (ISS) or an *HDL Simulator*, for SPP (Figure 1).

4 CC4CS IN THE SW DOMAIN

Once developed the framework, CC4CS has been evaluated first in the SW domain by considering two different processor technologies: an ASP (*Intel 8051*, an 8-bit CISC micro-controller) and a GPP (*LEON3*, a 32-bit RISC core). The first one allows to analyze the metric in the context of limited HW resources (i.e. limited registers, limited internal memory to store code and data, and no cache), while the second one allows to consider a more performing architecture that relies on external memory and cache.

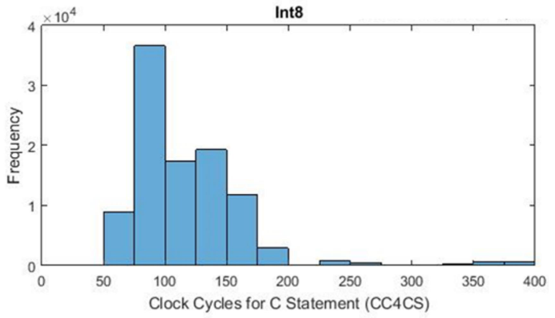

Figure 2: CC4CS (8051): frequency distribution for int8.

4.1 Evaluation of CC4CS for 8051

The first considered processor technology is an ASP: the original Intel 8051 microcontroller built around an 8-bit CPU core with Harvard architecture. The *University of California* has developed a project [6] which provides several tools useful for simulating C code execution on 8051. In particular, the *Dalton ISS* provides the number of clock cycles required by the 8051 to execute a program. So, it has been integrated into the CC4CS framework customized for 8051. The benchmark has been compiled, with SDCC (*Small Device C Compiler*) [7] by using default optimizations. At the end, the ISS has been used to simulate the program execution. Then, the framework provides the metric and some statistics. The results for 1000 executions of the benchmark functions are summarized in Table 1.

Table 1: CC4CS (8051)

Data Type	Min	Max	AM	SD	GM	95%
Int8	59	375	117,51	44,92	110,72	176
Int16	82	493	162,01	64,88	151,09	297
Int32	106	473	223,01	87,57	207,09	402
float	4	1322	526,56	271,68	457,88	1198

*AM: Arithmetic Mean, SD: Standard Deviation, GM: Geometric Mean, 95%: 95th Percentile

For each function, different data types have been considered (*int8, int16, int32*, and *float*). In fact, performances change with respect to the dimension of data since original 8051 is based on an 8-bit CPU core and an 8-bit ALU. Furthermore, with *float* data type, the values of CC4CS(8051) is considerably higher with respect to the other values due to the lack of a FPU (*Floating Point Unit*). For example, by considering int8 data type, CC4CS(8051) belongs to a *Min-Max* interval equals to 59-375, with an *Arithmetic Mean* near to 118 (with *Standard Deviation* near to 45), a *Geometric Mean* near to 111 and the 95th Percentile near to 176. It is worth noting the relevant difference between this last value and the *Max* one. As an example of further possible statistical analysis, Figure 2 shows the frequency distribution graphs of CC4CS(8051) for int8 for the whole benchmark. Such figure clarifies the reason behind the difference between *Max* and *95th Percentile* values. In [11] it is possible to find more details about performed analyses.

4.2 Evaluation of CC4CS for LEON3

The second processor technology is a GPP: the LEON3 microprocessor. LEON3 is a 32-bit synthesizable soft-processor that is compatible with SPARC V8 architecture: it has a seven-stage pipeline and Harvard architecture, and uses separate instruction and data caches. It represents a soft-processor for aerospace applications. *Cobham Gaisler* offers *TSIM System Emulator* as an accurate emulator of LEON3 processors. A free evaluation version of TSIM/LEON3 is available on Cobham website [8], but it does not support code coverage, configuration of caches, memories and so on. Anyway, it has been chosen as the reference ISS for first analysis since it provides the information needed to evaluate CC4CS. By default, TSIM/LEON3 emulates the FPU. Benchmark functions have been compiled, with the *Bare C Cross-Compiler* (BCC) for LEON3 with standard optimization options. Then, the framework has been used to evaluate the metric and some statistics. The obtained results for 1000 executions of the 10 benchmark functions are summarized in Table 2.

Table 2: CC4CS (LEON3)

Data Type	Min	Max	AM	SD	GM	95%
Int8	11	2197	193	304	90	721
Int16	12	2194	291,96	401,52	149,11	1322
Int32	23	2194	437,12	512,07	258,81	2053
float	28	2200	481,70	516,99	305,98	2058

*AM: Arithmetic Mean, SD: Standard Deviation, GM: Geometric Mean, 95%: 95th Percentile

For each function, different data types have been considered (*int8, int16, int32*, and *float*). In fact, performances, especially the average ones, change with respect to the dimension of data. However, in this case, the differences with *float* data type are not as relevant as in the 8051 case since LEON3 exploits a dedicated FPU. For example, by considering *int8* data type, CC4CS(LEON3) belongs to the *Min-Max* interval 11-2197, with an *Arithmetic Mean* equals to 193 (with *Standard Deviation* equals to 304), a *Geometric Mean* of 90 and the 95th Percentile equals to 721. It is still worth noting the relevant difference between this last value and the *Max* while the same difference is not so relevant when considering *int32* and *float* data types.

4.3 Exploitation of CC4CS in SW Domain

Since the main goal of this work is to evaluate usefulness and meaningfulness of CC4CS, this section presents, as a sort of validation, a first attempt to use CC4CS for very early performance analysis in the SW domain. In particular, the goal is to evaluate the errors to be considered when using CC4CS for execution time estimation at very early stages. For this, a set of 5 functions out of the benchmark has been used as testbench: *Selection Sort, Insertion Sort, GCD, Binary Search, Bellman Ford* (the source code is available on [11]). For each function it has been performed a profiling with respect to several inputs and measured the real execution time for 8051@12MHz. Then, such a time has been compared with the estimation made by multiplying the profiling results (i.e. the number of executed C statements) for CC4CS(8051) from Table 1 for the processor frequency. Considering all the testbench, Table 3 shows the average estimation errors obtained by using AM and GM as estimations.

Table 3: Estimation errors for 8051@12 MHz: AM and GM

Data Types	RMSE AM	PRMSE AM	RMSE GM	PRMSE GM	Min 95%	Min Max
Int8	2.90 ms	42.2%	2.50 ms	37.4%	5.74%	0.00%
Int16	1.35 ms	22.9%	1.43 ms	21.9%	4.40%	0.00%
Int32	1.84 ms	21.0%	1.99 ms	24.3%	7.20%	0.00%
float	1.05 ms	76.6%	0.92 ms	60.2%	0.00%	0.00%
AVG	1.79 ms	40.68%	1.71 ms	35.95%	4.34%	0.00%

*RMSE: Root Mean squared Error, PRMSE: Root Mean Squared Percentage Error

Considering all the testbench, Table 3 shows the percentage of estimations that are outside of the *Min-95th Percentile* and *Min-Max* intervals. It is worth noting as all the actual results are contained in *Min-Max*. Despite to fact that such intervals could be quite large (e.g. for *float*), this could mean that CC4CS is quite robust with respect to different common C statements. Moreover, by considering *Min-95%* it is possible to reduce sensitively the range (at least for *int8* and *int16*) while keeping limited errors.

5 CC4CS IN THE HW DOMAIN

To highlight another important feature of CC4CS, i.e. to be unifying with respect to HW and SW domains, this section provides a very preliminary evaluation of CC4CS for functions implemented by means of SPP (i.e. HW) exploiting FPGA technologies. For this, to avoid the need of synthesize all the previously adopted benchmark functions (it will be done for future analyses), it has been exploited the already synthesized (with *standard-optimization* options) benchmark used in [9]. The selected C functions originate from different application domains, which are control-flow as well as data-flow dominated. An important aspect of such benchmarks is that golden inputs and related output vectors are already available for each program. So, it has been possible to execute each function to perform profiling. Then, by exploiting the already available number of clock cycles needed to execute the HW function on a Virtex7, evaluated by means of RTL (*Register-Transfer Level*) simulations, it has been straightforward to evaluate CC4CS(Virtex7). Table 4 shows the corresponding CC4CS for each tool considered in [9].

Table 4: CC4CS (Virtex7)

Tool	Min	AM	GM	Max
Commercial	0.117	1.137	0.758	4.006
Bambu	0.015	1.179	0.468	7.357
DWARV	0.018	1.253	0.650	4.485
LegUp	0.001	1.339	0.583	7.404

*AM: Arithmetic Mean, GM: Geometric Mean

So, at a very first glance, it is possible to state that CC4CS(Virtex7), with standard optimizations, belongs to a *Min-Max* interval equals to 1-8 (rounding up to the nearest integer).

5.1 Exploitation of CC4CS in HW/SW Domain

The availability of CC4CS for both HW and SW processor technologies is very important to exploit such a metric in HW/SW Co-Design methodologies for both early comparison and selection, and for ESL HW/SW timing co-simulations. In the first case, by having available CC4CS for different processors technologies, with the same host-based profiling it is possible to estimate the execution time of a function of interest for the whole processor technologies set so making a very fast preliminary comparison and selection. As an example, given a target function and a related golden input, by means of host-based profiling is possible to count the number of executed C statements (e.g. 100). Then, as shown in Figure 3, it is straightforward to compare the whole processor technologies set by multiplying 100 for the related CC4CS (in this case by using the *Min-95%* interval and GM). Depending on the required execution time it is then possible to select a specific processor technology or, at least, to reduce the set for further analyses. In the second case, CC4CS is useful since several ESL HW/SW timing co-simulations approaches (e.g. [10][12]) rely on the availability of an estimated execution time for each statement composing the ESL specification.

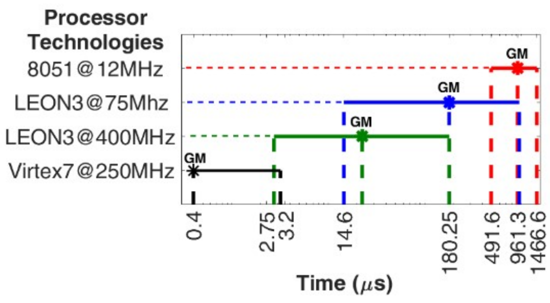

Figure 3: CC4CS-based HW/SW comparison.

5 CONCLUSION AND FUTURE WORKS

This work has presented an off-the-shelf unifying statement-level performance metric and a related evaluation framework. The metric, called CC4CS, has been evaluated both in SW and HW domains analyzing possible exploitation opportunities. Main goal has been to evaluate its usefulness and meaningfulness. For sure some improvements are needed, especially in the C statement and benchmark definition, and further statistical analyses must be performed. For example, Figure 2 shows a frequency distributions graph. Is it possible to individuate a known one (e.g. *Poisson-like*) that fits with it? Probably, analyses related to the specific processor technology features (e.g. registers and memory size, cache and pipeline interferences, etc.) can be considered as well. Moreover, this first approach has voluntary avoided any detailed analysis of the statements composing the given C functions. This kind of approach will be considered as an opportunity to obtain more accuracy but at more cost. Anyway, preliminary results are interesting enough to justify further efforts on the topic.

ACKNOWLEDGMENTS

This work has been partially supported by the ECSEL RIA 2016 MegaM@Rt2 and AQUAS projects.

REFERENCES

[1] D.J. Lilja, Measuring Computer Performance, A Practitioner's Guide, Cambridge University Press, New York, USA, 2000.
[2] Barry Boehm's. COCOMO, Software Engineering Economics, 1981.
[3] SystemC, http://accellera.org/downloads/standards/systemc
[4] M. Siegesmund. Embedded C Programming, Newnes, 2014
[5] GCov Profiler, https://gcc.gnu.org/onlinedocs/gcc/Gcov.html.
[6] Dalton Project, http://www.ann.ece.ufl.edu/i8051/.
[7] SDCC, http://sdcc.sourceforge.net/doc/sdccman.pdf
[8] TSIM/LEON3, http://gaisler.com/doc/tsim-2.0.23.pdf
[9] R. Nane et al., "A Survey and Evaluation of FPGA High-Level Synthesis Tools," in IEEE Trans. on CAD of Integrated Circuits and Systems, Oct. 2016.
[10] L. Pomante, P. Serri. "SystemC-based HW/SW Co-Design of Heterogeneous Multiprocessor Dedicated Systems", Int. Journal of Information Systems, 2014.
[11] CC4CS benchmark, https://github.com/vnzstc/cc4cs
[12] D. Di Pompeo, E. Incerto, V. Muttillo, L. Pomante, and G. Valente. An Efficient Performance-Driven Approach for HW/SW Co-Design. In Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering (ICPE '17). pages 323-326, ACM, 2017.