USING STATISTICAL ASSERTIONS TO GUIDE SELF-ADAPTIVE SYSTEMS

Tim Todman, Wayne Luk

Department of Computing
Imperial College London
180 Queen's Gate, London SW7 2AZ
email: {timothy.todman, w.luk}@imperial.ac.uk

Stephan Stilkerich

Software Engineering
EADS Innovation Works
Willy-Messerschmitt Str. 1, 85521 Ottobrunn
email: stephan.stilkerich@eads.net

ABSTRACT

Self-adaptive systems need to monitor themselves, to check their internal behaviour and design assumptions about run-time inputs and conditions. This kind of monitoring for self-adaptive systems can include collecting statistics about systems themselves which can be computationally-intensive (for detailed statistics) and hence time-consuming, with possible negative impact on self-adaptive response time. To mitigate this limitation, we extend the technique of in-circuit run-time assertions to cover statistical assertions in hardware. The presented designs implement several useful statistical operators that can be exploited by self-adaptive systems. To illustrate the practicability and industrial relevance of our proposed approach, we evaluate our designs, chosen from a class of possible application scenarios, for their resource usage and the tradeoffs between hardware and software implementations.

I. INTRODUCTION

Self-adaptive systems can configure themselves to flexibly deal with changing environments after they are deployed. The configuration itself is systematically guided by means of system self-monitoring to aid decisions about changing modes, or to check design assumptions about runtime data and conditions or their internal operation. Such monitoring could check elementary Boolean conditions or, more generally, could process collected run-time system data, feeding a process of deciding whether or how the system can be adapted. The response time to adaptation is a fundamental feature characterizing self-adaptive systems. For the class of applications from the avionics domain we investigate, a fast response time to adaptation is crucial and motivates our advocated approach, presented in the rest of the paper. Gathered system data can be used for many purposes: for example, design assumptions about input range, used to optimize operator bit-widths, can be checked by assertions about the standard deviation of the input.

In this paper, we propose *in-circuit, statistical assertions*, compiled into the hardware part of a software-hardware design as a dedicated self-monitoring facility for self-adaptive systems, with a fast response time to adaptation. Compared to the proposed in-circuit assertions that can compute in parallel with the rest of the design, purely software-implemented assertions need to wait until the hardware has finished computing its results before they can process their own tasks. Moreover, efficient hardware designs are often deeply-pipelined, operating on large batches of data, further prolonging the time until software assertions can start processing. Additionally, by preprocessing potentially large amounts of data, in-circuit data gathering can improve use of limited bandwidth between hardware and software of the self-adaptive system triggering and controlling system adaptation. In summary: in-circuit assertions are the necessary precondition to realize fast response times to adaptation not realizable by pure software assertions.

Figure I provides a structural overview of our approach. A hardware datapath is instrumented by in-circuit statistical operators which compute relevant statistics about the design. These are then sent back to a software engine running a self-adaptive system. The software builds up the self-adaptive representation which is used to control reconfiguration of the system. It should be mentioned that whilst we target a software-hardware system setting, our approach is not limited to this setting at the outset. The software could likewise run on a soft processor within a Field Programmable Gate Array (FPGA) fabric.

This paper makes the following contributions:

- The design and implementation of in-circuit statistical assertions, which can be used by self-adaptive systems to monitor themselves and control system adaptation;
- A case study on avionics systems, showing the potential of in-circuit statistical assertions;
- Evaluation of tradeoffs between assertion implementations in software and in hardware, showing the advantages of our proposed in-circuit assertions.

The rest of the paper is organized as follows: the next section describes related work. Section III shows our designs for assertions and implementations for Maxeler systems;

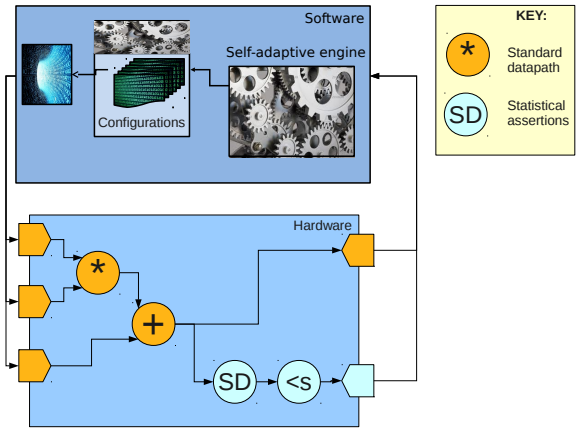

Fig. 1. Our approach: hardware datapath augmented with in-circuit statistical assertions feeding an engine running a self-adaptive algorithm in software.

section IV is a brief case study for avionics. Section V evaluates our implementation; section VI concludes and suggests future work.

II. BACKGROUND

Runtime verification: several researchers have used temporal logic for runtime verification; for example, Reinbacher et al. [1] implement hardware temporal logic monitors for a software system running on a soft processor on the same device. Calinescu et al. [2] propose that self-adaptive software needs quantitative runtime verification; our statistical in-circuit assertions could complement such approaches.

Assertion-based verification allows the use of Boolean and temporal assertions for debugging designs in simulation [3], extended to in-circuit assertions by Curreri [4]. We extend in-circuit assertions with statistical operators; in our approach, failed assertions do not necessarily indicate errors, but may be the trigger for a self-adaptive system to adapt or reconfigure itself.

Statistical assertions have been proposed by Dinh et al. [5], as a debug-time method to reason about large parallel programs – users can reason with derived metrics, rather than raw program output. The assertions are implemented efficiently using a map-reduce style computation. We use statistical assertions for run-time monitoring of reconfigurable hardware-accelerated systems.

III. IN-CIRCUIT STATISTICAL ASSERTIONS

This section shows our designs for in-circuit statistical assertions. Our assertion language comprises C-language style Boolean operators, augmented by statistical primitives. We choose the C language as it is familiar to many designers. The set of statistical primitives is as follows:

- $mean(e1)$, the mean value of expression $e1$;
- $stdev(e1)$, the standard deviation of expression $e1$;
- $variance(e1)$, the variance of expression $e1$.

We choose these as a useful set for expressing statistical conditions; future work could add further statistical operators such as covariance, skewness and kurtosis, or limit the number of cycles over which the statistics are calculated, potentially reducing hardware resources.

The following shows the grammar of our statistical assertions language in extended Backus-Naur form:

```
e = a
  | e bop e
  | uop e
  | mean(e)
  | stdev(e)
  | variance(e)
bop = == | != | < | > | ...
uop = + | - | ! | ~
```

where *bop* represents any C binary operator, *uop* any C unary operator and a any atomic expression (literals, variables, constants). This language allows the user to combine both Boolean and statistical operators.

Online algorithms for calculation of statistical metrics such as mean, variance and standard deviation are known [6] [7], which involve a single pass over the input data, using an accumulator and the current input element. Whilst this may seem suitable for streaming implementations, they contain feedback owing to the accumulator. Chan et al. developed a pairwise algorithm for variance [8] which can be parallelized; for N input elements, naively implementing this algorithm on streaming systems requires $O(NlogN)$ hardware. Chan et al's algorithm denotes the sum and mean of data points x_i as T_{ij} and M_{ij} respectively:

$$T_{ij} = \sum_{k=i}^{j} x_k \qquad M_{ij} = \frac{1}{j-i+1} T_{ij}$$

and the sum of squares S_{ij}:

$$S_{ij} = \sum_{k=i}^{j} (x_k - M_{ij})^2$$

calculated by their pairwise algorithm:

$$S_{1,2m} = S_{1,m} + S_{m+1,2m} + \frac{1}{2m}(T_{1,m} - T_{m+1,2m})^2$$

We propose two designs suitable for streaming systems: a systolic design adapted from Chan's parallel algorithm, and a C-slowed variant of the online algorithm. Figure 2 shows the datapath for the systolic design, combining stream offsets with Chan's pairwise operators for calculating variance or mean; for clarity, we omit the calculation of $T_{i,j}$, which

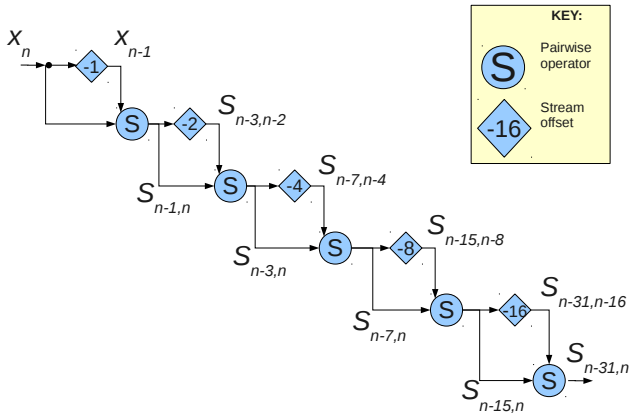

Fig. 2. Systolic partial calculation of variance using pairwise operators.

has the same pattern. Note that the leftmost operator can be optimized, because $S_{i,i} = 0$ (the variance of a single point is zero). The systolic design uses the observation that in a streaming system, iterating through the input data in order, sums of neighbouring elements can be accessed by stream offsets, so using Chan et al's notation:

$$\begin{aligned} T_{1,2^k} &= T_{1,2^{k-1}} + T_{2^{k-1}+1,2^k} \\ &= T_{1,2^{k-1}} + offset(T_{1,2^{k-1}}, -2^k) \end{aligned}$$

where $offset(e, n)$ means the value of expression e sampled n cycles in the past; $S_{1,2^k}$ is calculated in the same way. Unlike the naive implementation of Chan et al's algorithm, which requires $O(NlogN)$ hardware, this requires $O(logN)$ statistical operators plus $O(N)$ delay elements used to implement the $offset$ operation.

Note that this only calculates part of the variance, specifically the local variance around each sample; however, it greatly reduces the amount of data sent back to software. The design consists of repeating units of the pairwise operator and stream offsets to delay the input. Each repeating unit reduces both the output data and the remaining calculations to be done in software by half, so K units reduces it 2^K-fold.

Implementation targeting Maxeler streaming designs: we choose Maxeler streaming systems to implement our designs, though the approach is not Maxeler-specific, and can be ported to other design descriptions such as Verilog and VHDL. We focus on a systematic approach to translating assertions into Maxeler designs; future work could implement a compiler from Maxeler designs extended with statistical assertions into the base language.

The Maxeler system generates *streaming* designs, where inputs and outputs are large arrays used as streams. Each output element is calculated from corresponding elements in one or more input streams; offsets allow reading from neighbourhood stream elements. The user programatically builds a datapath using a domain-specific language based on Java. The control path may be counters or state machines generated from another domain-specific language.

Maxeler tools compile designs into hardware description languages and control FPGA vendor tools to build a reconfigurable device bitstream implementing a design. Software can interact with the generated hardware using a Maxeler application programming interface to configure the FPGA device with the bitstream and run on user data stored in C arrays. The Maxeler tools automatically pipeline the datapath, resulting in deeply-pipelined operators at a high clock rate. This works well for feed-forward designs, but feedback requires some manual intervention and reordering or duplicating of input data.

IV. CASE STUDY: AVIONICS SYSTEMS

Avionics systems are electronic systems used for control or information in the aviation or aerospace industries [9].

Self-adaptive systems with a fast response to adaptation (where fast means quicker than $500ms$), are promising architectures for dedicated application scenarios in the avionics and space-flight industry. Systems that profit from architectures with fast response time to adaptation are:

- autonomous flying systems,
- special satellites,
- deep-space mission systems,
- exploratory space mission systems.

All these systems operate in environments that can not fully be described right from the beginning and hence the systems cannot be statically designed to cover and handle all environmental settings. Furthermore, these systems have strong constraints on power consumption, weight and packaging volume. Additionally, these systems may never be reachable after deployment.

We choose a $500ms$ limit as this duration fits perfectly into most processing and control-loops of the systems and application scenarios mentioned in the paper. Hence, if we realize our self-adaptation and self-expression with the configuration within this limit, it would seamlessly fit into our systems, applications and the already available developed systems.

We analyze the processing structure of these systems for the functionality of guidance, navigation and orientation, revealing that the processing is composed of different blocks/kernels with inputs and outputs. Determining the adequate bit-widths and hence precision for the inputs and outputs is difficult and is today based on worst-case assumptions involving unnecessary resources. An alternative is to start with an initial, more optimistic design assumption about the input/output range, used to optimize operator

bit-widths. Such assumptions can be checked by assertions about the standard deviation of the input and adapted by another kernel-version accordingly if required. Obviously, fast response time to adaptation is to avoid compromising system functionality, and to simultaneously optimize the system at run-time with respect to energy efficiency and environmental adaptability.

V. EVALUATION

We evaluate our implementations of on-chip statistical assertions showing the tradeoff between hardware and software implementations. We compare: 1) scalability: operator size versus hardware size; 2) software statistics versus hardware-assisted: speed, bandwidth.

Experimental setup: we implement our designs using Maxeler compiler version 2012.1 and Xilinx ISE 13.1. Designs target a MAX3 system, containing a Xilinx xc6vsx475t FPGA, with a speed goal of 200MHz. We implement a single variance assertion, with 32-bit input data in IEEE Single-Precision (SP) floating-point format, one data element per cycle.

Figure 3 shows the effect of unroll factor on the area resources for the systolic variance operator; the area is measured in Look-Up Tables (LUTs), Flip-Flops (FFs) and Digital Signal Processing (DSP) blocks. The area cost is linearly proportional to the unroll factor (for LUTs and FFs), but the output data reduction factor is exponential: increasing the unroll factor by one halves the output volume. For unroll factor 15, the data reduces by 2^{15} and the variance takes about 5% of flip-flops, 8% of other resources. For LUTs and FFs there is also a small fixed cost which is due to the Maxeler runtime system used to communicate with the host. The cost in block RAMs (BRAMs) is exponential in the unroll factor, as they are used to store delayed stream elements used to calculate the $offset$ expressions; however, the cost is still modest even for large unroll factors.

The C-slowed design uses a small fixed area per assertion (about 3.5% of LUTs, 2% of FFs for 32-bit SP variance). For 32-bit SP data, the pipeline is 85 stages long, padded to 128 stages. The data are reduced to 128 partial variances, which can be further reduced to a single variance by Chan et al's method. The design runs at 300MHz.

Case study: avionics: we assume a hard $0.5s$ limit for hardware run time. Figure 4 shows estimated run times versus number of statistical assertions for both software and hardware implementations. We assume the design is limited by the bandwidth between software and hardware (MAX3 has 2GB/s maximum speed); stream length is 2^{26}, each output is 4 bytes wide, so the run time with no assertions is 0.15 seconds. Software calculations are limited to two assertions within the time limit, because all 2^{26} values must be streamed across the bus for each exception. In contrast, the C-slowed design summarizes 2^{26} data to 128 values,

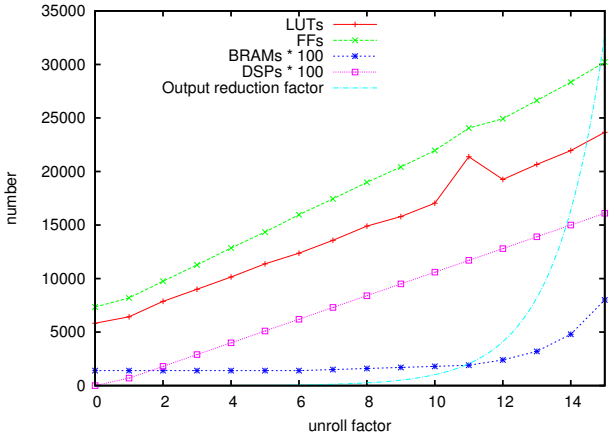

Fig. 3. Area usage and output reduction versus unroll factor for systolic variance operator.

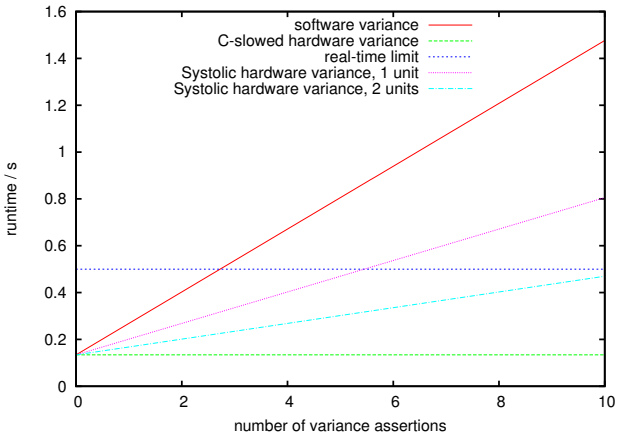

Fig. 4. Estimated time taken by software and C-slowed hardware variance assertions versus number of assertions.

meaning the time cost of each exception is much lower. Systolic hardware designs allow the number of assertions to be traded for hardware area. Note we do not include time to calculate the variance on the host.

VI. CONCLUSION

To allow efficient monitoring for self-adaptive systems, we design and implement in-circuit statistical assertions, allowing designs to use several frequently-occurring statistical operators to express desired runtime properties of design inputs, outputs and internal signals. Results show that response time can be greatly reduced at a modest cost in hardware area per exception.

Current and future work includes enlarging the set of statistical primitives to allow more general assertions on the state of the design. We would also like to explore the interaction of the statistical operators with run-time

reconfiguration. Statistical conditions can be used to decide when to reconfigure. More generally, the statistics operators themselves can be reconfigured, allowing the system to alter the balance of configurable hardware between assertions and computation depending on runtime conditions.

Acknowledgements: The research leading to these results has received funding from the European Union Seventh Framework Programme under grant agreement numbers 257906 and 287804.

VII. REFERENCES

[1] T. Reinbacher, M. Függer, and J. Brauer, "Real-time runtime verification on chip," in *Runtime Verification*, ser. Lecture Notes in Computer Science, S. Qadeer and S. Tasiran, Eds. Springer Berlin Heidelberg, 2013, vol. 7687, pp. 110–125.

[2] R. Calinescu, C. Ghezzi, M. Kwiatkowska, and R. Mirandola, "Self-adaptive software needs quantitative verification at runtime," *Communications of the ACM*, vol. 55, no. 9, pp. 69–77, 2012.

[3] S. Vasudevan, "What is assertion-based verification?" *SIGDA E-News*, vol. 42, no. 12, December 2012.

[4] J. Curreri, G. Stitt, and A. D. George, "High-level synthesis of in-circuit assertions for verification, debugging, and timing analysis," *International Journal of Reconfigurable Computing*, vol. 2011, 2011.

[5] M. N. Dinh, D. Abramson, J. Chao, D. Kurniawan, A. Gontarek, B. Moench, and L. DeRose, "Debugging scientific applications with statistical assertions," *Procedia Computer Science*, vol. 9, no. 0, pp. 1940 – 1949, 2012, proceedings of the International Conference on Computational Science, (ICCS) 2012.

[6] D. E. Knuth, *The Art of Computer Programming, volume 2: Seminumerical Algorithms*, 3rd ed. Addison-Wesley, 1998.

[7] B. P. Welford, "Note on a method for calculating corrected sums of squares and products," *Technometrics*, vol. 4, no. 3, pp. pp. 419–420, 1962.

[8] T. F. Chan, G. H. Golub, and R. J. LeVeque, "Updating formulae and a pairwise algorithm for computing sample variances," Department of Computer Science, School of Humanities and Sciences, Stanford University, Tech. Rep. STAN-CS-79-773, November 1979.

[9] R. P. G. Collinson, *Introduction to avionics systems*, 2nd ed. Kluwer, 2003.