INVITED Time-Predictable Computing by Design: Looking Back, Looking Forward

Tulika Mitra
School of Computing, National University of Singapore
tulika@comp.nus.edu.sg

ABSTRACT

We present two contrasting approaches to achieve time predictability in the embedded compute engine, the basic building block of any Internet of Things (IoT) or Cyber-Physical (CPS) system. The traditional approach offers predictability on top of unpredictable processors with numerous optimizations for enhanced performance and programmability at the cost of huge variability in timing. Approaches such as Worst-Case Execution Time (WCET) analysis of software have been struggling to model the complex timing behavior of the underlying processor to provide guarantees. On the other hand, the inevitable slowdown of Moore's Law and the end of Dennard scaling have curtailed the performance and energy scaling of the processors. This stagnation in conjunction with the importance of cognitive computing have motivated widespread adoption of non-von Neumann accelerators and architectures. We argue that these emerging architectures are inherently time-predictable as they depend on software to orchestrate the computation and data movement and are an excellent match for the real-time processing needs.

ACM Reference Format:
Tulika Mitra. 2018. INVITED Time-Predictable Computing by Design: Looking Back, Looking Forward. In *Proceedings of ACM Design Automation Conference (Mitra '19).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

The Internet of Things (IoT) and Cyber-Physical Systems (CPS) revolution is rapidly ushering in myriad of new applications that need to interact with the physical environment in real time but are as prevalent as smartphones. Examples of such time-critical systems range from personal smart-drones, medical devices, robots to industrial IoT, in addition to the traditional real-time systems such as avionics, automotive. All these emerging applications have irrefutable safety concerns associated with the failure to meet timing deadlines. Thus predictable response time of a computing system has become the first-class design target for diverse application scenarios rather than being relegated to only a selected few [11].

Real-time systems require strict timing guarantees for the computation performed on the underlying computer architecture. In particular, given a software program implementing the computation and an architecture on which the program will execute, one needs to find out the worst-case execution time (WCET) of the program on the architecture. As long as the WCET is less than the deadline, it is safe to deploy the software in the real-time system. The WCET value is influenced by two primary factors: (a) the input to the program that determines the path that is taken through the program, and (b) the characteristics of the underlying architecture. Thus, we can estimate the WCET value by first determining the execution time of each instruction in the program through accurate timing models of the architecture. Next, we determine the worst-case path through the program, given the timing properties of the individual instructions. Identifying the wost-case path, while challenging, can be achieved through a variety of techniques including Integer Linear Programming (ILP) formulation [9]. This is a mature technology and we will assume the existence of robust and sophisticated worst-case path analysis approach in the rest of the article. Instead, our focus here is on the timing behavior of the architecture, which is very complex and difficult to model.

2 LOOKING BACK

In the beginning, the real-time embedded systems were built with very simple micro-controllers, where each instruction executes for a fixed and known clock cycles. Thus, it was trivial to compute the WCET of each individual instruction and the main challenge was in finding the worst-case path through the program. The period between 1985 to early 2000 witnessed a golden revolution in the micro-architecture of the processors that resulted in roughly 52% performance improvement per year [7]. This astonishing performance

Mitra '19, June 02–06, 2019, Las Vegas, NV

ACM ISBN 978-1-4503-9999-9/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

trajectory was fueled by two factors. First, Moore's Law ensured that the number of transistors incorporated in a chip approximately doubles every 18-24 months, resulting in an exponential increase in transistor density. As the processing speed is inversely proportional to the distance between the transistors on an integrated circuit, the implication is that the clock frequency of the processors also doubled every 24 months. Second, the simple Reduced Instruction Set Computer (RISC) instruction-set architecture (ISA) emerged in this era. The simplified ISA allowed the micro-architectural optimizations to flourish that exploited the instruction-level parallelism (ILP) in the program through various hardware mechanisms such as pipeline, memory hierarchy including on-chip caches, branch prediction, out-of-order execution, speculation to name a few. As the ISA remained unchanged across generations and the performance gain was achieved by harvesting parallelism by the micro-architecture under the hood, the optimizations were not exposed to the programmers. This separation of concern between functionality (that programmers need to focus on) and performance (that is automatically boosted across processor generations without programmer involvement) was the biggest force behind the success of the micro-architectural changes. The real-time systems community could not remain immune to this dramatic performance advances and had to embrace these new generation processors with simple ISA and complex micro-architecture. Ironically, while the transparent performance growth was the biggest asset in the general-purpose computing domain with substantial legacy code, it created considerable complications in the context of real-time systems and in particular WCET estimations.

Recall that the WCET estimation of a program requires us to compute the WCET of each individual instruction. Prior to the introduction of the micro-architectural optimizations, there was no variation in the execution time an instruction. The pipeline, caches, and the additional performance-enhancing features in a processor are focused on improving the average-case performance; but in the process they introduce significant variability in the execution time of an instruction. Consider a memory load instruction as an example. In the absence of caches, it has a pre-determined latency, which is equal to the high latency of accessing the off-chip main memory (we ignore the access latency variations in modern DRAM chips used in main memory design for the moment). When the processor includes on-chip caches, the memory access can be a hit or a miss depending on whether the data is already present in the cache or not. The hit has very low latency, while the miss has high latency due to off-chip main memory access to bring the data on-chip. More importantly, the same memory load instruction may have different latency at different instances of execution (e.g., within a loop) depending on the context. In order to estimate the average-case execution time of the program, one can simply profile the execution with representative inputs and find the cache hit rate. Unfortunately, one cannot rely on an approach based on representative inputs for the worst-case analysis because it is unsafe (i.e, may miss corner cases that result in worse execution time). The same problem arises for other micro-architectural features including pipeline, branch prediction, out-of-order execution etc. and their interaction.

Abstract Interpretation (AI) is currently the preferred approach that employs static analysis to estimate the dynamic timing properties of the instructions in a program in a context-sensitive fashion. Let us consider the cache behavior prediction via AI [1]. The concrete cache state at a program point depends on the context through which it is reached, i.e., the paths taken before reaching the program point. Thus a program point can have an exponential number of concrete cache states for all the different contexts, which makes any analysis based on concrete cache states infeasible. The AI-based static analysis maintains an abstract cache state at each program point that approximates all the concrete cache states into a single abstract state. To be more specific, the WCET estimation requires three different abstract cache states: the *may*, *must*, and *persistent* states that differ in the direction of approximation. The *may* state captures the data that are present in at least one context, while the *must* state only includes data that are present in all the contexts. The *persistent* state is required for memory accesses that are cold miss in the first instance and hit in all subsequent instances. It is easy to observe that a data that is present in the *must* state is guaranteed to be *always hit*, a data that is missing from the *may* state is guaranteed to be *always miss*, and *persistent* data will have one miss. The remaining data accesses are conservatively assumed to the cache miss. Similar abstract states can be created for pipeline, out-of-order execution etc. for full modeling of the micro-architectural features.

While the WCET analysis has made tremendous progress in the last two decades, there are several inherent shortcomings. First, the analysis assumes perfect knowledge of the underlying micro-architectural optimizations. In reality, the processor vendors do not expose all the details of the micro-architecture because they treat the ISA as the functional interface between hardware and software. As mentioned earlier, the micro-architectural advances are meant to be transparent to the external world and provide the competitive edge to the processor architecture. Recently, there has been call to expose the micro-architectural details, both from timing analysis and security perspectives (e.g., Spectre and Meltdown attacks), as part of an extended ISA. At present, however, one needs to rely on extensive reverse engineering efforts and micro-benchmarking to uncover the parameters and secrets of the underlying architecture, including sometimes simple parameters such as cache block size, number

of sets, replacement policy etc. Second, the WCET analysis becomes increasingly challenging with growing complexity of the processors. For example, the popularity of shared memory multi-core processors in real-time systems demand the WCET analysis to be extended to model shared resources such as the bus, the network-on-chip, and the shared caches. This is far more complicated than uni-processor timing modeling. In summary, the WCET analysis techniques are perpetually attempting to catch up with the hardware progress.

An alternative to the present approach is to design architectures that are inherently predictable in nature but still provides sufficient performance. It is not possible to go back to the days of the micro-controllers as embedded real-time systems today perform intensive computations that require high-performance architectures. In the past, there have been efforts towards both time-predictable architectural components as well as complete architectures. In the memory hierarchy, cache locking [4], cache partitioning [13], and software-controlled scratchpad memory (SPM) [14] are the popular approaches to predictability. Cache locking loads selected content in the cache and then locks it to prevent changes due to cache replacement policy. Cache partitioning isolates the cache content for each task so that a task cannot replace another task's content in the cache. Finally, in SPM, the memory content is divided with frequently accessed content placed in the on-chip SPM and the remaining data in the off-chip memory. In all these cases, the software controls the content in the on-chip memory and hence the latency of each memory access is known depending on where in the memory hierarchy the corresponding data has been placed.

For a complete processor core, Edwards and Lee [5] argues the case for precision timed (PRET) machines in time-critical systems. Their vision is an architecture that provides cycle-accurate timers, a predictable memory hierarchy based on SPM, and an interleaved pipeline that provides predictable hardware-efficient concurrency. The PRET machine needs support from a C-like programming language extended to include user-specified timing constraints and concurrency. A multi-core version of PRET is presented in [2] with temporal semantics at micro-architecture level, in the memory hierarchy, in on-chip communication, and in the instruction-set architecture. Other examples of time-predictable architectures include T-CREST [12], MERASA [15], Kalray [3], and SPECTRUM [16].

However, all these approaches still remain firmly situated in the traditional von Neumann architecture domain.

3 LOOKING FORWARD

At present, the architectural landscape is going through a radical transformation and we argue that the future looks bright from the timing predictability prospects.

For almost thirty years, Moore's Law was aided by Dennard Scaling to keep the processor power within limit. The breakdown of Dennard Scaling has curtailed the rise in clock frequency due to thermal limits and instead the progress has shifted to multi- and many-core architectures to take advantage of the abundance of transistors as Moore'a Law is still alive. However, we are slowly but surely reaching the end of the road in terms of performance gain from general-purpose processors to a meager 3% per year [7]. There are several constraints that have contributed to this fall including memory wall, power wall, and the ILP wall as well as speedup limit of parallel code due to sequential bottleneck in accordance with Amdahl's Law. Together, these technology trends have led to the emergence of heterogeneous computing where general-purpose processor cores coexist on-chip with various accelerators including GPU, reconfigurable computing, digital-signal processors as well as domain-specific accelerators (e.g., Neural Processing Unit, Video encoding/decoding units etc.) [10]. From real-time systems viewpoint, the heterogeneity seems to add another level of complexity and timing unpredictability; indeed, the whole system-level timing analysis becomes somewhat more challenging. Nonetheless, a closer look reveals that the current trend is actually beneficial for predictability as the individual hardware accelerators are no longer tied to the von Neumann architecture and either expose sufficient architectural states at the software level or expose very little but have limited timing variability. The kernels are offloaded to these accelerators and hence accurate timing analysis of the computation on the accelerators becomes the major element in providing timing guarantees.

Let us first consider reconfigurable computing in the form of popular Field Programmable Gate Arrays (FPGA) and Coarse-Grained Reconfigurable Arrays (CGRA) that are becoming ubiquitous in real-time embedded systems. In case of FPGAs, the high-level synthesis (HLS) techniques perform both spatial and temporal mapping of the computation onto the reconfigurable substrate. The entire computation schedule is pre-orchestrated including memory transfers. Moreover, the clock frequency that covers the critical path in a clock cycle is also known. In other words, as the FPGA architecture with all the details of the timing properties is made available fully to the HLS tools, the timing guarantees are obtained naturally as part of the synthesis process. A CGRA [8] can be thought of as an array of very simple processing elements (PE) with a simple interconnect (typically 2D mesh). Different from the bit-level reconfiguration in FPGA, the PEs and the interconnect in the CGRA are reconfigured at word level (hence the term coarse-grained) each cycle. Again, the compiler is responsible to come up with cycle-by-cycle spatial and temporal mapping of the loops onto the CGRA fabric through software pipelining. The mapping determines the *Initiation Interval (II)*, which is the number of cycles between

two consecutive loop iterations. Unlike the FPGA, where the clock frequency is determined by the critical path after mapping, the clock frequency is fixed for the CGRA. Thus the II value dictates the execution time of the loop and there is no timing variation from the underlying architecture. In summary, FPGA and CGRA can be though of as a broader class of software-defined hardware architectures where the timing variability is completely eliminated.

Finally, we move the spotlight to the emerging workloads and the domain-specific accelerators that are primarily responsible to execute them. Traditionally, the real-time guarantees have been mostly required for the control systems part of the embedded CPS and IoT devices. The control systems software generally is not heavy in terms of computation but is full of control divergent branches. The new generation of CPS and IoT systems, specially in domains such as self-driving cars, 5G baseband processing etc., perform substantial computation that is mapped to domain-specific accelerators and should be processed in real-time. Another dominant trend is the inclusion of cognitive computing and in particular, deep learning as part of the workload. The self-driving cars deploy deep neural network (DNN) inference for real-time object detection to avoid obstacles on road. These workloads are executed either on the GPU or dedicated neural-network accelerators with real-time deadlines. At present, GPUs present challenges [6] towards its integration into real-time systems (partially hindered again by the lack of propriety micro-architectural details including thread scheduling). But the neural processing units, such as Google TPU (Tensor Processing Unit) [7] are inherently far more predictable. The TPU architecture is minimalist in nature and precludes sophisticated micro-architectural features such as caches, branch prediction, out-of-order execution, multiprocessing, speculative pre-fetching, address coalescing, multi-threading, context switching, etc. that improve average-case performance but does nothing to the worst-case performance. Essentially, the TPU is very large two-dimensional multiply unit at its heart with decoupled access-execute model to bring in data from the off-chip memory while concurrently executing time-predictable matrix multiplication. The simple and repeatable execution model of the TPU makes it easy to provide response time guarantees for DNN inferences. Interestingly, the TPU brings back into vogue the Complex Instruction-Set Computer (CISC) ISA with the domain-specific complex instructions and avoids the timing predictability pitfalls of RISC machines comprising of complex internal hardware-level optimizations and reordering of the instructions.

4 CONCLUSION

In summary, the real-time systems community in the past has struggled to develop software with provable timing guarantees in the face of increasingly complex but invisible architectural details. As the architectural landscape evolves towards hardware accelerators in response to the emerging application workloads, the research in providing timing predictability for software needs to move in tandem to fully exploit and embrace these exciting new opportunities. Greater synergy, understanding, and co-operation among the architecture, design automation, and real-time software community are essential to reach the predictability goals.

ACKNOWLEDGMENTS

This work was supported by the National Research Foundation, Prime Ministers Office, Singapore under its Industry-IHL Partnership Grant NRF2015-IIP003.

REFERENCES

[1] Martin Alt, Christian Ferdinand, Florian Martin, and Reinhard Wilhelm. 1996. Cache behavior prediction by abstract interpretation. In *SAS*.
[2] Dai Bui, Edward Lee, Isaac Liu, Hiren Patel, and Jan Reineke. 2011. Temporal isolation on multiprocessing architectures. In *DAC*.
[3] Benoît Dupont de Dinechin, de Massas, et al. 2013. A distributed run-time environment for the kalray mppa®-256 integrated manycore processor. *Procedia Computer Science* 18 (2013).
[4] Huping Ding, Yun Liang, and Tulika Mitra. 2012. WCET-centric partial instruction cache locking. In *DAC*.
[5] Stephen A Edwards and Edward A Lee. 2007. The case for the precision timed (PRET) machine. In *DAC*.
[6] Glenn A Elliott and James H Anderson. 2011. Real-world constraints of GPUs in real-time systems. In *RTCSA*.
[7] Norman P Jouppi, Cliff Young, Nishant Patil, and David Patterson. 2018. A domain-specific architecture for deep neural networks. *Commun. ACM* (2018).
[8] Manupa Karunaratne, Aditi Kulkarni Mohite, Tulika Mitra, and Li-Shiuan Peh. 2017. Hycube: A cgra with reconfigurable single-cycle multi-hop interconnect. In *DAC*.
[9] Yau-Tsun Steven Li and Sharad Malik. 1995. Performance analysis of embedded software using implicit path enumeration. In *ACM SIGPLAN Notices*, Vol. 30. ACM, 88–98.
[10] Tulika Mitra. 2015. Heterogeneous multi-core architectures. *Information and Media Technologies* 10, 3 (2015).
[11] Tulika Mitra, Jürgen Teich, and Lothar Thiele. 2018. Time-critical systems design: A survey. *IEEE Design & Test* 35, 2 (2018).
[12] Martin Schoeberl et al. 2015. T-CREST: Time-predictable multi-core architecture for embedded systems. *Journal of Systems Architecture* (2015).
[13] Vivy Suhendra and Tulika Mitra. 2008. Exploring locking & partitioning for predictable shared caches on multi-cores. In *DAC*.
[14] Vivy Suhendra, Tulika Mitra, Abhik Roychoudhury, and Ting Chen. 2005. WCET centric data allocation to scratchpad memory. In *RTSS*.
[15] Theo Ungerer et al. 2010. Merasa: Multicore execution of hard real-time applications supporting analyzability. *IEEE Micro* 30, 5 (2010).
[16] Vanchinathan V and T Mitra A Kulkarni, LS Peh. 2019. SPECTRUM: A Software Defined Predictable Many-core Architecture for LTE Baseband Processing. In *LCTES*.