Is Time Predictability Quantifiable?

Martin Schoeberl

Abstract—**Computer architects and researchers in the real-time domain start to investigate processors and architectures optimized for real-time systems. Optimized for real-time systems means time predictable, i.e., architectures where it is possible to statically derive a tight bound of the worst-case execution time. To compare different approaches we would like to quantify time predictability. That means we need to *measure* time predictability. In this paper we discuss the different approaches for these *measurements* and conclude that time predictability is practically not quantifiable. We can only compare the worst-case execution time bounds of different architectures.**

I. Introduction

In computer architecture, and in natural science in general, we want to measure and quantify. Computer architecture can advance when we can say that processor A is 2 times faster than processor B. We all like *bigger is better*. Of course this simplified answer is not really possible, as average-case performance depends also on the workload. But with a known workload, such as SPEC benchmarks, such comparisons are possible.

For real-time systems we need to know the worst-case execution time (WCET). For modern processors deriving a safe and tight bound on the WCET becomes practically infeasible. Therefore, researchers in real-time systems and computer architecture started to design processors to simplify the static estimation of the WCET. Those architectures are often called time predictable. The question now arises what time predictable exactly mean. And can one find a measurement methodology so we can state: processor A is more time predictable than processor B? Or better: processor a is twice as predictable as processor B.

II. Predictable Hardware

Digital hardware is in principle perfectly predictable. With the same hardware start state and the same input data two runs of a program will result in the same result and execution time. However, the initial hardware state and variable input data make WCET analysis for modern processor very hard.

As an answer to this issue several groups have started to investigate predictable processors and memory hierarchies.

A. PRET

Edwards and Lee argue: “It is time for a new era of processors whose temporal behavior is as easily controlled as their logical function” [4]. The focus of the precision timed (PRET) machine is primarily on repeatable timing and less on predictable timing. A deadline instruction can be used to enforce repeatable timing of a task. A first simulation of their PRET architecture is presented in [12]. PRET implements a RISC pipeline and performs chip-level multithreading for four threads to eliminate data forwarding and branch prediction [13]. Scratchpad memories are used instead of instruction and data caches. The shared main memory is accessed via a time-division multiple access (TDMA) scheme, called the memory wheel. A recent version of PRET [3] defines time-predictable access to SDRAM by assigning each thread a dedicated bank in the memory chips. The access to the individual banks is pipelined and the access time fixed. As the memory banks are not shared between threads, thread communication has to be performed via the shared scratchpad memory. Although the PRET architecture is not a CMP system, the concepts used in the multi-threaded pipeline can also be applied to a CMP system. Each bank of the SDRAM can be assigned to a set of CPUs. Within this set a TDMA based memory arbitration can be used.

B. MERASA

The FP-7 project MERASA (Multi-Core Execution of Hard Real-Time Applications Supporting Analysability) [24] investigated a bus-based CMP with a multi-threaded version of the TriCore processor. The memory hierarchy issue for real-time systems was attacked by a dynamic instruction scratchpad [14]. The WCET analysis tool used in MERASA was adapted to support the instruction set of the TriCore. A memory arbiter was designed to support measurement-based WCET analysis with the RapiTime tool [15].

C. PREDATOR

Thiele and Wilhelm argue that a new research discipline is needed for time-predictable embedded systems [23]. Berg et al. identify the following design principles for a time-predictable processor: “... recoverability from information loss in the analysis, minimal variation of the instruction timing, non-interference between processor components, deterministic processor behavior, and comprehensive documentation” [2].

The FP-7 project PREDATOR investigated time-predictable hardware architectures and WCET-driven compilation. The authors propose a processor architecture that meets these design principles. The processor is a classic five-stage RISC pipeline with minimal changes to the instruction set. Suggestions for future architectures of memory hierarchies are given in [28]. Falk et al. [5], [6] developed the WCET-driven compiler WCC. WCC is guided by the results of the aiT WCET analysis tool and optimizes the worst-case path.

D. JEOPARD

The FP-7 JEOPARD (Java Environment for Parallel Realtime Development) investigated architectures and tools for real-time Java on CMP systems. Within the hardware architecture work package the Java processor JOP [20] was extended to support time-predictable execution of Java applications on a CMP. The TDMA based memory access arbitration [16] was incorporated into the WCET analysis tool of JOP. Research on time-predictable caching [21] for Java started within JEOPARD.

E. York RTS Group

Whitham argues that the execution time of a basic block has to be independent of the execution history [25]. To reduce the WCET, Whitham proposes to implement the time critical functions in microcode on a reconfigurable function unit (RFU). The main processor implements a RISC ISA as a microprogrammed, sequential processor. With several RFUs, it is possible to explicitly extract instruction level parallelism (ILP) from the original RISC code in a similar way to VLIW architectures. Whitham and Audsley extend the MCGREP architecture with a trace scratchpad [26]. The trace scratchpad caches microcode and is placed after the decode stage. The authors extract ILP at the microcode level and schedule the instructions statically – similar to a VLIW architecture.

Superscalar out-of-order processors can achieve higher performance than in-order designs, but are difficult to handle in WCET analysis. Whitham and Audsley present modifications to out-of-order processors to achieve time-predictable operation [27]. Virtual traces allow static WCET analysis, which is performed before execution. Those virtual traces are formed within the program and constrain the out-of order scheduler built into the CPU to execute deterministically.

III. Predictability Definitions

Thiele and Wilhelm are one the first to define time predictability [23]. The consider the difference between the real WCET and the WCET bound, which is often called the *pessimism* of the analysis, as a measure for predictability. They also include the difference between the lower bound and the best-case execution time (BCET) for the predictability definition.

Kirner and Puschner present several variants of time predictability definitions [11]. The first definition is the interval between the BCET and the WCET: [BCET, WCET], where a smaller interval means better predictable and BCET = WCET considered highest predictability. More informal they also define that time predictability of a *model* is the ability to calculate the execution time of a *concrete* system. Specific they define predictability by the multiplication of *analyzability* with *stability*, including weighting factors. The stability is given by the quotient of BCET/WCET of the *timing model*. For WCET-predictability the stability is ignored and the factor for it set to zero. The analyzability, which is now equal to the WCET-predictability, as the quotient of WCET and WCET bound.

Grund, Reineke, and Wilhelm investigate predictability and provide a template for the definition [7]. The template consists of three aspects: (1) the property to be predicted, (2) the source of uncertainty, and (3) a quality measure. In the following they concentrate on the property time and present a definition of time predictability, which is defined as the a quotient between the minimum and maximum execution time. A system with quotient 1 is perfectly predictable. They argue that each system has an inherent predictability factor and the WCET analysis shall not be part of this factor. However, the real WCET and BCET are usually not known. Therefore, it is not possible to 'measure' time predictability with this definition.

In summery following definitions of time predictability can be found:

- BECT and WCET interval
- WCET/WCET bound quotient
- BCET/WCET quotient (excluding any analysis)
- BCET/WCET bound quotient (including the analysis)

All the above definitions share the same issue: they are very impractical. We do not know the real WCET or BCET, therefore we cannot calculate the predictability factors in praxis. How can we help the real-time computer architect or compiler writer?

IV. What to Quantify?

Time predictability itself is is an interesting, but incomplete property for a concrete system. We are still interested in performance. For real-time systems it is the worst case performance, which counts.

Let us assume a concrete example to look at the different real-time related properties. For a given task X processor A has a WCET of 1000 cycles and processor B 800 cycles. So B is better. Maybe B is better be a factor of 10/8.

However, this is not the end of the story as we do not know the exact WCET values. We can only infer a WCET bound with static WCET analsys. We now introduce two WCET tools Wa and Wb for the two processors. Wa gives a bound of 1100 cycles for task X on processor A and Wb a bound of 1300 cycles on processor B. Now processor A is better.

It looks like Wa is inferior to Wb as it gives a tighter bound, even when the WCET on processor B is lower. Maybe Wa performs the analysis in one day, but Wb gives a result after one minute. Is in this case system B with Wb is more practical than system A with Wa.

Furthermore, we have not yet included the compiler in this WCET performance comparison. Similar to average case measurements, also for the WCET performance the compiler plays an important role.

We argue that a time predictability comparison shall be a comparison of WCET bounds between two systems. And those two systems include three components: the hardware, the compiler, and the static WCET analysis tool.

We can state that any WCET performance measurement is a relative one with different factors that can be considered. (we did not yet consider the size or energy consumption of processor A and B). Furthermore, we do not have a base line as SPEC measurements have with the VAX machine for comparison.

V. T-CREST

Guided by the fact that the WCET bound depends on the hardware architecture, the compiler, and the WCET analysis tool the EC funded project T-CREST (Time-predictable Multi-Core Architecture for Embedded Systems) will research all these aspects to build future time-predictable multicore systems.

Within T-CREST novel solutions for time-predictable multicore and many-core system architectures will be proposed. The resulting time-predictable resources (processor, interconnect, memories, etc.) will be a good target for WCET analysis and the WCET performance will be outstanding compared to current processors. Time-predictable caching and time-predictable chip-multiprocessing (CMP) will provide a solution for the need of increased processing power in the real-time domain.

Besides the hardware, a compiler infrastructure will be developed in the project. WCET-aware optimization methods will be developed along with detailed timing models such that the compiler benefits from the known behavior of the hardware. The WCET analysis tool aiT will be adapted to support the developed hardware and guide the compilation.

The T-CREST hardware will be open-source under the industry friendly, simplified BSD license.

A. The Processor

The basis of a time-predictable system is a time-predictable processor. Within T-CREST we will develop a time-predictable processor, named Patmos [22], as one approach to attack the complexity issue of WCET analysis. Patmos is a statically scheduled, dual-issue RISC processor that is optimized for real-time systems. All instruction delays are visible at the instruction set architecture (ISA). This puts more burden on the compiler, but simplifies the WCET analysis tool. A major challenge for the WCET analysis is the memory hierarchy with multiple levels of caches. We attack this issue by caches that are especially designed for WCET analysis. For instructions we adopt the method cache [19], which operates on whole functions/methods and thus simplifies the modeling for WCET analysis. Furthermore, we propose a split-cache architecture [21] for data, offering dedicated caches for the stack area, constants, static data, heap allocated objects, as well as a compiler and program managed scratchpad memory.

Accesses to the different types of data areas are explicitly encoded with the load and store instructions. We call this typed load and store instructions, which direct the loads and stores to the relevant cache. This feature helps the WCET analysis to distinguish between the different data caches. Furthermore, it can be detected earlier in the pipeline which cache will be accessed.

Patmos also supports predication of all instructions. This feature reduces the number of conditional branches and supports generation of single-path code [17], [18]. The compiler LLVM will be extended with an optimization path that translates normal code into single-path code.

B. The Interconnect

In order to build a chip-multiprocessor system out of Patmos processor cores we need a suitable interconnect – a network-on-chip (NoC). The Patmos multi-processor platform will use non-coherent distributed memory and the address space will be populated by smaller memory blocks within the different processor nodes and a large block of off-chip memory. The NoC will support time-predictable read and write transactions to memories in other processor nodes as well as to the off-chip memory. The NoC is a shared resource and it must support multiple concurrent read and write transactions.

To enable time-predictable usage of a shared resource the resource arbitration has to be time-predictable. In the case of a NoC, statically scheduled TDMA is a time-predictable solution. This static schedule is repeated and the length of the schedule is called the *period*. Like tasks in real-time systems, also the communication is organized in periods. One optimization point of the design is minimizing the period to minimize the latency of delivering data and the size of the schedule tables.

In the field of embedded systems, multi-processor platforms are typically optimized for a given application or application domain. The NoC structure and/or the routing schedules are then optimized and are then application-specific. In the T-CREST project our aim is to develop a general-purpose platform – a platform that can be configured to optimize the performance of the system or a platform which can be used as is without any configuration.

Different types of data is transferred on the NoC, e.g., message passing data between cores, cache fills from main memory, synchronization operations such as compare-and-swap. In most architectures a single NoC serves all those different types of data. However, the requirements of these different data types with respect to e.g., packet size, address ranges, and flow control are different. Therefore, T-CREST will evaluate if several, for the traffic type optimized, NoCs result in a more efficient solution than a single shared NoC.

C. Memory Hierarchy

The only memory layer that is under direct control of the compiler is the register file. Other levels of the memory hierarchy are usually not visible – they are not part of the ISA abstraction. The placing of data in the different layers is automatically performed. While caches are managed by the hardware, virtual memory is managed by the operating system (OS). The access time for a word that is in a memory block paged out by the OS is several orders of magnitude higher than a first level cache hit. Even the difference between a first level cache access and a main memory access is in the order of two magnitudes.

Cache memories for instructions and data are classic examples of the paradigm *Make the common case fast*. Plenty of effort has gone into researching the integration of the instruction cache into the timing analysis [1] and the integration of the cache analysis with the pipeline analysis [8]. The influence of different cache architectures on WCET analysis is described in [9].

Caches in general, and particularly data caches, are usually hard to analyze statically. Therefore, we introduce caches that are organized to speed-up execution time and provide tight WCET bounds. We propose a split cache architecture consisting of: (1) an instruction cache for full methods, (2) a stack cache, (3) a cache for static data, constants, and type information, and (4) a small, fully associative buffer for heap access. Furthermore, we will also consider the integration of program- or compiler-managed scratchpad memory for data storage and inter-processor communication to tighten bounds for hard-to-analyze memory-access patterns.

The shared memory abstraction that is prevalent in (embedded) systems introduces contention on the shared memories. Especially off-chip SDRAM and Flash memories, which are performance bottlenecks, are heavily shared between multiple processors. Cache misses or scratchpad memories prefetches therefore have a highly variable response time due to contention between processors. Worse, even without sharing, response times are variable due to the effects such as different read/write latencies, and bank open/close effects. In T-CREST we will create time-predictable external SDRAM memory controllers, as this is an essential ingredient in every embedded system.

The combination of the TDMA based NoC and the time-predictable memory controller allows, even on a CMP system, to provide upper bounds on memory transactions. This upper bound enables WCET analysis of individual tasks executing on a CMP system.

D. *Compiler and WCET Analysis*

The performance of the dual-issue processor depends on statically scheduled instructions. We argue that all architectural features of a processor shall be exposed to the compiler to generate time-predictable code. Within T-CREST the LLVM compiler framework will be adapted to target Patmos. Furthermore, we will explore compiler optimizations for the WCET instead of the average case execution time.

The processor is intended as a platform to explore various time-predictable design trade-offs and their interaction with WCET analysis techniques as well as WCET-aware compilation. We propose the co-design of time-predictable processor features with the WCET analysis tool, similar to the work by Huber et al. [10] on caching of heap allocated objects in a Java processor. Only features where we can provide a static program analysis shall be added to the processor. This includes, but is not limited to, time-predictable caching mechanisms, chip-multiprocessing (CMP), as well as novel pipeline organizations.

The WCET analysis tool aiT from AbsInt will be adapted to support the VLIW processor Patmos. It is also the platform for exploration of time-predictable processor features. The WCET oriented optimization in the compiler will be tightly integrated with the WCET analysis tool. The WCET tool will provide informations on the worst-case path and basic block timings to guide the optimization process.

E. *Evaluation*

T-CREST will be evaluated by use cases from two industrial partners: GMV and INTECS. GMV will port a set of real-world applications from the aeronautical domain with extreme safety requirements to the T-CREST platform. INTECS will port a specific use case from the railway industry.

VI. Conclusion

To build reliable real-time systems we need architectures that support WCET analysis of tasks. Intuition tells that predictable and especially time-predictable processors shall be easier to analyze and shall result in tight WCET bounds. We would like a notion of quantifiable time predictability to guid our design of time-predictable processors. However, time predictability is not quantifiable. Furthermore, just being predictable, but very slow is not a practical solution. Therefore, we shall compare statically derived WCET bounds of a task on different architectures. The result depends then on (at least) three components: the processor, the compiler, and the WCET analysis tool.

Acknowledgement

I would like to thank the T-CREST project members for the interesting discussions on time-predictable architectures during the project meetings. Contributions came form all partners, in particular from Benny Akesson, Neil Audsley, Florian Brandner, Christian Ferdinand, Kees Goossens, Scott Hansen, Reinhold Heckmann, Guido Ioele, Peter Puschner, Tobias Schoofs, Jens Sparsø, and Jack Whitham. Furthermore, I would like to thank Edward Lee for discussions on predictability versus repeatability.

This work was partially funded under the European Union's 7th Framework Programme under grant agreement no. 288008: Time-predictable Multi-Core Architecture for Embedded Systems (T-CREST).

References

[1] R. Arnold, F. Mueller, D. Whalley, and M. Harmon. Bounding worst-case instruction cache performance. In *IEEE Real-Time Systems Symposium*, pages 172–181, 1994.

[2] C. Berg, J. Engblom, and R. Wilhelm. Requirements for and design of a processor with predictable timing. In L. Thiele and R. Wilhelm, editors, *Perspectives Workshop: Design of Systems with Predictable Behaviour*, number 03471 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2004. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.

[3] S. A. Edwards, S. Kim, E. A. Lee, I. Liu, H. D. Patel, and M. Schoeberl. A disruptive computer design idea: Architectures with repeatable timing. In *Proceedings of IEEE International Conference on Computer Design (ICCD 2009)*, Lake Tahoe, CA, October 2009. IEEE.

[4] S. A. Edwards and E. A. Lee. The case for the precision timed (PRET) machine. In *DAC '07: Proceedings of the 44th annual conference on Design automation*, pages 264–265, New York, NY, USA, 2007. ACM.

[5] H. Falk and J. C. Kleinsorge. Optimal static WCET-aware scratchpad allocation of program code. In *DAC '09: Proceedings of the Conference on Design Automation*, pages 732–737, 2009.

[6] H. Falk and P. Lokuciejewski. A compiler framework for the reduction of worst-case execution times. *Real-Time Systems*, pages 1–50, 2010.

[7] D. Grund, J. Reineke, and R. Wilhelm. A template for predictability definitions with supporting evidence. In P. Lucas, L. Thiele, B. Triquet, T. Ungerer, and R. Wilhelm, editors, *Bringing Theory to Practice: Predictability and Performance in Embedded Systems*, volume 18 of *OpenAccess Series in Informatics (OASIcs)*, pages 22–31, Dagstuhl, Germany, 2011. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[8] C. A. Healy, R. D. Arnold, F. Mueller, D. B. Whalley, and M. G. Harmon. Bounding pipeline and instruction cache performance. *IEEE Trans. Computers*, 48(1):53–70, 1999.

[9] R. Heckmann, M. Langenbach, S. Thesing, and R. Wilhelm. The influence of processor architecture on the design and results of WCET tools. *Proceedings of the IEEE*, 91(7):1038–1054, Jul. 2003.

[10] B. Huber, W. Puffitsch, and M. Schoeberl. WCET driven design space exploration of an object cache. In *Proceedings of the 8th International Workshop on Java Technologies for Real-time and Embedded Systems (JTRES 2010)*, pages 26–35, New York, NY, USA, 2010. ACM.

[11] R. Kirner and P. Puschner. Time-predictable computing. In S. Min, R. Pettit, P. Puschner, and T. Ungerer, editors, *Software Technologies for Embedded and Ubiquitous Systems*, volume 6399 of *Lecture Notes in Computer Science*, pages 23–34. Springer Berlin / Heidelberg, 2011.

[12] B. Lickly, I. Liu, S. Kim, H. D. Patel, S. A. Edwards, and E. A. Lee. Predictable programming on a precision timed architecture. In E. R. Altman, editor, *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES 2008)*, pages 137–146, Atlanta, GA, USA, October 2008. ACM.

[13] I. Liu, J. Reineke, and E. A. Lee. A PRET architecture supporting concurrent programs with composable timing properties. In *Signals, Systems and Computers, 2010 Conference Record of the Forty-Four Asilomar Conference on*, November 2010.

[14] S. Metzlaff, S. Uhrig, J. Mische, and T. Ungerer. Predictable dynamic instruction scratchpad for simultaneous multithreaded processors. In *Proceedings of the 9th workshop on Memory performance (MEDEA 2008)*, pages 38–45, New York, NY, USA, 2008. ACM.

[15] M. Paolieri, E. Quiñones, F. J. Cazorla, G. Bernat, and M. Valero. Hardware support for WCET analysis of hard real-time multicore systems. In *The 36th International Symposium on Computer Architecture (ISCA 2009)*, pages 57–68, Austin, Texas, USA, 20-24, June 2009. ACM.

[16] C. Pitter and M. Schoeberl. A real-time Java chip-multiprocessor. *ACM Trans. Embed. Comput. Syst.*, 10(1):9:1–34, 2010.

[17] P. Puschner. Transforming execution-time boundable code into temporally predictable code. In B. Kleinjohann, K. K. Kim, L. Kleinjohann, and A. Rettberg, editors, *Design and Analysis of Distributed Embedded Systems*, pages 163–172. Kluwer Academic Publishers, 2002. IFIP 17th World Computer Congress - TC10 Stream on Distributed and Parallel Embedded Systems (DIPES 2002).

[18] P. Puschner and A. Burns. Writing temporally predictable code. In *Proceedings of the The Seventh IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2002)*, pages 85–94, Washington, DC, USA, 2002. IEEE Computer Society.

[19] M. Schoeberl. A time predictable instruction cache for a Java processor. In *On the Move to Meaningful Internet Systems 2004: Workshop on Java Technologies for Real-Time and Embedded Systems (JTRES 2004)*, volume 3292 of *LNCS*, pages 371–382, Agia Napa, Cyprus, October 2004. Springer.

[20] M. Schoeberl. A Java processor architecture for embedded real-time systems. *Journal of Systems Architecture*, 54/1–2:265–286, 2008.

[21] M. Schoeberl. Time-predictable cache organization. In *Proceedings of the First International Workshop on Software Technologies for Future Dependable Distributed Systems (STFSSD 2009)*, pages 11–16, Tokyo, Japan, March 2009. IEEE Computer Society.

[22] M. Schoeberl, P. Schleuniger, W. Puffitsch, F. Brandner, C. W. Probst, S. Karlsson, and T. Thorn. Towards a time-predictable dual-issue microprocessor: The Patmos approach. In *First Workshop on Bringing Theory to Practice: Predictability and Performance in Embedded Systems (PPES 2011)*, pages 11–20, Grenoble, France, March 2011.

[23] L. Thiele and R. Wilhelm. Design for timing predictability. *Real-Time Systems*, 28(2-3):157–177, 2004.

[24] T. Ungerer, F. Cazorla, P. Sainrat, G. Bernat, Z. Petrov, C. Rochange, E. Quiñones, M. Gerdes, M. Paolieri, and J. Wolf. Merasa: Multi-core execution of hard real-time applications supporting analysability. *Micro, IEEE*, 30(5):66–75, 2010.

[25] J. Whitham. *Real-time Processor Architectures for Worst Case Execution Time Reduction*. PhD thesis, University of York, 2008.

[26] J. Whitham and N. Audsley. Using trace scratchpads to reduce execution times in predictable real-time architectures. In *Proceedings of the Real-Time and Embedded Technology and Applications Symposium (RTAS 2008)*, pages 305–316, April 2008.

[27] J. Whitham and N. Audsley. Time-predictable out-of-order execution for hard real-time systems. *IEEE Transactions on Computers*, 59(9):1210–1223, 2010.

[28] R. Wilhelm, D. Grund, J. Reineke, M. Schlickling, M. Pister, and C. Ferdinand. Memory hierarchies, pipelines, and buses for future architectures in time-critical embedded systems. *IEEE Transactions on CAD of Integrated Circuits and Systems*, 28(7):966–978, 2009.