Fault-Tolerant Computing in the Multicore Era

376.073 System Architectures for Automation and Control, Winter Term 2020

Severin Jäger
Mat. Nr. 01603004
severin.jaeger@tuwien.ac.at

***Abstract*—Modern cyber-physical systems require computing units featuring both high computational performance and high reliability. Multicore processors allow for applications with high throughput, however they tend to suffer from insufficient dependability for safety-critical applications. This paper reviews the state of the art in reliable computing and investigates both software and hardware approaches for fault-tolerance in the light of increasing computational demands. Furthermore, major application domains and current examples are discussed.**

I. Introduction

Today's complex embedded systems such as aircrafts or advanced driver-assistance systems require safe control systems based on reliable hardware. Modern silicon technology enables extremely complex and powerful processor designs, however typical consumer grade CPUs are not suitable for safety-critical systems. Thus, dedicated fault-tolerant computing hardware is required which currently is facing substantial changes as multicore architectures are emerging even in applications with high reliability standards.

A. Error Sources in Digital Electronics

As technology nodes shrink, the charge held by one transistor is reduced. At the same time, the number of transistors is still following Moore's Law. As a result, the fault rates of modern processors due to cosmic radiation and alpha particles are increasing [1]. In addition, process variations and extreme operating conditions of the transistors decrease reliability even further [2]. Thus, the dependability of processors is no longer only relevant for deep space applications, but even plays an increasing role in general purpose computing.

Transistor level faults can lead to two sorts of errors: Firstly, there are permanent so-called hard errors like latch-ups witch lead to a persistent divergence of the processor state from its specification. In contrast, soft errors are transient. They are typically known as Single Event Upsets (SEUs) and are considered as a major limit for digital hardware reliability [3].

B. Fault-Tolerance Approaches

One way to tackle the described dependability challenges is fault avoidance. This is typically applied in space applications with high SEU rates where radiation-hardened standard cells are used. However, this is costly and by itself insufficient. This paper deals with the fault-tolerant approach which does not aim at avoiding faults, but at operating correctly in their presence. In particular, the aspect of error detection is covered while recovery, diagnosis, and self-repair are not dealt with in detail.

Classically, dependable CPUs are application-specific designs for the industrial, automotive, aerospace, or space market. They are designed with safety certification in mind and achieve high reliability. However, they are expensive and as they are used mainly for control applications, they offer rather limited computational power.

In contrast, the last 15 years of general-purpose computer architecture were dominated by the rise of multicore processors [4]. This has enabled resource-demanding applications like convolutional neural networks (CNNs) even on systems with rather limited computational power. These applications are also interesting for safety-critical systems with notable dependability constraints like autonomous vehicles. As classical reliable computing hardware does not offer excessive multicore capabilities, new techniques are developed. One approach is software-implemented fault tolerance (SIFT) which allows using commercial off-the-shelf (COTS) multicore processors and exploits the inherent redundancy of the parallel cores to reach sufficient reliability levels.

C. Structure of This Paper

This paper is structured as follows: In Section II, the design space for fault-tolerant processors is briefly discussed. Section III covers established hardware approaches and presents some examples while in Section IV methods for software-implemented fault tolerance based on standard components are presented. Section V lists the most significant application domains and refers to interesting designs from the last years. Finally, in Section VI the main points are concluded.

II. Design Space for Fault-Tolerant Processors

All fault-tolerant designs require some form of redundancy. This can be achieved in four domains:

- Space (a functional unit is replicated),
- Time (a computation is executed repeatedly),
- Information (complementary bits), or
- Design (different designs work in parallel) [1].

As redundant designs are very costly, they are only applied in special applications like the Boeing 777 primary flight computer [5]. Temporal redundancy is not feasible for high reliability as a single hardware failure can already cause wrong results. Information redundancy is frequently applied

April 2, 2021

in communication and memory systems [1], however it is still unfavourable for processor designs as it tends to slow them down significantly. Thus, all approaches discussed in the following use some sort of spatial redundancy in the sense of multiple execution units working in parallel.

This parallelism can be implemented at different levels:

- Replication of transistors and metal layers on the process level,
- Duplicated flip-flops on the circuit level,
- Redundant cores and supervisory circuitry on the architectural level, and
- Parallel instructions or threads on the compiler/software level [3].

While low-level measures significantly improve the processor reliability, they decrease the maximum achievable clock frequency and thus the performance of the chip. Thus higher level measures (i.e. architectural or software) are preferred for high-performance chips.

Any redundancy-based design requires some form of voting or checking unit to determine whether the outputs of the redundant units match. This is simplified by the fact that replica determinism [6] is no major problem in the context of CPU design, as the input-output relations for instructions are well-defined.

In the simplest case, two functional units operate in parallel and a checker invalidates the output if the two results diverge. This is called duplication with comparison and allows fault detection [2], however the system can only restart the computation if it detects an error. In case of a permanent fault of one component, the system cannot continue with reliable operation. To achieve fault-tolerant computation in the narrower sense, at least three functional units are required. Typically N-modular redundancy (NMR), usually with N=3 (TMR), is used. This architecture uses three replicated execution units and a majority voter. This can tolerate any kind of error in one module and hardly reduces the performance. However, it comes with a significant area and power consumption overhead. Another important limitation is the single point of failure in the voter [1].

III. Hardware Fault Tolerance Techniques

Currently available processors for safety-critical applications typically feature hardware measures to increase their reliability. Exemplarily, lockstep execution, built-in self-tests and dynamic verification are discussed in this section. Processor cores featuring some of these techniques are available as COTS components. For instance, Arm provides Cortex-R processor cores with dual-core lockstep configuration [3].

A. Lockstep Execution

A lockstep processor consist of at least two processor cores processing the same instruction stream and a voter or checker comparing the results on a cycle-by-cycle basis. Frequently, a dual-core lockstep setup is used for applications in the industrial or automotive sector as they can achieve IEC 61508 SIL 3 or ISO 26262 ASIL D certification [3, 7]. This approach suffers from a high area overhead ($> 100\,\%$), but achieves high levels of fault coverage [2]. In special applications like space, triple-core lockstep execution in a TMR configuration can be used due to its fault-tolerant computation capabilities [3].

One major challenge for lockstep designs is the reduction of common-cause failures. Erroneous behaviour of the clock distribution network, the power supply or the voter can lead to a single point of failure. To avoid this, the two cores are either operated with an offset of a few clock cycles [3] or even connected to independent clock networks [7]. Furthermore, redundant power sources can be used [7]. The single point of failure in the voter and the whole lockstep control logic can be mitigated by using radiation-hardened technology for these parts of the circuit [3].

B. Built-In Self-Tests

One rather simple, yet effective technique are built-in self-tests. They require some additional circuitry that performs checks periodically or during idle time and feature a non-redundant way to detect errors. As their area overhead is rather small, they are frequently used in the context of manufacturing testing [2]. However, they have to be combined with other measures to reach high levels of reliability.

C. Dynamic Verification

Dynamic verification is based on additional checker hardware that observes certain invariants that are known to be true under error-free operation. In case of CPU design, these invariants might for instance be related to control flow, data flow or cache coherence. This technique is mainly suitable for complex CPU designs, as the overhead for the additional checks is relatively small compared to large cores [2]. Again, this approach is insufficient for safety-critical systems due to its limited fault coverage.

IV. Software-Implemented Fault Tolerance with COTS components

While general-purpose processors have developed towards powerful multicore systems during the last decade, the aforementioned dependable CPUs are still limited to rather small chips due to their high cost and certification effort. However, several applications like image processing or neural network inference are desirable for critical systems, require relatively high computational power, and benefit from multicore architectures.

Typically, multicore processors do not feature comprehensive hardware fault-tolerance [8]. However, the are implicitly redundant as they consist of several homogeneous cores. This can be exploited in order to increase the overall reliability of the computations performed on the processor by the means of software-implemented fault tolerance. This is attractive as the development of dedicated fault-tolerant processors is several years behind COTS CPU development. Particularly, this is the case for radiation-hardened space components [9]. SIFT is a chance to increase the computational performance in reliability-constrained applications significantly. As emerging

applications like machine vision, neural networks or sensor fusion require significant processing resources, there is a demand for this development.

Another advantage of fault handling on the software level is that not all hardware faults have to be treated. As long as they do not alter any state visible for the software, there is in the case of soft errors no need to take action. This reduces the number of faults to be resolved significantly [10].

The techniques discussed in this section again deal with spatial redundancy in the sense of software entities operating in parallel. Similar to the aforementioned hardware measures, software reliability techniques can be implemented at different levels. They include

- Replication of instructions on the compiler level,
- Execution of parallel threads on the operating system or task scheduler level, and
- Redundant applications on the middleware level which is not treated in detail in this paper.

A. Compiler Techniques

Compiler-based techniques are usually based on NMR redundancy on the instruction level. One well-understood approach is error detection by duplicated instructions (EDDI) [11]. It is based on redundant CPU registers which hold the results of duplicated instructions. In addition, checker or voting instructions are required to implement NMR. As all those instructions are part of one instruction stream, they are scheduled to one core, thus the redundancy of multicore systems is not utilised. However, EDDI exploits instruction-level parallelism (ILP) well, as it adds many independent instructions. As a result, the runtime overhead is in the order of $10\,\%$ for TMR [9]. However, the code size is increased drastically due to the added instructions.

B. Redundant Multithreading and its Variants

An higher-level approach for fault detection is the duplication of programs or notable program parts. One opportunity is the replication of processes, however this imposes a significant performance overhead [10]. Thus, replication on the thread level is the state of the art. The simplest approach is redundant multithreading (RMT) [12]. It executes two identical threads and compares their results. The threads are however still scheduled to the same core. The runtime overhead can be reduced significantly on CPUs with simultaneous multithreading (SMT) as the utilisation of the core can be increased by the additional independent threads [12].

In order to utilise multicore processors, SMT has been extended to chip-level redundant threading (CRT). In this case, the threads are no longer executed on the same core, but distributed over the multicore processor. This results in a loose lockstep configuration which outperforms tight lockstep approaches [12].

RMT and CRT can implement different redundancy schemes such as duplication with comparison or TMR while non-replicated threads are processed simultaneously. This flexibility can be exploited by mixed-criticality systems which execute tasks with different real-time and dependability requirements in parallel. In [13], a self-adaptive approach is presented which applies the most suitable execution mode for each task.

C. Non-Redundant Techniques

So far, the discussed techniques were based on duplication of software primitives. Similar to the built-in self-tests or dynamic verification covered in Section III, non-redundant software routines can be used to improve the reliability of a processor.

One way to do so are control-flow techniques. In contrast to the replication on the instruction level (EDDI), these techniques do not detect faults in the data flow but protect the control flow by means of detecting erroneous branches or jumps [14]. To achieve this, the code is split into basic blocks without branches and jumps. So valid jumps have to be at the end of a basic block and to the beginning of a basic block. This requires additional instructions that assign signatures to the basic blocks and checks. The overall runtime overhead is in the order of $55\,\%$ to $85\,\%$ depending on benchmark and algorithm [14].

Furthermore, anomaly detection can be implemented in software. These techniques are usually rather lightweight in terms of performance overhead, but perform rather simple checks and reach only insufficient coverage for operation in safety-critical systems [2].

D. Limits of Software Techniques

The most obvious bottleneck for software-implemented fault tolerance is the reliability of the underlying hardware. In case only one core is used, almost any hardware fault can result in a single point of failure and even in the case of multicore processors, there are several shared functional units like the I/Os and the power supply. Their failure will cause common-cause faults for all redundant instructions or threads. So especially in the context of ultra-dependable systems like aerospace or space applications, additional hardware measures are required. In general, it has to be concluded that SIFT can never recover all types of errors [8].

Furthermore, SIFT on multicore processors requires reliable communication between the cores as well as fault isolation [8]. In the case of shared-memory multicore systems, the latter is not trivial as memory faults can affect the redundant threads operating on different cores. For this reason, the memory of dependable systems is usually protected with error-correcting codes in hardware [9]. However, in the case of caches this imposes a significant overhead [1].

V. Application Domains and Examples

A. Space

Space applications are traditionally dominated by ultra-reliable radiation-hardened space processors. However, they are extremely expensive and consume much power [3]. Additionally, their development is years behind standard components for safety-critical ground applications. As the computational requirements for space missions increase with better

sensors, vision processing and high data rates [15], multicore systems based on COTS cores are interesting for the space market.

One development is the Arm Triple Core Lock-Step Processor [3] which uses only slightly adapted cores from terrestrial applications in an TMR setup. Only a very small portion of the chip ($< 4\,\%$) requires radiation-hardening, thus the processor is both powerful and energy-efficient.

An even more radical approach is the NASA High Performance Spaceflight Computing Platform [15] which features a heterogeneous architecture based on Arm cores. It consist of a Timing, Reset, Config & Health Controller based on a TMR setup, a Real Time Processing Subsystem with a dual-core lockstep processor and the High Performance Processing Subsystem composed by eight performant cores. In addition to the redundant processors, it uses information redundancy, fault detection on the system software level as well as fault and redundancy management services in the middleware. This setup is sufficient for unmanned missions, however in the context of life control systems on a manned gateway, four of these chips are used in combination with triplicated Time-Triggered Ethernet to construct the Notional Two Fault Tolerant System.

In contrast, satellites in the low earth orbit (LEO) are exposed to less cosmic radiation and can therefore be based on cheaper and less redundant hardware. For these applications, COTS hardware in combination with software-implemented fault tolerance against soft errors and FPGA reconfiguration against hard errors are promising approaches [16].

B. Aerospace

The transformation towards COTS components has already taken place in the aerospace market [17]. During the last decades, integrated modular avionics architectures have emerged. They require central processing units with notable performance. Multi-core processors can cope with the increasing computational requirements of these units. One major challenge for their adoption in commercial aircrafts is flight safety certification due to the high complexity of these CPUs.

C. Terrestrial Safety-Critical Applications

In the safety-critical systems market sufficiently reliable COTS processors are state of the art. Frequently, lockstep processors are used, as most industrial systems are fail-safe systems. Still, established dual-core lockstep processors cannot fulfil the requirements of fail-operational systems like autonomous vehicles as their fault-recovery process (i.e. rollback or reset) causes a significant delay and thus potential deadline misses. Due to cost considerations, triplicated hardware is not desirable for many terrestrial applications. Thus, different approaches like the commander/monitor approach [18] are investigated for autonomous driving.

Automotive architectures are highly distributed. This is in principle beneficial for fault tolerance, however fault detection becomes a rather complicated distributed problem. Furthermore, several applications are running on one system on a chip with a hypervisor managing the hardware access. These challenges can be addressed with software solutions [19].

However, certain critical control systems in vehicles rely on dedicated dependable hardware like lockstep processors which fulfil the ISO 26262 ASIL D requirements [7]. As the classical lockstep approach discussed in Section III cannot offer the computational power required for convolutional neural networks for autonomous driving, parallel accelerators capable of lockstep operation might be a way to combine high throughput with safety requirements [20].

VI. Conclusion

This paper discussed both hardware and software approaches for fault-tolerant processors in the light of emerging multicore architectures. Most fault-tolerant techniques are based on well-understood redundancy schemes like duplication with comparison or TMR. However, they can be implemented at different levels of the design, ranging from transistors to whole programs being duplicated. Typical hardware redundancy is based on the duplication of cores, usually in a dual-core lockstep configuration. This suffices many common requirements, however modern multicore CPUs offer way more processing power and inherent redundancy. This can be exploited with software-based redundancy which uses redundant threads scheduled to different cores. Unfortunately, the reliability of complex multicore processors is not sufficient for critical applications. As a result, for highly complex applications with significant dependability constraints like spaceflight both approaches are combined in order to create powerful and ultra-dependable computing platforms.

References

[1] D. J. Sorin, "Fault Tolerant Computer Architecture," *Synthesis Lectures on Computer Architecture*, vol. 4, no. 1, pp. 1–104, Jan. 2009.

[2] D. Gizopoulos, M. Psarakis, S. V. Adve, P. Ramachandran, S. K. S. Hari, D. Sorin, A. Meixner, A. Biswas, and X. Vera, "Architectures for online error detection and recovery in multicore processors," in *2011 Design, Automation & Test in Europe*, IEEE, Mar. 2011.

[3] X. Iturbe, B. Venu, E. Ozer, J.-L. Poupat, G. Gimenez, and H.-U. Zurek, "The Arm Triple Core Lock-Step (TCLS) Processor," *ACM Transactions on Computer Systems*, vol. 36, no. 3, pp. 1–30, Aug. 2019.

[4] K. Rupp, *Microprocessor trend data*, 2020. [Online]. Available: https://github.com/karlrupp/microprocessor-trend-data.

[5] Y. C. Yeh, "Triple-triple redundant 777 primary flight computer," in *1996 IEEE Aerospace Applications Conference. Proceedings*, vol. 1, 1996, 293–307 vol.1.

[6] S. Poledna, A. Burns, A. Wellings, and P. Barrett, "Replica determinism and flexible scheduling in hard real-time dependable systems," *IEEE Transactions on Computers*, vol. 49, no. 2, pp. 100–111, 2000.

[7] J. Han, Y. Kwon, Y. C. P. Cho, and H.-J. Yoo, "A 1GHz fault tolerant processor with dynamic lockstep and self-recovering cache for ADAS SoC complying with ISO26262 in automotive electronics," in *2017 IEEE Asian Solid-State Circuits Conference (A-SSCC)*, IEEE, Nov. 2017.

[8] C. Villalpando, D. Rennels, R. Some, and M. Cabanas-Holmen, "Reliable multicore processors for NASA space missions," in *2011 Aerospace Conference*, IEEE, Mar. 2011.

[9] Y. Nezzari and C. P. Bridges, "Compiler extensions towards reliable multicore processors," in *2017 IEEE Aerospace Conference*, IEEE, Mar. 2017.

[10] J. P. Walters, R. Kost, K. Singh, J. Suh, and S. P. Crago, "Software-based fault tolerance for the Maestro many-core processor," in *2011 Aerospace Conference*, IEEE, Mar. 2011.

[11] N. Oh, P. Shirvani, and E. McCluskey, "Error detection by duplicated instructions in super-scalar processors," *IEEE Transactions on Reliability*, vol. 51, no. 1, pp. 63–75, Mar. 2002.

[12] S. Mukherjee, M. Kontz, and S. Reinhardt, "Detailed design and evaluation of redundant multi-threading alternatives," in *Proceedings 29th Annual International Symposium on Computer Architecture*, IEEE Comput. Soc, 2002.

[13] C. Bolchini, M. Carminati, and A. Miele, "Self-Adaptive Fault Tolerance in Multi-/Many-Core Systems," *Journal of Electronic Testing*, vol. 29, no. 2, pp. 159–175, Apr. 2013.

[14] E. Chielle, F. Rosa, G. S. Rodrigues, L. A. Tambara, J. Tonfat, E. Macchione, F. Aguirre, N. Added, N. Medina, V. Aguiar, M. A. G. Silveira, L. Ost, R. Reis, S. Cuenca-Asensi, and F. L. Kastensmidt, "Reliability on ARM Processors Against Soft Errors Through SIHFT Techniques," *IEEE Transactions on Nuclear Science*, pp. 1–9, 2016.

[15] A. Cudmore, "High-performance spaceflight computing (HPSC) middleware overview," *2018 Flight Software Workshop*, 2018. [Online]. Available: https://ntrs.nasa.gov/citations/20190001377.

[16] C. M. Fuchs, N. M. Murillo, A. Plaat, E. van der Kouwe, and P. Wang, "Towards Affordable Fault-Tolerant Nanosatellite Computing with Commodity Hardware," in *2018 IEEE 27th Asian Test Symposium (ATS)*, IEEE, Oct. 2018.

[17] J. Athavale, R. Mariani, and M. Paulitsch, "Flight Safety Certification Implications for Complex Multi-Core Processor based Avionics Systems," in *2019 IEEE 25th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, IEEE, Jul. 2019.

[18] A. Mehmed, M. Antlanger, and W. Steiner, "The Monitor as Key Architecture Element for Safe Self-Driving Cars," in *2020 50th Annual IEEE-IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)*, IEEE, Jun. 2020.

[19] T. Bijlsma, A. Buriachevskyi, A. Frigerio, Y. Fu, K. Goossens, A. O. Ors, P. J. van der Perk, A. Terechko, and B. Vermeulen, "A Distributed Safety Mechanism using Middleware and Hypervisors for Autonomous Vehicles," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, Mar. 2020.

[20] K. Matsubara, L. Hanno, M. Kimura, A. Nakamura, M. Koike, K. Terashima, S. Morikawa, Y. Hotta, T. Irita, S. Mochizuki, H. Hamasaki, and T. Kamei, "4.2 A 12nm Autonomous-Driving Processor with 60.4TOPS, 13.8TOPS/W CNN Executed by Task-Separated ASIL D Control," in *2021 IEEE International Solid- State Circuits Conference (ISSCC)*, IEEE, Feb. 2021.