Runtime Verification and Validation for Multi-Core Based On-Board Computing

Hans P. Zima and Mark L. James
Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA 91109
{zima,mjames}@jpl.nasa.gov

Introduction

Future deep-space missions will need support for autonomy and enhanced science processing. The integration of emerging commodity multi-core technology into space-borne systems can provide the required performance; however, protecting such systems against faults has become a critical research issue. In this paper we present an approach to fault tolerance based on an introspection framework that supports runtime monitoring of program execution and feedback-oriented recovery. We discuss the relationship of this approach to traditional Verification and Validation (V&V) and propose methods for the automatic generation of assertions from static and dynamic analysis.

A New Paradigm for Spacecraft Architectures

Today's space-qualified on-board systems rely primarily on fault tolerance provided by radiation-hardened components—an approach that is not expected to scale with the requirements of future missions. Recent developments in the area of commercial multi-core architectures have resulted in simpler processor cores, enhanced efficiency in terms of performance per Watt, and a dramatic increase in the number of cores on a chip. Examples include the Cell Broadband Engine, Tilera Corporation's Tile64 [Tile64], and the 80-core Intel chip announced for 2011 that will reach Terascale performance.

These trends suggest a new paradigm for spacecraft architectures, in which the ultra-reliable radiation-hardened core component responsible for control, navigation, data handling, and communication will be complemented with a Commercial-Off-The-Shelf (COTS)-based multi-core system for autonomy and science processing, providing the basis for a powerful parallel on-board supercomputing capability. However, bringing COTS components into space implies the need to address their vulnerability to faults, in particular the transient effects caused by Single Event Upsets (SEUs), which change the state of a single bit in a register or memory. SEUs can cause multiple problems, such as the corruption of instruction fetch/decode, address selection, memory units, synchronization, communication, or signal/interrupt processing. The effects resulting from such faults may range from the (unrecognized) use of corrupted data to the execution of wrong or illegal instructions, branches, and data accesses in the program. Hangs or crashes of the program, as well as unwarranted exceptions are other possible consequences. In a distributed system, transient faults can cause communication errors, livelock, deadlock, data races, or arbitrary Byzantine failure.

We have developed a software-based approach to fault tolerance that relies on an introspection framework supporting application-adaptive fault tolerance for mission software executing on multi-core based high-performance on-board architectures. Introspection performs dynamic monitoring, analysis, and feedback-oriented management of applications, supported by a real-time inference engine [IJHPCA.09]. The relationship between an application and the introspection system is based upon software-implemented sensors and actuators, as outlined in Figure 1.

In this paper we discuss the relationship of the introspection-based approach to traditional Verification and Validation (V&V) and propose methods for the automatic generation of assertions from static and dynamic analysis.

Relationship between V&V and Introspection

A program executing on a space-borne computing system may be subject to software design errors as well as faults caused by hardware malfunction or environmental influences such as radiation and thermal effects. We propose a systematic integration of introspection-based fault tolerance with traditional V&V technology.

Consider a program, P, over a given input domain, and assume that the intended behavior of P is defined by a formal specification. *Verification* of P in the traditional sense implies a static proof—performed before execution of the program—that for all legal inputs, the application of P to an input value conforms to the specification. Thus, verification is a methodology that seeks to avoid faults. A myriad of verification techniques related to all aspects of sequential and parallel programs has been developed over the past decades. They have been highly successful when judiciously applied under the right conditions in well-defined contexts. However, in general verification faces a number of challenges and limitations, including theoretical limits (undecidability and NP-completeness) and practical scalability issues. A second major V&V technique is *test.* As noticed by Dijkstra as early as 1972, tests can prove the existence of an error but never the absence of all errors. Consequently, V&V alone cannot provide a complete solution to the problem of proving the correctness of programs. It is a well-known fact that large programs always contain design errors, no matter how much effort has been invested in verification and test. Thus, complementing V&V with runtime technology for error detection and recovery, as presented by our introspection framework, is essential.

Adaptive Fault Tolerance for On-Board Computing

Our approach to fault tolerance is adaptive in the sense that faults can be handled in a flexible way, depending on the potential damage caused by them. Methods that are useful in this context include assertion-based acceptance tests that check the value of an assertion and transfer control to the introspection system in case of violation, and fault detectors

that can effectively mask a fault by using redundant code based on analysis information. Furthermore, faults in critical sections of the code can be masked by leveraging fixed redundancy techniques such as TMR or NMR. Another technique is the replacement of a function with an equivalent version that implements Algorithm-Based Fault Tolerance (ABFT). Information supporting the generation of assertion based acceptance tests as well as fault detectors can be derived from static or dynamic automatic program analysis, retrieved from domain- or system specific information contained in the knowledge base or can be directly specified by an expert user.

Automatic analysis of program properties relevant for fault tolerance can be based on a rich spectrum of tools and methods. This includes the static analysis of the control and data structures of a program, its intra- and inter-procedural control flow, data flow and data dependences, data access patterns, and patterns of synchronization and communication in multi-threaded programs. Other static tools can check for the absence of deadlocks or race conditions. Profiling from simulation runs or test executions can contribute information on variable ranges, loop counts, or potential bottlenecks. Furthermore, dynamic analysis provides knowledge that cannot be derived at compile time, such as the actual paths taken during a program execution and dynamic dependence relationships. Consider a simple example: an SEU can break the link between an assignment and the use of the assigned value by modifying the target address of the assignment. Possible consequences include an attempt to use an undefined variable, dereference an undefined pointer, or destroy a loop bound. The results of static analysis (as well as results obtained from program profiling) can be exploited for fault detection and recovery in a number of ways, including the generation of assertions in connection with specific program locations or program regions, which express known properties that need to hold in such places. The generation of such assertions can be based on the analytically derived information in combination with the generation of code that records the corresponding relationships at runtime. A more elaborate technique that exploits static analysis for the generation of a fault detector using redundant code generation can be based on program slicing. Some techniques applied to sequential programs can be generalized to deal with multi-threaded programs, in particular to programs whose execution is organized as a data-parallel set of thread according to the Single-Program-Multiple-Data (SPMD) paradigm. This is highly relevant since the vast majority of parallel scientific applications belong to this category.

Related Work

Our work is specifically related to efforts for providing fault-tolerant on-board computing in space, including the Remote Exploration and Experimentation (REE) project [Some.99] and NASA's Millenium ST-8 project [Samson.07]. Some significant work has been done in the area of assertions. The EAGLE system [EAGLE] provides an assertion language with temporal constraints. The Design for Verification (D4V) system [DV4] uses dynamic assertions, which are objects with state that are constructed at design time and tied to program objects and locations. Language support for assertions and invariants has been provided in Java 1.4, Eiffel for pre- and post conditions in Hoare's logic, and the Java Modeling Language (JML). Intelligent resource management in an introspection-based approach has been proposed in [ISI.07]. Finally, ideas similar to introspection have been used by Iyer and co-workers for application-specific security [Iyer.07].

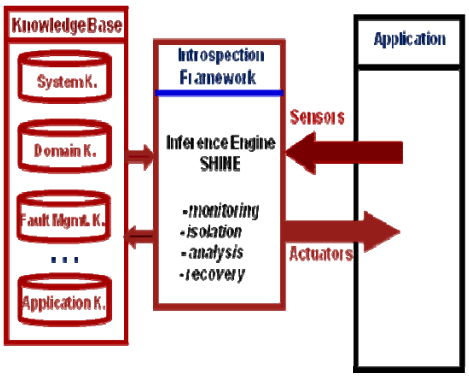

Figure 1: Introspection Framework for Fault Tolerance

References

[EAGLE] A.Goldberg,K.Havelund, and C.McGann: Runtime Verification for Autonomous Spacecraft Software. *Proc.2005 IEEE Aerospace Conference, pp.507-516, March 2005*

[IJHPCA.09] M.L.James,A.A.Shapiro,P.L.Springer, and H.P.Zima: Adaptive Fault Tolerance for Scalable Cluster Computing in Space. *Intl. Journal of High Performance Computing Applications, 2009 (in print)*

[Iyer.07] R.K.Iyer et al.: Toward Application-Aware Security and Reliability.
IEEE Security and Privacy, 5(1):57-62, 2007.

[ISI.07] D.-I.Kang,J.Suh,J.O.McMahon,S.P.Crago: Preliminary Study toward Intelligent Run-Time Resource Management Techniques for Large Multi-CoreArchitectures. *Proc.HPEC'07, September 2007*

[DV4] P.C.Mehlitz and J.Penix: Design for Verification with Dynamic Assertions.
Proc.2005 29th Annual IEEE/NASA Software Engineering Workshop (SEW'05), 2005

[Samson.07] J.Samson et al.: High Performance Dependable Multiprocessor II.*Proc.2007 IEEE Aerospace Conference, Big Sky, MT (March 2007)*

[Some.99] R.Some and D.Ngo: REE: A COTS-Based Fault-Tolerant Parallel Processing Supercomputer for Spacecraft Onboard Scientific Data Analysis.
Proc.Digital Avionics Systems Conference, pp.7.B.3-1-7.B.3-12, 1999

[Tile64] Tile 64 Processor Family
http://www.tilera.com (2007)