Reliable and Efficient Concurrent Synchronization for Embedded Real-Time Software

Damian Dechev
Texas A&M University
Email: dechev@tamu.edu

Bjarne Stroustrup
Texas A&M University
Email: bs@cs.tamu.edu

***Abstract*—The high degree of autonomy and increased complexity of future robotic spacecraft pose significant challenges in assuring their reliability and efficiency. To achieve fast and safe concurrent interactions in mission critical code, we survey the practical state-of-the-art nonblocking programming techniques. We study in detail three nonblocking approaches: (1) CAS-based algorithms, (2) Software Transactional Memory, and (3) Predictive Log Synchronization. We identify a framework of seven critical evaluation criteria and analyze the strengths and weaknesses of each approach. Our study investigates how the application of nonblocking synchronization can help eliminate the problems of deadlock, livelock, and priority inversion and at the same time deliver a performance improvement in embedded real-time software.**

I. Introduction

Future space exploration projects, such as Mars Science Laboratory (MSL), demand the engineering of some of the most complex embedded software systems. The notion of concurrency is of critical importance for the design and implementation of such systems. The software development and certification methodologies applied at NASA [6] do not reach the level of detail of providing guidelines for the engineering of reliable concurrent software. In this work, we present a detailed survey of the state-of-the-art *nonblocking* programming techniques that can help in implementing *efficient* and *safe* concurrent interactions in mission critical embedded code.

A. Parallelism and Complexity

The most common technique for controlling the interactions of concurrent processes is the use of mutual exclusion locks. The application of mutually exclusive locks poses significant safety hazards and incurs high complexity in the testing and validation of mission-critical software. Even for efficient and highly optimized locks, the interdependence of processes implied by the use of locks introduces the dangers of *deadlock*, *livelock*, and *priority inversion*. The incorrect application of locks is hard to determine with the traditional testing procedures and a program can be deployed and used for a long period of time before the flaws can become evident and eventually cause anomalous behavior.

B. Nonblocking Synchronization

To achieve higher safety and gain performance, we suggest the application of *nonblocking synchronization*. A concurrent object is *nonblocking* if it guarantees that *some* process in the system will make progress in a *finite* amount of steps [3]. Nonblocking algorithms do not apply mutually exclusive locks and most commonly rely on a set of atomic primitives supported by the hardware architecture. The most ubiquitous and versatile data structure in the ISO C++ Standard Template Library is *vector*, offering a combination of dynamic memory management and constant-time random access. Because of the vector's wide use and challenging parallel implementation of its nonblocking dynamic operations, we illustrate the efficiency of each approach with respect to its applicability for the design and implementation of a shared nonblocking vector. A number of pivotal concurrent applications in the Mission Data System [4] framework employ a shared STL vector (in all scenarios protected by mutually exclusive locks). Such is the Data Management Service library described by Wagner in [8].

II. Analysis and Results

We survey the practical state-of-the-art nonblocking programming techniques. We study in detail three approaches:

1) CAS-based algorithms design [1]
2) Software Transactional Memory (STM) [2]
3) Predictive Log Synchronization (PLS) [7]

We demonstrate how each nonblocking technique can be utilized to implement a concurrent shared vector. We analyze each approach according to our framework of seven evaluation criteria: 1. preservation of program semantics and space and time complexities, 2. nonblocking guarantees (wait-free vs. lock-free vs. obstruction-free) [3], 3. correctness model (linearizability, Lamport's consistency, others) [3], 4. portability (assumptions and reliance on atomic primitives and hardware instructions), 5. simplicity and ease of use, 6. high degree of parallelism and fast performance, 7. thread-safety (any hazards or race conditions that each particular approach might need to address). We investigate how the application of each synchronization technique can help eliminate the problems of *deadlock*, *livelock*, and *priority inversion*. Our experimental evaluation compares the performance of the three nonblocking approaches and provides an estimate of the possible performance gains of each in contrast to the application of some of the most optimal blocking techniques.

III. Impact for Space Systems

A study on the challenges for the development and certification of modern spacecraft software by Lowry [5] reveals that in July 1997 The Mars Pathfinder mission experienced a number of anomalous system resets that caused an operational delay and loss of scientific data. The follow-up analysis identified the presence of a *priority inversion* problem caused by the low-priority meteorological process blocking the the high-priority bus management process. Providing reliable and efficient concurrent synchronization is of significant importance for the design and validation of complex autonomous future robotic spacecraft.

IV. Conclusion

In this study we investigate how the application of nonblocking synchronization can help eliminate the problems of deadlock, livelock, and priority inversion in embedded real-time mission critical software. We apply a comparison framework consisting of seven evaluation criteria to analyze three known approaches for nonblocking synchronization and explain in detail their strengths and weaknesses. We measure the performance of each nonblocking approach and indicate the possible performance gains in contrast to the application of some of the most optimal blocking techniques. Understanding the advantages (over mutual exclusion) as well as the usability and performance trade-offs of the modern nonblocking programming techniques is of critical importance for engineering reliable and efficient concurrent flight software.

References

[1] D. Dechev, P. Pirkelbauer, and B. Stroustrup. Lock-Free Dynamically Resizable Arrays. In *OPODIS 2006*.
[2] D. Dice and N. Shavit. Understanding tradeoffs in software transactional memory. In *CGO*, 2007.
[3] M. Herlihy. The art of multiprocessor programming. In *PODC '06*, pages 1–2, New York, NY, USA, 2006. ACM.
[4] M. Ingham, R. Rasmussen, M. Bennett, and A. Moncada. Engineering Complex Embedded Systems with State Analysis and the Mission Data System. In *AIAA*, 2004.
[5] M. R. Lowry. Software Construction and Analysis Tools for Future Space Missions. In *TACAS*, 2002.
[6] RTCA. Software Considerations in Airborne Systems and Equipment Certification (DO-178B), 1992.
[7] O. Shalev and N. Shavit. Predictive log synchronization. In *EuroSys 06*.
[8] D. Wagner. Data Management in the Mission Data System. In *Proceedings of the IEEE System, Man, and Cybernetics Conference*, 2005.