Astronomical Data Analysis Software and Systems X
ASP Conference Series, Vol. 238, 2001
F. R. Harnden Jr., F. A. Primini, and H. E. Payne, eds.

Software Fault Tolerance for Low-to-Moderate Radiation Environments

R. Sengupta[1], J. D. Offenberg[1], D. J. Fixsen[1], D. S. Katz[2], P. L. Springer[2], H. S. Stockman[3], M. A. Nieto-Santisteban[3], R. J. Hanisch[3], J. C. Mather[4]

Abstract. The primary intention of NASA's Remote Exploration and Exploration (REE) project is to use commercial off-the-shelf, scalable, low-power, fault-tolerant, high-performance computation in space. Most of the faults caused by the radiation environments in regions of space of interest to REE (Deep Space, Low Earth Orbit) are transient, single event effects. Some of these faults can cause errors at different application levels. System and applications software can potentially detect and correct some or many of these errors. We discuss different software fault tolerance approaches such as replication, voting, and masking with a focus on algorithm-based fault-tolerance. Combined software and hardware approaches such as fault avoidance, redundancy, masking, and reconfiguration are discussed. These approaches allow trade-offs between reliability, power, cost, and computation power for spacecraft in a low-to-moderate radiation environment.

1. Introduction

The Remote Exploration and Exploration (REE) project's global objective is to move ground-based supercomputers to space. This means building a high-performance, reliable, low-power, parallel computer for space applications, using mostly commercial off-the-shelf (COTS) components. REE intends to achieve this goal by using software-implemented fault tolerance (SIFT) techniques that will perform fault detection, isolation, and recovery without compromising reliability or performance. In order to gain experience with SIFT techniques, several generations of test beds are being built to perform fault tolerance experiments. These test beds will be provided with a SIFT middle-ware layer residing between the operating system and the applications, and between the operating system and the hardware. A software fault injector for simulation of radiation faults

[1]Raytheon ITSS, 4500 Forbes Blvd, Lanham MD 20706

[2]Jet Propulsion Laboratory, 4800 Oak Grove Dr., Pasadena CA 91109, http://www-ree.jpl.nasa.gov

[3]Space Telescope Science Institute, 3700 San Martin Dr., Baltimore MD 21818

[4]Code 685, NASA's Goddard Space Flight Center, Greenbelt MD 20771

has been generated to test the system and study its performance. REE has the goal of a flight test in 2005.

2. Hardware Fault Tolerance

Hardware fault tolerance can be attempted at different levels and in different ways. *Fault avoidance* can be defined as using better components or radiation hardened components to avoid single event upsets caused by cosmic ray hits. Although this is the only approach chosen by current projects and ranks high in reliability, the cost of radiation hardened components is the main tradeoff. Power consumption is higher and computational power is lower than for systems based on radiation hardened components versus parallel COTS-based systems. *Fault detection* would stop the system when a fault occurs. This has the advantage of identifying fatal errors but at a loss of throughput. *Masking redundancy* means running in the presence of faults. A few processors can run the same program and vote to identify errors in any single processor. Errors can be masked from application software. No software rollbacks will be required to fix errors. *Reconfiguration* means removing failed components from the system. When failure occurs in a component, its effects on the remaining portion of the system can be isolated. We are not concerned about how it failed. We expect to have enough other similar components to take its place. These are typically off-the-shelf processors, disk (for ground-based test beds only), and memory units but the desired reliability units e.g., EDAC, CRC checks, may not be built into these components. A large number of functional units with spares can be used, which will be switched automatically to replace a failing component. One can build desired the level of redundancy through system-level algorithms to coordinate these components.

3. Software Implemented Fault Tolerance (SIFT)

SIFT is a contrasting approach to a fault tolerant multi-processor. *Fault isolation* can be obtained by physical isolation of failed hardware components. *Fault masking* is enhanced through multiple buses or a redundantly linked network over which multiple copies of data are transmitted. Reliable fault-masking uses non-faulty elements to compensate for the effects of faulty elements. *Voting* takes place as a function of the software. Critical tasks in SIFT are required to be iterative tasks such that voting is done on the state of data before each iteration. This guarantees that each processor is working on the same loop. If an error is discovered in the vote on this state of the system, the software attempts to locate and remove the faulty components. This can be done by changing the bus of the presumed faulty processor. If the fault persists after the move, the processor is retired. If the fault disappears, the bus is returned. Thus, the SIFT can configure around the fault. Each processor can make a report of a failed component to the run-time supervisor. Depending on the type of fault, it may not be decidable which processor, or bus, is actively failing. However, the system can continue to run in a fault-masking mode (Banâtre & Lee, 1994).

The clocks in SIFT are not necessarily synchronous. This is a major advantage, as clock synchronization can be a significant added expense.

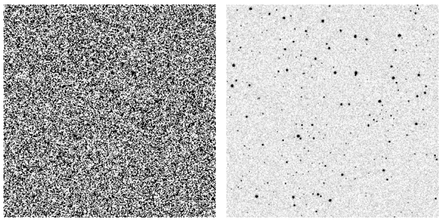

Figure 1. Sample 10000s NGST image, before and after deglitching (Fixsen, 2000).

Replication ensures reliability but is expensive in terms of hardware or runtime cost. The idea is to take a majority vote on a calculation replicated N times. A hardware solution requires N times the number of processors, as well as a reliable voter. Software solutions require each processor to run N copies of surrounding computations and then vote on the result. This slows down the computation by at least a factor of N. However, the savings from using COTS components could offset the added costs. A candidate application for this approach appears in Figure 1.

3.1. Algorithm-Based Fault Tolerance (ABFT)

In the broadest sense, algorithm-based fault tolerance (ABFT) refers to a self-contained method for detecting, locating, and correcting faults with a software procedure. REE is looking at techniques for numerical linear matrix multiplication, LU decomposition, QR decomposition, single value decomposition (SVD), and Fast Fourier Transforms (FFT). These techniques typically involve augmenting data with a (linear) checksum, whose value is predictably transformed by the linear algorithm. Checksum encoding is embedded in the data.

ABFT can detect, locate, and correct faults by exploiting the structure of numerical operations. REE is focusing on error detection, using result checking. This implies that a simple post-condition exists which can be checked at a cost that is low compared with the cost of the base algorithm. If (transient) errors are detected, the routine can be re-run. Many REEs science applications spend much of their time in core numerical operations; for example, NGST's phase retrieval code[5] (Figure 2) spends ∼70% of its time on FFTs.

4. Faults and Errors

The radiation environment causes faults at the hardware level; many of which become errors at the application or system level. About 99.9% of faults are transient single event effects; faults can cause errors that cause the node or application to crash or hang. System software can detect such errors. Restarting the application, rollback, or rebooting will be acceptable solutions in those cases.

Hard to detect errors would change the application data, allowing the application to complete but with incorrect output. Applications can detect most

[5] `http://www.ngst.nasa.gov/cgi-bin/doc?Id=486`

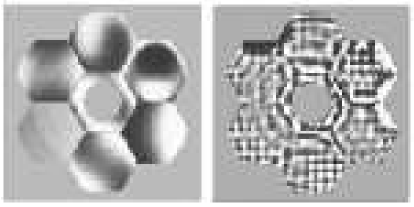

Figure 2. Sample wavefront corrected by 349 actuator deformable mirror and primary mirror segments, using NGST's phase retrieval code.

errors using ABFT, assertion checking, and other application-level techniques. The ability to correct errors with low overhead is a major advantage.

Huang and Abraham (1984) proposed the idea of using properties of the solution as acceptable tests: *"Since we test the intermediate results for correctness with respect to the algorithm, the end solution is correct if the intermediate results are correct. If processor errors occur that do not affect the solution, then they are not errors."* One application of this principle would be to only detect and correct processor errors in excess of either the inherent noise of the data set or the desired accuracy in the end result.

5. Conclusion and Future Work

Combined hardware and software fault tolerance approaches would be most effective for space-borne components in a harsh environment. The combination of COTS and SIFT can dramatically lower the cost and power usage while increasing computation power. *"A system should be as simple as possible in order to achieve its required function—and no simpler"* (Banâtre & Lee).

Future work planned for this study includes making provision for corruption of a counter or corruption of input data during the computation. It is also necessary to develop a better practical understanding of failure rates to have confidence in ultimate reliability; methods for detecting faults during the correction procedures themselves are also needed.

Acknowledgments. These studies are supported by NASA's Remote Exploration and Experimentation Project, which is administered at the Jet Propulsion Laboratory.

References

Huang & Abraham 1984, IEEE TC, 33(6), 518

Wang & Jha 1994, IEEE TC, 43(7), 849

Banâtre, M. & Lee, P. 1994, "Hardware and Software Architectures for Fault Tolerance", Springer Verlag

Fixsen, D. J., et al. 2000, PASP, 112, 1350