Development of a Spaceborne Embedded Cluster

Daniel S. Katz and Paul L. Springer
Jet Propulsion Laboratory
California Institute of Technology
{Daniel.S.Katz, Paul.L.Springer}@jpl.nasa.gov

Abstract

Over the last decade and continuing into the foreseeable future, a trend has developed in the spacecraft industry of both number of missions and the amount of data taken by each mission increasing faster than bandwidth capabilities to send these data to Earth. The result of this trend is a bottleneck between data gathering (on-board) and data analysis (on the ground.) This bottleneck can be overcome by performing data analysis on-board and only transferring the results of this analysis to the ground, rather than the raw data. One attempt to do this is being made by the NASA HPCC Remote Exploration and Experimentation (REE) Project, which is developing spaceborne embedded clusters. Spaceborne embedded clusters share many characteristics of traditional, ground-based clusters such as POSIX-compliant operating systems and message-passing applications, but also have significant differences, including packaging and the need for fault-tolerance and real-time scheduling in software. This paper discusses these similarities and differences, and how they impact application development.

1. INTRODUCTION

Over the last decade, the National Aeronautics and Space Administration (NASA) has adopted a new strategy for its spacecraft missions: "better, faster, cheaper." In September 1992, NASA Administrator Daniel S. Goldin told the World Space Congress "We must push out beyond our comfort zone, and make ourselves build spacecraft smaller, faster, and cheaper." Later, "better" replaced "smaller" in this phrase. One of the results of this strategy is that many smaller (and cheaper) missions are taking the place of a few large (and expensive) missions. However, the capabilities of the communication systems used to operate these missions have not correspondingly increased [1]. While the number of missions is increasing, the capabilities of the instruments on these missions are also increasing. They are providing data at ever increasing rates, far faster than is feasible to send the data to Earth, and the rate of growth of the data from the instruments is also faster than the rate of growth of bandwidth to Earth, so this problem will only get worse in the future. (Even though recent events have cast some doubts on the current implementation of the "better faster, cheaper" strategy and it will probably undergo some changes in the near future, it is very unlikely that NASA will return to an overall strategy of fewer, more expensive missions.)

One obvious answer to this problem is to process the data where it is collected, and to return only the results of the analysis to Earth, rather than the raw data. Similarly, processing data and autonomously making decisions on-board spacecraft will eliminate the problem of latency between Earth and space. Many events that we wish to observe and record are both short-lived and unpredictable, and thus, do not allow an observer on Earth to modify the behavior of a spacecraft as would be needed. For example, a gamma-ray burst may be observed by a spacecraft with a large field-of-view detector, which could then autonomously change orientation to aim a more detailed instrument at the source of the burst. If an astronomer on the ground had to command this action, it is likely the burst, or some large part of it, would be complete before the spacecraft was ready to observe it.

Traditionally, very little data analysis has been done in space, and what has been done has relied on radiation-hardened processors. These architectures are quite old by the time they complete the radiation-hardening process, and do not solve the problems of bandwidth and latency. One solution is an embedded cluster of COTS (Commercial-Off-The-Shelf) processors, where the processors can be selected and placed in the system shortly before mission launch. In as much as COTS processors are not radiation hardened, this requires software that can detect and correct errors caused by the cosmic ray environment found in space. Such a system is being developed by the Jet Propulsion Laboratory under the Remote Exploration and Experimentation (REE) Project [2].

The constraints of the space environment (mass, power, volume, resistance to vibration, shock, thermal cycling and natural space radiation) demand work in packaging that is very different than for most ground-based clusters. Additionally, replacement of faulty components is either impossible or extremely expensive, and uploading new or changing existing software is very difficult from a system design as well as operational perspective. Finally, reliability of the software as well as the hardware/system is a significant concern due to the difficulty in validating computational results, and the potential impact of erroneous behavior with respect to decision making and scientific data analysis. Thus, a

0-7695-0896-0/00 $10.00 © 2000 IEEE

different set of requirements exists for spaceborne computers than for ground-based computers.

2. HARDWARE REQUIREMENTS

Form factor is a critical factor in embedded computing. Thus compute density (GFlops per cubic foot and GFlops per watt) is often as or more important than aggregate processing power. Typically, embedded system vendors are able to achieve roughly a factor of 10 increase in compute density over conventional systems. These gains are achieved by constructing boards containing multiple nodes (typically 2 or 4). Each node consists of a low power processor (e.g., Motorola PowerPC) with a limited amount of memory (e.g. 128 MBytes). In addition, there are no local disks and access to the node is limited to the interconnect, which may be custom or commodity (e.g. Myrinet, though the interconnect form factor may be a problem), and will have been packaged to minimize size and power consumption. These various design tradeoffs allow embedded vendors to fit nearly 100 processing nodes in a volume that can fit underneath a typical office desk.

As mentioned previously, many embedded systems need to withstand much more severe conditions than standard clusters. These systems may be used in the aerospace or military industries, leading to requirements on tolerance to shock, vibration, radiation, thermal conditions, etc. While many of today's commercial components can handle these conditions, they are not packaged to do so, as this increases cost and is not needed by most ordinary users. Thus, for this niche market, different vendors have sprung up to package standard commercial parts with more consideration of these concerns.

There are a variety of vendors that manufacture systems along the above lines. Mercury, CSPI and Sky are three of the more popular systems. Some of the general capabilities are shown below [3]. In addition, for comparison, a tradition cluster system built at JPL is also shown.

TABLE I:
COMPARISON OF COMMERCIAL EMBEDDED SYSTEMS

Vendor	CPU	Interconnect	OS	CPU/ft^3
Mercury	PowerPC	Raceway	MCOS	~10
CSPI	PowerPC	Myrinet	VxWorks	~10
Sky	PowerPC	Sky Channel	SKYmpx	~10
N/A (JPL)	Intel	Ethernet	Linux	~1

In addition to these vendors that specialize in embedded systems, a number of other companies build embedded systems, both parallel and distributed for their customers. These vendors may take systems from the standard vendors listed above and ruggedize and/or repackage them, and they include many US defense contractors (Lockheed, Honeywell, General Dynamics, etc.)

3. THE REMOTE-EXPLORATION AND EXPERIMENTATION PROJECT

Within the NASA High Performance Computing and Communications (HPCC) Program, the Remote Exploration and Experimentation (REE) Project at the Jet Propulsion Laboratory (JPL) intends:

> To bring commercial supercomputing technology into space, in a form which meets the demanding environmental requirements, to enable a new class of science investigation and discovery.

Specifically, the project will:

> Demonstrate a process for rapidly transferring commercial high-performance computing technology into ultra-low power, fault-tolerant architectures for space.
>
> Demonstrate that high-performance onboard processing capability enables a new class of science investigation and highly autonomous remote operation.

The project consists of three initiatives: applications, computing testbeds, and system software. The purpose of the applications initiative is to demonstrate that the unique high-performance low-power computing capability developed by the project enables new science investigation and discovery. In order to do this, five Science Application Teams (SATs) were chosen to develop scalable science applications, and to port these to REE testbeds running REE system software. The applications are meant to be developed on standard ground-based clusters, and then ported to the REE embedded cluster with minimal changes. The needs of the applications also lead to requirements on the system software, and ensure that the hardware and system software meet the needs of the NASA spaceborne applications community. The set of SATs will change over time to increase the project's exposure to NASA missions and to help the project understand the needs of newly proposed missions.

Under the testbed initiative, an initial testbed (a cluster composed of PCs running Linux connected by Fast Ethernet) was built and used for initial applications demonstrations. The next testbed (the first generation embedded scalable computing testbed,) which is designed to operate at least at 30 MOPS/watt[1], is currently being built, and is scheduled to be delivered in September 2000.

[1] MOPS are Millions of OPerations per Second, where operations are both floating point and integer. For a spaceborne embedded computer, performance per unit power is a key metric.

This testbed consists of 40 commodity off-the-shelf (COTS) processors connected by a COTS network fabric. Through future RFPs (Requests For Proposals), the project will obtain additional testbeds that perform faster while using less power (at least 300 MOPS/watt). Criteria that are required of the testbeds are: consistency with rapid (18 month or less) transfer of new Earth-based technologies to space, no single point of failure, and graceful degradation in the event of hardware failure.

The purpose of the system software initiative is to provide the services required to let the applications make as full a use as possible of the hardware while assuring reliable operation in space and providing an easy-to-use development environment. Much like the hardware, the system software is intended to use commercial components as much as possible. The major challenge for the system software is to develop a middleware layer between the operating system and the applications that accepts that both permanent and transient faults will occur and provides for recovery from them.

4. FAULT-TOLERANCE REQUIREMENTS

The spaceborne embedded cluster described in this paper is being designed with certain characteristics that impact fault-tolerance requirements [4]. They are:

[1] The REE system is not intended for use in high radiation environments such as the Van Allen Belts or the Jovian System. This is key to the ability to use non-radiation hardened components, and thus gain a two to three generation advantage over available radiation hardened flight computers.

2. The REE system is being designed primarily for the processing of science data, rather than hard (vs. soft) real time, mission critical, spacecraft control functions. Thus, occasional resets, processing delays, and possibly even dropped frames[2] or other service interruptions are potentially acceptable. The advantage here is that the use of non-replicated fault tolerance techniques with concomitant advantages in power/performance is permitted.
3. The system is intended, with appropriate replication techniques, such as software implemented triple-modular-redundancy, to be capable of performing a limited range of high-reliability, operate-through, real time tasks. This will be, at least initially, in a segregated portion of the system which will operate in a relatively poor power/performance mode (providing only a 2.5 to 3.0 power/performance improvement over available radiation hardened computers vs the expected 10X improvement in the rest of the system). This segregation of real time activities will allow the system to perform these types of tasks if necessary, but with resultant penalties. In the future, the possibility of performing real time tasks in a minimally- or non-replicated and non-segregated mode will be investigated.

The science applications are generally MPI programs that are not replicated and therefore can take full advantage of the computing power of the hardware. (However, we also will support Triple or Quad Modular Redundancy (TMR/QMR) in software for smaller applications that require high reliability, as opposed to high availability.) As the processors are COTS components, they are not radiation-hardened, and will suffer from faults. The primary concerns for the REE environments, i.e., Low Earth Orbit (LEO), Geosynchronous Earth Orbit (GEO), and Deep Space (DS), are transient errors induced by natural Galactic Cosmic Rays (GCR's) and energetic protons. The principal faults are single bit flips (also known as single event upsets or SEUs) in memory and registers on the CPU. (Note: memory off the CPU will be error-detecting and correcting – EDAC.) Current understanding and modeling indicate that the REE first generation testbed would see approximately 1 single bit fault per CPU-hour in either the GEO or DS environments, and approximately 7 single bit faults per CPU-hour in the LEO environment [4].

These error rates require that the applications and system software be self-checking, or tolerant of errors. The REE Project's goal is to minimize the changes that the developer of the application has to make, and to make the application as portable as possible between various parallel systems, whether these are standard clusters, embedded clusters, or massively parallel processors. In order to do this, much of fault-tolerance must be pushed onto the system software or middleware, either hidden from the application or in the form of tools that the application can choose to use.

One example of this is research in Algorithm-Based Fault Tolerance (ABFT) techniques, and development of ABFT libraries for linear algebra and Fourier analysis tasks shared by the applications [5]. As an example, the fine optical control application from NGST (Next Generation Space Telescope, see section V) consists of three parts: phase retrieval (Misell algorithm), phase unwrapping, and actuator fitting. Approximately 70% of the CPU time spent in the phase retrieval code is used to perform FFTs. An ABFT wrapper for the version of the distributed FFT that is being used (FFTW [6]) has been written that will allow the application to determine if this routine completed correctly, or if an SEU occurred during the calculation, in which case the FFT can be repeated. There are two versions of the ABFT library, "naïve" and "expert". The "naïve" library is a simple library replacement that is not seen by the application. The calls to the library are identical, and the only changes that must be made are an additional include file and a new library in the link command. With this version, the application will

[2] "Frame" is used in this paper to mean an iteration of an application's main loop of the form: input a set of data, process the data, and output the processed data. A dropped frame is thus an incomplete iteration of the loop, where no processed data is output for given set of input data.

attempt the FFT a preset number of times, each time checking for correctness within a preset tolerance, and if each fails, then abort to a higher level of middleware control. The "expert" version allows the application more control over the results. The tolerance used for determination of correctness and the behavior upon failure can be changed, perhaps allowing the application to use a partially correct result and move on.

An example of the required middleware is an application manager. REE is currently working with Chameleon [7], which can be thought of as an application manager. It starts the application, performs checks of its progress, and restarts the application if needed. For the REE science applications, which are generally frame-based, restarting the application means that the frame that was being processed when the application crashed is the starting point when the application is restarted. Other applications may need automated check-pointing schemes. Of course, Chameleon itself must also be fault-tolerant. This is an example of the classical paradox: sed quis custodiet ipsos custodes? (Who will watch the watchers? [8])

Because the current MPI standard (1.2) does not allow dynamic MPI processes, when any one processor fails the complete job must be restarted. While this is reasonable for jobs where all the processors are working together on solving one large problem, some of the REE applications are in a different category, where fairly large amounts of work may be assigned by a master to some number of slaves, with the master integrating the results. In this case, the failure of any one slave only should mean that the work being done by that slave should be reassigned to another slave, as long as the work units are fairly independent. REE will likely build or adapt an application manager (e.g.; MW [9]) for this paradigm, as well.

Additionally, the system must deal with two types of hardware failures: complete node failure, long-lasting and/or permanent faults which cause a node to be unusable; and partial node failure, long-lasting and/or permanent faults which only effect part of a node (i.e.; one memory bank) so that the node is degraded. In both cases, the system software may notice a large number of errors on a given processor, and decide to take that processor out of the active cluster for testing. Any software that was running on the processor must be migrated to another processor, if possible by transparently moving it, or else by stopping it and restarting it on the new set of processors. A new processor may be available from the set of cluster resources that are currently on-line, or it may have to be brought on-line, or for reasons of power or limited resources, no new processors may be available. These requirements imply system management with control of hardware similar to those of a traditional cluster's system administrator and the ability to make tradeoffs between application, power, and mission requirements, which can be particularly difficult when partial node failure exists. A software-implemented system administrator (SISA) that will be developed by the REE Project will perform this management function.

5. APPLICATIONS

The first round of Science Application Teams consists of the following five teams:

Gamma-ray Large Area Space Telescope (GLAST): This team, led by Prof. Peter Michelson (Stanford) and Prof. Toby Burnett (U. of Washington) will examine detection of gamma rays in a sea of background cosmic rays (about 1 in 10,000 events will be a gamma ray), and reconstruction of the gamma-ray trajectory.

Mars Rover Science: Dr. R. Steven Saunders (JPL) leads this team, which has two applications. First, texture analysis and image segmentation are used to identify various materials on Mars for further scientific analysis. Second, images obtained from a stereo camera are analyzed for use in autonomous navigation.

Next Generation Space Telescope (NGST): Led by Dr. John Mather (Goddard Space Flight Center – GSFC), this team also has two applications. The first is to perform multiple fast reads of the charge coupled devices (CCDs) which take the telescope images in order to eliminate or reduce the effect of cosmic rays which hit these CCDs during an exposure. The second is to perform fine optical control by using a wavefront sensing algorithm to control a deformable mirror.

Orbiting Thermal Imaging Spectrometer (OTIS): This team is led by Prof. Alan Gillespie (U. of Washington). They are designing an application to take hyperspectral imaging data and retrieve temperature and emissivity, as well as performing spectral matching and unmixing, then image classification.

Solar Terrestrial Probe Project (STP): This team, led by Dr. Steven Curtis (GSFC), is examining using fleets of spacecraft for two applications: radio astronomical imaging and plasma moment analysis.

These applications take advantage of large amounts of computing, as well as performance/power ratios that are at least an order of magnitude above those available in today's spacecraft. They are attempting to implement and test new approaches to science data processing and autonomy. They have all delivered parallel code to the REE project, and currently 7 of the 9 codes have been successfully run on an embedded cluster.

The initial applications development aims at running the applications on the testbed without fault-detection or fault-recovery. Once the applications run successfully,

these topics will be addressed through the development of an applications programming guide. This will define an interface between the application and the middleware, including progress messages, error reporting, application-specified checkpointing, ABFT calls, and other tools that will be developed. This is intended to be a living document, because the REE Project plans to iteratively test applications using both random and focussed fault-injection, and to use the results of these experiments to modify or add to the set of tools available to the application. We are eager to collaborate with others in examining, developing and testing these tools. The overall iterative process should drive down the number of undetected and therefore uncorrected faults to a sufficiently low number for the environment in which each mission will operate to satisfy the mission scientist and to be similar in scale to other errors that are commonly accepted, including instrument noise, and transmission errors.

6. PRELIMINARY CONCLUSIONS

Understanding application behavior in an environment where faults may occur at any time and any place is challenging. This was made obvious at a recent demonstration of the REE testbed system. At one point in the demonstration, after a fault was injected, the program became stuck in an infinite loop. When this happens, the application manager software (Chameleon) is supposed to detect a lack of progress, and restart the application. Unfortunately, a call to the application manager was inside the loop that became infinite, reporting (falsely, in this case) that progress was being made. These calls were made at regular intervals, and fooled the application manager into believing that they were legitimate. This incident and others has led to changes in the initial thinking about what programming guidelines work best in such an environment.

Two possible solutions to the above problem were proposed. One such proposal was that the application use multiple progress indicators. Thus the application would issue a progress call to indicator n just before entering the loop. Within the loop, only progress calls to indicator m would be issued. If the above problem were to occur in this case, after a predefined interval, the application manager would have a timeout on progress indicator n, and would therefore restart the application. An alternative solution would be for the program not to issue progress calls unless it had independently verified that progress was actually being made.

Any programming guidelines should of course suggest following good programming practices such as always checking for errors. In addition, because the program will operate in an "anything can happen" environment, it can not safely assume that its current state is consistent. For example, even though a program design may call for two variables to have values that are always correlated, an SEU may change this. One of the unresolved issues is how to handle this kind of situation while still minimizing the impact on the application programmer of operating in an environment with faults. As the REE project's experience in this area increases, it is likely that many other interesting issues will also arise.

The REE project is still fairly young, but our experience to this point looks promising for the future of the project. We have a number of problems to solve, but none seem intractable, and we are convinced that our model of moderately fault-tolerant applications on embedded COTS clusters in space will play a major role in many future NASA missions.

7. ACKNOWLEDGEMENTS

This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. The authors also acknowledge helpful conversations with Jeremy Kepner (MIT Lincoln Laboratory) and Raphael Some (JPL).

8. REFERENCES

[1] W. Schober, F. Lansing, K. Wilson, E. Webb, "High Data Rate Instrument Study," JPL Publication 99-4, JPL, Pasadena, California, 1999.

[2] Remote Exploration and Experimentation Project Plan, March 1999, http://ree.jpl.nasa.gov/.

[3] D. S. Katz and J. Kepner, Embedded Clusters, in A Whitepaper on Cluster Computing, submitted to *International Journal of High-Performance Applications and Supercomputing*, April 2000.

[4] J. Beahan, L. Edmonds, R. D. Ferraro, A. Johnston, D. S. Katz, and R. R. Some, "Detailed Radiation Fault Modeling of the Remote Exploration and Experimentation (REE) First Generation Testbed Architecture," Proceedings of IEEE Aerospace Conference, IEEE, 2000.

[5] M. Turmon and R. Granat, "Algorithm-Based Fault Tolerance for Spaceborne Computing: Basis and Implementations," Proceedings of IEEE Aerospace Conference, IEEE, 2000.

[6] Fastest Fourier Transform in the West, http://www.fftw.org/.

[7] Z. Kalbarczyk, S. Bagchi, K. Whisnant, R. Iyer, "Chameleon: A Software Infrastructure for Adaptive Fault Tolerance," *IEEE Transactions on Parallel and Distributed Computing*, pp. 560-579, June 1999.

[8] Juvenal, Satura VI, circa 115 A. D.

[9] J.-P. Goux, S. Kulkarni, J. Linderoth, and M. Yoder, "An Enabling Framework for Master-Worker Applications on the Computational Grid," Proceedings of the Ninth IEEE International Symposium on High Performance Distributed Computing, pp. 43-50, IEEE Computer Society, 2000.