High-Performance, Dependable Multiprocessor

Jeremy Ramos , John Samson, David Lupia
Honeywell Aerospace, Defense and Space
13350 US Highway 19 North
Clearwater, Florida 33764-7290
jeremy.ramos@honeywell.com
john.r.samson@honywell.com

Ian Troxel, Rajagopal Subramaniyan, Adam Jacobs,
James Greco, Grzegorz Cieslewski, John Curreri,
Michael Fischer, Eric Grobelny, Alan George
HCS Research Lab, University of Florida
Gainesville, Florida 32611-6200
george@hcs.ufl.edu

Vikas Aggarwal, Minesh Patel
Tandel Systems
mpatel@tandelsys.com

Raphael Some
NASA Jet Propulsion Lab
rsome@jpl.nasa.gov

Abstract—[1][2][3] With the ever-increasing demand for higher bandwidth and processing capacity of today's space exploration, space science, and defense missions, the ability to efficiently apply commercial-off-the-shelf (COTS) processors for on-board computing is now a critical need. In response to this need, NASA's New Millennium Program office has commissioned the development of Dependable Multiprocessor (DM) technology for use in payload and robotic missions. The Dependable Multiprocessor technology is a COTS-based, power-efficient, high-performance, highly dependable, fault-tolerant cluster computer. To date, Honeywell has successfully demonstrated a TRL4 prototype of the Dependable Multiprocessor [1], and is now working on the development of a TRL5 prototype. For the present effort Honeywell has teamed up with the University of Florida via its High-performance Computing and Simulation (HCS) Research Laboratory, and together the team has demonstrated major elements of the Dependable Multiprocessor TRL5 system. This paper provides a detailed description of the basic Dependable Multiprocessor technology, and the TRL5 technology prototype currently under development.

TABLE OF CONTENTS

1. Introduction

NASA has a long and relatively productive history of space exploration as exemplified by recent rover missions to Mars. Traditionally, space exploration missions have essentially been remote-control platforms with all major decisions made by operators located in control centers on Earth. The onboard computers in these remote systems have contained minimal functionality, partially in order to satisfy design size and power constraints, but also to minimize complexity as a means of coping with high-dependability requirements. Hence, these traditional space computers have been capable of doing little more than executing small sets of real-time spacecraft control procedures, with little or no processing bandwidth left over for instrument data processing. This approach has worked reasonably well until now, as instruments have consisted of low-complexity imagers, with compressible output streams transmittable to ground stations for post-processing knowledge extraction. As the capabilities of instruments on exploration platforms increase, more processing and autonomy will be necessary onboard to fully exploit their vast output data streams [2]. Autonomous spacecraft will further increase knowledge returns through opportunistic explorations conducted outside the Earth-bound operator control loop. In response, NASA has initiated several projects to develop technologies that address the onboard processing gap. One such program is the NASA New Millennium Program's ST8 Project [3].

The vision of the New Millennium Program's Dependable Multiprocessor experiment is to migrate COTS-based computers to space, thereby enabling new classes of science [4]. In support of this vision, the Honeywell and University of Florida team is developing the Dependable Multiprocessor (DM) technology. This technology combines a set of innovative solutions to enable efficient use of high-performance COTS processors in the harsh space environment, while maintaining the required system reliability and availability. Dependable Multiprocessor is a sophisticated technology composed of four chief components [5].

First, Dependable Multiprocessor is an architecture and methodology enabling the use, in space, of COTS-based, high-performance, scalable, multi-computer systems. A distinguishing feature of the architecture, critical to achieving high performance and efficiency, is the use of reconfigurable FPGA co-processors. Furthermore, through

[1] 0-7803-9546-8/06/$20.00© 2006 IEEE; This paper has not been published elsewhere and is offered for exclusive publication except that Honeywell reserves the right to reproduce the material in whole or in part for its own use and, where Honeywell is obligated by contract.

[2] Paper 1511

[3] The project formerly was known as the Environmentally-Adaptive Fault-Tolerant Computing (EAFTC) project.

Reprinted from the *Proceedings of the 2006 IEEE Aerospace Conference*, Big Sky, MN, March 6, 2006

accommodation for upgrades to future COTS parts, the Dependable Multiprocessor architecture can evolve along side commercial technologies thereby ensuring its longevity.

Second, Dependable Multiprocessor is a parallel processing environment for science codes that incorporates an application development and runtime environment familiar to science application developers. By adopting these standard environments, the Dependable Multiprocessor can significantly reduce the cost and schedule associated with porting of applications from the laboratory to the spacecraft payload data processor.

Third, Dependable Multiprocessor is a set of algorithms for system and fault tolerance management. These algorithms allow systems to dynamically manage resources in response to environment, application criticality, and system mode, in order to maintain mission required dependability and maximal system efficiency.

Lastly, Dependable Multiprocessor is a methodology and associated tools that allow developers of Dependable Multiprocessor systems to predict their implementation's behavior in the target environment, including: predictions of availability, dependability, fault rates/types, and system-level performance.

2. RELATED WORK

Dependable Multiprocessor builds on earlier projects at JPL, Honeywell and Raytheon, which were sponsored by NASA, DARPA, and USAF.

The Advanced Onboard Signal Processor (AOSP), developed by Raytheon Corporation, for the USAF in the late 70s and mid 80s made significant breakthroughs in understanding the effects of natural and man-made radiation on computing systems and components and in developing architectural, hardware, and software techniques for detection, isolation, and mitigation of these effects. AOSP, though never flown, was instrumental in developing the fundamental concepts, modeling, and testing techniques behind much of the current work in fault-tolerant, high-performance distributed computing.

Advanced Architecture Onboard Processor (AAOP), a follow-on effort to AOSP, also developed at Raytheon Corporation, engineered alternative concepts and new approaches to spacecraft onboard data processing. The AAOP architecture found its way into both commercial and military platforms, but was never commercialized or popularized as it was, in large measure, overkill for most applications.

The DARPA-sponsored Space Touchstone computer, developed at Honeywell, was ground-breaking in its goal of using COTS components and a COTS system architecture in high-performance, highly reliable, airborne and spaceborne computing.

NASA's Remote Exploration and Experimentation (REE) project [6], at JPL, extended fault-tolerant computing to the world of parallel and cluster processing. Among other advances, REE addressed, in a general manner, the issue of low cost and tailored fault tolerance. The REE project developed fault-tolerant middleware for cluster computers, methods and tools for test and characterization of components and systems, and Software-Implemented Fault Tolerance (SIFT) techniques and libraries. The project led to fundamental concepts upon which to develop fault-tolerant, high-performance parallel processing and, more specifically, fault-tolerant, low-cost, high-performance, power-ratio, embedded clusters.

3. OVERALL SYSTEM ARCHITECTURE

Figure 1 depicts the Dependable Multiprocessor hardware architecture, which is based upon Honeywell's Integrated Payload concept [7]. The Dependable Multiprocessor is essentially a reconfigurable cluster computer with centralized control. The essential hardware elements of the system are a redundant radiation-hardened System Controller, a cluster of COTS-based reconfigurable Data Processors, redundant COTS-based Packet Switched networks, and a radiation-hardened Mass Data Store. Additional peripherals or custom modules may be added to the network to extend the system's capability; however, these peripherals are outside of the scope of the base architecture. To increase system reliability it is possible to employ redundancy of the System Controller and network as depicted in the block diagram. Likewise, N-of-M sparing of Data Processors may be used for added reliability. Redundancy, however, may not be affordable or necessary for all missions, and therefore it is not a required architectural element. Command and Telemetry is exchanged directly between the active System Controller and the Spacecraft Control Computer via direct 1553 spacecraft interfaces. The primary dataflow in the system is from instrument to Mass Data Store, through the cluster, back to the Mass Data Store, and finally to the ground via the spacecraft's Communication Subsystem.

The primary mechanism for hardware scalability provided by the architecture is the number of Data Processors inserted into the network. First adopters are expected to need up to 30 nodes in their clusters, a node count that is well within the capabilities of Gigabit Ethernet for selected applications. Alternative approaches to scalability include forming a cluster-of-clusters. This alternative may be more suitable for eventual product development, since standard cluster configurations can be developed as fully integrated products, and later combined to form a larger machine as required by a particular mission.

Reprinted from the *Proceedings of the 2006 IEEE Aerospace Conference*, Big Sky, MN, March 6, 2006

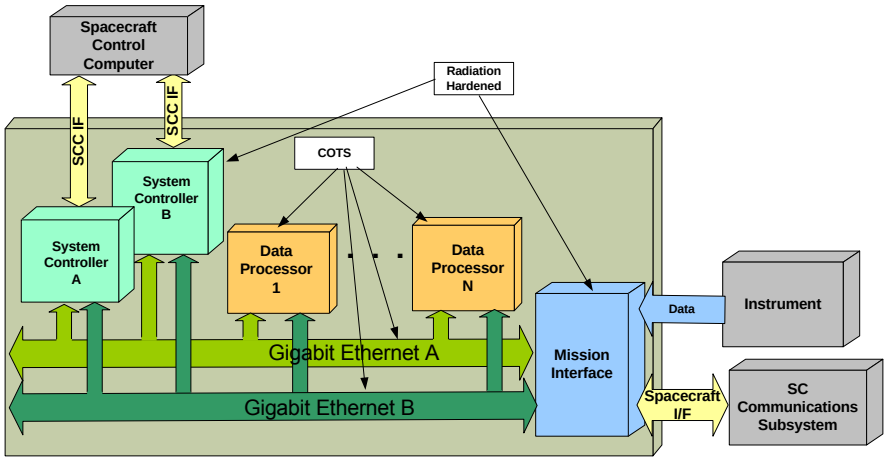

Figure 1. System Hardware Architecture of the Dependable Multiprocessor.

3.1. System Controller

All central control software for the cluster executes on the System Controller node. Due to the critical nature of centralized control we have selected the Honeywell Radiation-Hardened PPC (RHPPC) Single-Board Computer (SBC) for implementation of the System Controller. By implementing the System Controller in highly reliable and radiation-hardened electronics, we reduce the likelihood of experiencing major system control faults due to single-event upsets (SEUs). The RHPCC SBC is based upon the Motorola PowerPC 603e microprocessor technology; its key features are summarized in Table 1 [8].

3.2. Data Processors and FPGAs

The core processing elements of the cluster are the Data Processors. As depicted in Figure 2, the Data Processor's architecture is similar to a standard SBC, with the exception of the FPGA co-processing element. In support of our COTS goal, the Data Processor employs a COTS IBM PowerPC 750FX microprocessor [9], a Xilinx VirtexII 6000 FPGA co-processor [10], and their associated standard support chips (e.g., COTS bridge, and I/O chips, clocks, and memories).

The reconfigurable FPGA co-processor is a key to achieving high-performance and efficiency in the cluster. The FPGA provides a capability for implementing algorithms directly in hardware, thereby exploiting algorithmic parallelism. This approach typically results in speedup of 10-to-100x with significant reductions in power [11]. Additionally, FPGAs make the cluster a highly flexible platform, allowing on-demand configuration of hardware. Via FPGA reconfiguration, the Data Processor can support a variety of application-specific modules such as digital signal processing (DSP) cores, data compression, and vector processors. This overall flexibility allows application designers to adapt the cluster hardware for a variety of mission-level requirements. For DSP and other algorithm-intensive applications, greater efficiency and performance may be achieved by using custom hardware modules in the FPGAs. Then again, for applications that are logic-intensive, microprocessors are more suitable targets. Some key features of the Data Processor are listed in Table 2.

Table 1. Key Features of the RHPPC SBC

3.3V and 5.0 V Power
RHPPC delivering 100 MIPS
Peripheral Enhancement Component support chip
4MB EEPROM with Single Error Correction and Double Error Detection
512KB EEPROM
128 MB DRAM with SuperEDAC
6U x 220mm Euro Card Form Factor
Max Power Draw 15W
Mass >3lbs
Redundant 1553 (interface to spacecraft computer)
32-bit 33MHz PCI (interface to cluster and MIB electronics)

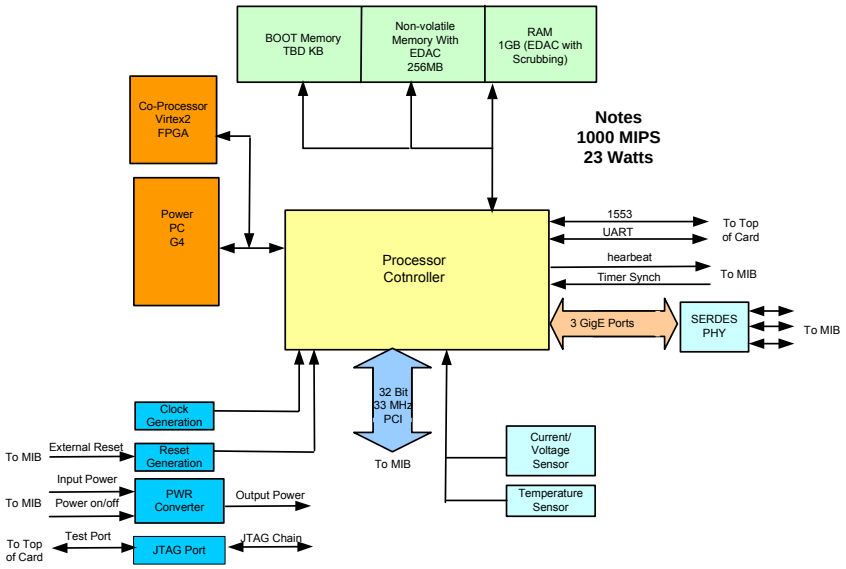

Figure 2. Hardware Architecture of the Data Processor.

Table 2. Key Features of the Data Processor

COTS Based
750 fx @ 650 MHz Delivering 1300 MIPS
VirtexII 6000 FPGA co-processor
PCI 32-bit 33 MHz
Gigabit Ethernet
1 GB DRAM with ECC
12MB EEPROM with SECDED EDAC
256 MB Flash
JTAG test interface
UART interface for development
6U x 220 mm Euro Card Form Factor
Mass <3 lbs
Max Power Draw 20W

3.3. Network Interconnect

Gigabit Ethernet (GigE) [12] is the prevalent networking system for cluster architectures. GigE is a low-cost packet switched network that offers bandwidths up to 1 Gb/s. Additionally, GigE has a promising growth path to 10 GigE, a new standard that will support bandwidths up to 10 Gb/s. GigE offers many network topology options allowing system-level architectural optimization. Furthermore, many COTS microprocessors and peripherals include GigE network interfaces, allowing for direct connection to a GigE network without additional hardware.

In the Dependable Multiprocessor, GigE is the data exchange medium. Sideband, low-bandwidth, low-latency buses can be used for control. This method allows for optimization of the GigE network to address the high throughput needs of the parallel-processing science applications.

3.4. Mission Interface

The last hardware element of the system that we will discuss herein is the Mission Interface. In general terms, the Mission Interface is the clusters port to the instruments and communication system. Its functions include providing the primary cluster data input/output interface, and isolating the cluster from other spacecraft subsystems. As with the System Controller, the Mission Interface should be radiation-hardened. This approach minimizes the likelihood that data will be corrupted by faults, ensuring reliable input and output of data from the system. By making the Mission Interface an independent cluster component we reduce the impact of porting the cluster computer to new instruments, communication systems, and missions.

3.5. TRL5 Testbed Architecture

The focus of the current project phase is the development of a TRL5 system prototype, including hardware and software. As depicted in Figure 3, the TRL5 prototype hardware consists of a cluster computer, a development workstation, reset controller, and power supply.

The cluster computer is implemented using seven Orion Technologies Inc CPC7510 SBCs in a CompactPCI chassis interconnected over redundant Gigabit Ethernet switches [13]. Four of the SBCs are configured as Data Processors, two as redundant System Controllers, and one as a Mass Data Store.

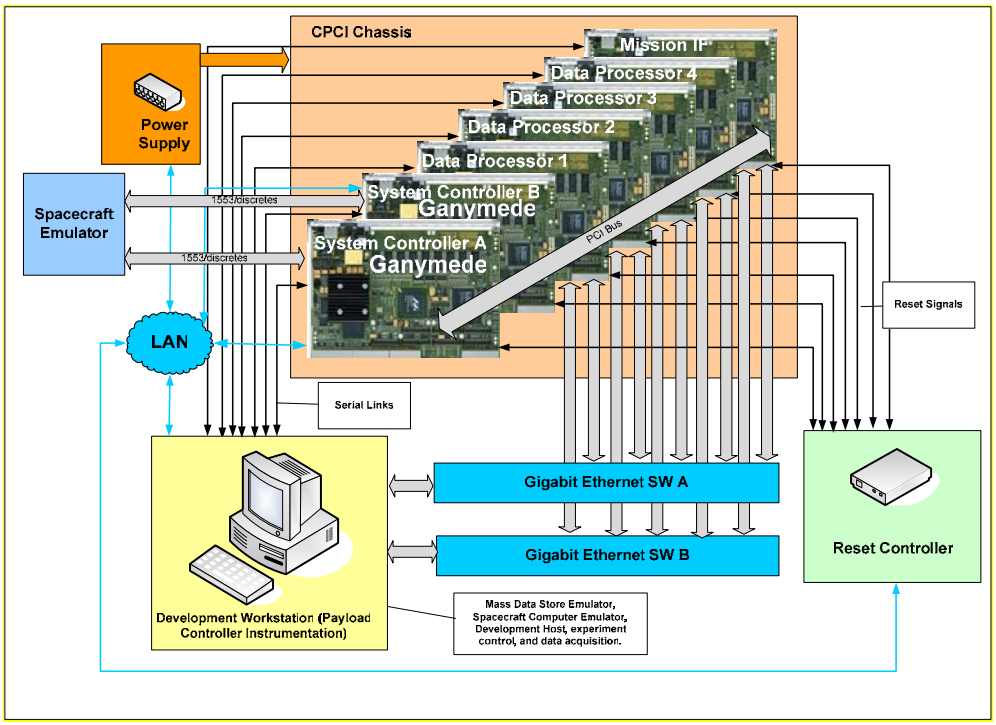

Figure 3. System Configuration of the TRL5 Prototype Testbed.

The CPC7510 is a hot swappable, high-performance, IBM 750fx PowerPC SBC designed for high-availability applications. The CPC7510 is extremely versatile with two PMC slots, variable operating frequencies between 650 MHz and 1 GHz, and support for chassis controller or peripheral slot placement. Other key features of the CPC7510 are summarized in Table 3.

As a System Controller in our testbed, the CPC7510 has been outfitted, via the PMC, with a third Ethernet Network Interface for experimental control interfacing from the development workstation. As a Data Processor, the standard CPC7510 includes an ADM-XRC-II PMC. The ADM-XRC-II is a high-performance FPGA co-processing PMC from Alpha Data Parallel Systems [14], and is representative of the flight FPGA co-processor. The final configuration of the CPC7510 includes a hard drive PMC to emulate the storage capacity of a Mass Data Store.

Table 3. Key Features of the CPC7510

Orion Technologies (OTI) Linux Kernel
PowerPC 750FX v2.3 @ 600MHz
64/32 bit Sentinel64 universal-mode PCI-to-PCI bridge
Marvell Discovery II (MV64360) system controller 133MHz front side host interface
128 MB high-speed DDR SDRAM
Dual Gigabit Ethernet interfaces
Dual PMC slots (64-bit 133MHz, 32-bit 66MHz)
Dual serial RS-232 interfaces
6U x 220 mm Euro Card Form Factor

Software development, experiment control, instrumentation data collection, and Spacecraft Control Computer emulation are achieved with the development workstation. The development workstation is a standard Dell PC configured with a 3 GHz Xeon CPU running the Fedora Core 4 Linux OS. The basic configuration has been altered to include support for 8 serial ports using an Axxon Serial Port Mux 8 I/O board, and expanded the network interface count from one to three GigE ports with an Intel Dual Gigabit Ethernet NIC.

Additional elements of the testbed include a software-controlled, instrumented power supply, which is used to take detail measurements of power usage, and a reset control device, integrated by Tandel Systems, which provides the ability for the software on either the development workstation or the active System Controller to reset each node in the system individually.

4. Middleware Architecture

A top-level overview of the Dependable Multiprocessor software architecture is illustrated in Figure 4. The system is composed of three primary layers: mission layer, middleware layer, and platform layer. A key feature of this architecture is the integration of generic software fault-tolerant techniques implemented in the middleware framework. The Dependable Multiprocessor framework is independent of and transparent to the specific mission application, and independent of and transparent to the

Reprinted from the *Proceedings of the 2006 IEEE Aerospace Conference*, Big Sky, MN, March 6, 2006

underlying platform. This transparency is achieved at the interface to the mission layer through well-defined, high-level, Application Programming Interfaces (APIs), and at the platform layer through a System Abstraction Layer (SAL) which isolates the middleware from the underlying platform. This isolation and encapsulation makes the middleware services available to future mission applications by facilitating its porting to new platforms.

The lowest layer of the system is the platform layer, which includes a COTS operating system, hardware-specific software such as network drivers, and the hardware elements. The basic platform software is implemented using MontaVista's version of the Linux Operating System, and is common to all of the processors in the cluster. Other Operating Systems may also be used, but Linux facilitates leverage of many existing software tools. The central component of the system is the middleware layer, which contains the essential Dependable Multiprocessor system services that provide the fault tolerance, job management, and other applications services detailed in the following sections.

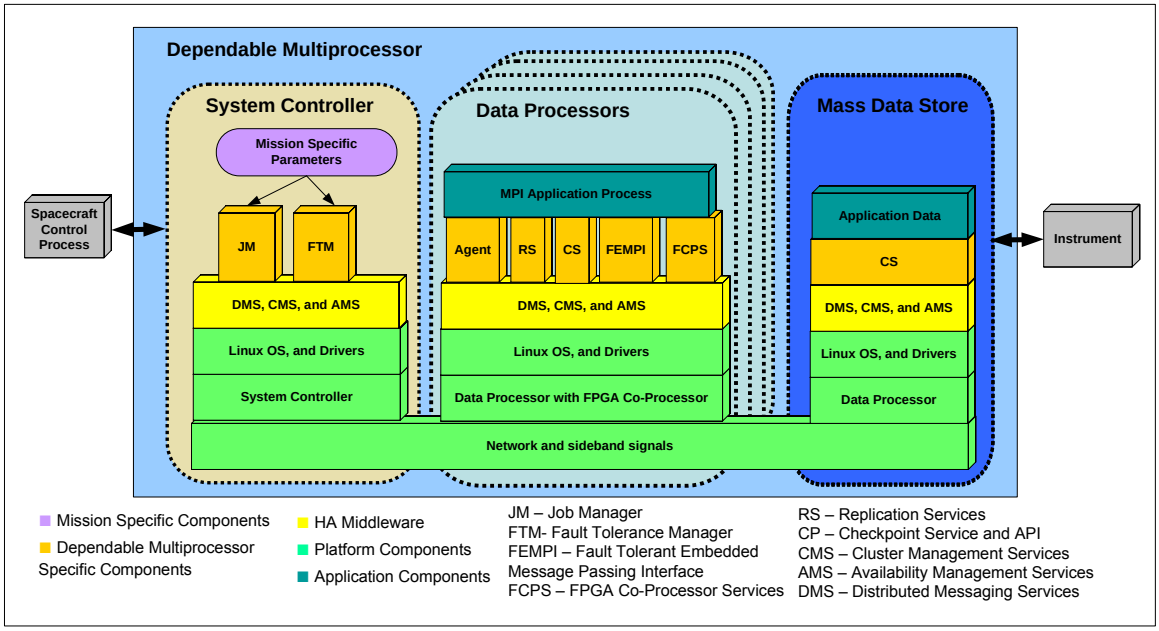

Figure 4. System Software Architecture of the Dependable Multiprocessor.

4.1. High-Availability Middleware

The High-Availability (HA) Middleware, the foundation component for the Dependable Multiprocessor Middleware, is composed of numerous services. For Dependable Multiprocessor, we focus on Availability Management, Distributed Messaging, and Cluster Management. In the Dependable Multiprocessor design, the functionality of these basic elements is extended and augmented by system-specific components to be covered in subsequent sections of this paper. The primary functions of the HA Middleware are resource monitoring, fault detection, fault diagnosis, fault recovery, fault reporting, cluster configuration, event logging, and distributed messaging. High Availability is based on a small, reliable, cross-platform kernel that provides the foundation for all standard services, and its extensions. The kernel also provides a portability layer limiting user dependencies on the underlying operating system and hardware.

Availability Management Service (AMS) provides the core availability management framework and is hosted on the cluster computer's System Controller. Managed AMS resources can include applications, operating system, chassis, I/O cards, redundant CPUs, networks, peripherals, clusters, and other middleware. These system resources and their relationships are abstractly represented in AMS and shared with the Fault-Tolerance Manager, which in turn uses it to assess the system's health.

The Distributed Messaging Service (DMS) is a vital service offered by the HA Middleware. Its function is to provide a reliable messaging layer for communications in the Dependable Multiprocessor cluster. Distributed messaging is designed to address the need for intra- and inter-process communications between system elements for numerous application needs such as checkpointing, client/server communications, event notification, fault management, and time-critical communications. The messaging service provides an effective and uniform way for distributed messaging components to efficiently communicate and coordinate their activities.

Reprinted from the *Proceedings of the 2006 IEEE Aerospace Conference*, Big Sky, MN, March 6, 2006

Communication using DMS begins when an application opens a DMS connection creating a path between interested subscribers to the data. When an application opens a connection, it specifies a desired channel allowing DMS to segment connections into smaller logical networks. The application can then transmit a message to the registered subscribers on that channel. Instead of managing communications within a network at the lower socket and address level that requires the developer to build headers, DMS enables application developers to group similar information together into logical classifications. This approach is unlike sockets for which APIs must be provided with the exact IP addresses and ports for all communicating machines. By contrast, by classifying messages into families and types, DMS can route data to intended destinations without having to explicitly address each message. Machines "register" to receive messages of specific families and types, and on specific channels, so the sending machine does not need to know the destination. This architecture also facilitates the implementation of network failover that is transparent to the application. DMS identifies, classifies, and manages the addresses in order to streamline message delivery. The message publisher can select between two types of connections, standard or direct. These connections can be to another application, an extension, or a server pool.

The Cluster Management Service (CMS) interacts with, and is dependent upon, other HA Middleware services. CMS manages the physical nodes or instances of HA Middleware, while AMS manages the logical representation of these and other resources in the availability system model. CMS is responsible for discovering, incorporating, and monitoring the nodes within the cluster along with their associated network interfaces. The addition or failure of nodes and their network interfaces is communicated to AMS, and the FT Manager through the DMS. CMS also works in conjunction with AMS to provide manager node redundancy, thus eliminating the manager node as a possible single point of failure.

HA Middleware provides some additional minor services such as database management, logging services, and tracing. The in-memory management database is a high-performance, distributed, replicated database for configuration, data storage, and retrieval. The database supports distributed architectures and offers portable and extensible database architecture. It includes facilities such as table creation, row insertion, reading and deleting, and search with indexed retrieval. The HA Middleware Logging Services are used to capture the activity of the system for later download. Logs are used to help perform fault analysis and root cause determination. Any service and application code can use the Logging Services, which provide a variety of features, including multiple and fixed-size circular (automatic overwrite) logs. Developers use the trace facility primarily during the engineering process as well as for capturing system behavior during operation. It sends output to a file or other output device.

Extension components have been developed allow the Fault-Tolerance Manager (FTM) component of the Dependable Multiprocessor, described in Section 4.3, to interface with HA Middleware. In particular, the extensions allow the FTM to detect when a service or application (including the HA Middleware itself), has initialized correctly or failed. Also, one of the HA Middleware extensions is the mechanism by which the FTM starts other middleware services in a fault-tolerant manner.

4.2. Control Process

The Control Process (CP) provides a unified view of the embedded cluster to the spacecraft control computer and the ground-based station user. It directly communicates to an independent process running on the active system controller via a communication link to the embedded cluster. This process, residing on the system controller, translates the commands from the CP into DMS messages that can be interpreted by the other Dependable Multiprocessor system components, and relays the status and other information from the embedded cluster to the CP. The CP monitors the system health via a system-wide heartbeat, generated by the FTM as described in Section 4.3. This heartbeat is employed by the CP to detect system-level failures, to which the CP responds by performing required diagnostics and failing over to the standby system controller after a system-wide reboot. In addition to monitoring system status, the CP also presents a mechanism to remotely initiate and monitor diagnostic features provided by the Dependable Multiprocessor middleware.

4.3 Fault-Tolerance Manger and Agents

The Fault-Tolerance Manager (FTM) is the central fault recovery function for the Dependable Multiprocessor system. The FTM works closely with the HA Middleware's AMS to detect and recover from system and application faults. Each resource in the system is abstracted in AMS service. If a resource's health state transitions, the FTM is updated, thus triggering an appropriate recovery action. At runtime, the FTM refers to a set of recovery policies from soft reboot to power off for various system and application failures. For application recovery, the user can define a number of recovery modes based on runtime conditions. This configurability is particularly important when executing parallel applications with FEMPI (discussed in further detail in Section 4.5). The job manager frequently directs the recovery policies in the case of application failures. Additional information is provided on the interactions between the FTM and the Job Manager in the next section.

In addition to the HA middleware, the central FTM relies on distributed software agents to gather system and application liveliness information. The distributed nature of the agents ensures that the central FTM does not become a monitoring bottleneck, especially since the FTM and other central Dependable Multiprocessor software core components

Reprinted from the *Proceedings of the 2006 IEEE Aerospace Conference*, Big Sky, MN, March 6, 2006

execute on a relatively low-performance, radiation-hardened processor. Numerous mechanisms are in place to ensure the integrity of remote agents running on non-radiation protected data processors as described in Section 4.1. In addition to implementing recovery policies, the FTM also maintains a fault history of various metrics for use in the diagnosis and recovery process. This information is also used to make decisions about system configuration and application scheduling, and thus, to ensure maximum availability. Also, the FTM is the central software component through which the embedded system sends heartbeats to the spacecraft.

4.4. Job Manager and Agents

The Job Manager (JM) primarily functions to provide application scheduling, resource allocation, process dispatching, and directing application recovery based upon user-defined policies. The JM employs an opportunistic load balancing scheduler, which receives frequent system status updates from the FTM in order to maximize system availability. Jobs are registered and tracked in the system by the JM via tables detailing the state of all jobs, be they pending, currently executing, or suspected as failed and under recovery. These various job buffers are frequently checkpointed to the Mass Data Store to enable seamless recovery of the JM and all outstanding jobs. Should an unrecoverable failure of the control processor occur, the JM on the backup controller will load the checkpointed tables upon reboot and continue job scheduling from the last checkpoint. A more detailed explanation of the checkpointing mechanisms is provided in Section 4.7. To ensure the manager's integrity, the JM heartbeats to the FTM via the HA Middleware.

Much like the FTM, the centralized JM employs distributed software agents to gather system and application liveliness information. The JM also relies upon the agents to fork the execution of jobs, including forwarding information required by applications at runtime such as the job's identification number, which is used to uniquely identify checkpointing files. The distributed nature of the agents ensures that the central JM does not become a bottleneck, especially since the JM and other central Dependable Multiprocessor core software components execute on a relatively slow radiation-hardened processor. Numerous mechanisms are in place to ensure the integrity of remote agents running on non-radiation-protected data processors as described in Section 4.1.

In the event of an application failure, the JM refers to a set of user-defined policies to direct the recovery process. In the event one or more processes fail in a parallel application (i.e. one spanning multiple coordinating data processors), then special recovery actions must be taken as dictated by the particular algorithm. Several recovery options exist for parallel jobs including blank mode (i.e. continue with other processors assuming the extra workload), rebuild mode (i.e. the JM either migrates the failed processes to healthy processors or instructs the FTM to recover the faulty components in order to reconstruct the system as before), and shrink mode (i.e. the remaining processes continue by evenly dividing the remaining workload amongst themselves). As mentioned, the ability of a job to recover in any of these modes is dictated by the underlying application. A more detailed discussion of these recovery modes is provided in the next section.

An additional feature that the JM provides is the ability to schedule traditional-processor-only and FPGA-accelerated jobs seamlessly. Portions of the JM have been borrowed from the CARMA runtime job management service framework and middleware [16] developed at Florida, but with improved fault-tolerance capabilities. Also, custom components have been developed to interface with the HA Middleware and other Dependable Multiprocessor services.

4.5. FEMPI

Fault tolerance is a critical factor for HPC systems in space, and is required to meet the emerging high-availability and high-reliability requirements. Recovery from failure needs to be fast and automatic, while the impact of failures on the system as a whole should be minimal. The impact of failures can be minimized through several indirect approaches (i.e. through mechanisms that do not address direct recovery from faults). The indirect approaches certainly avoid computation loss but, in order to enable applications to meet high-availability and high-reliability requirements, we need to consider other options. Some of the options include: incorporating fault-tolerant features directly into the applications; developing specialized hardware subsystems that are fault-tolerant; making use of and enhancing the fault-tolerant features of the operating system; and developing application-independent middleware that would provide fault-tolerant capabilities. Among these options, developing application-independent middleware has the minimal intrusion in the system and can support any general application including legacy applications that fall into the umbrella of the corresponding middleware model. In our system, we design and develop an application-independent, fault-tolerant, message-passing middleware called FEMPI (Fault-tolerant Embedded Message Passing Interface). With FEMPI, we take a direct approach to providing fault tolerance and improving the availability of the HPC system in space. FEMPI is a light-weight, fault-tolerant design and implementation of the common Message Passing Interface (MPI) standard.

Because of its widespread usage, MPI [17] has emerged as the de-facto standard for development and execution of high-performance parallel applications. By its nature as a library that facilitates user-level communication and synchronization amongst a group of processes, the MPI

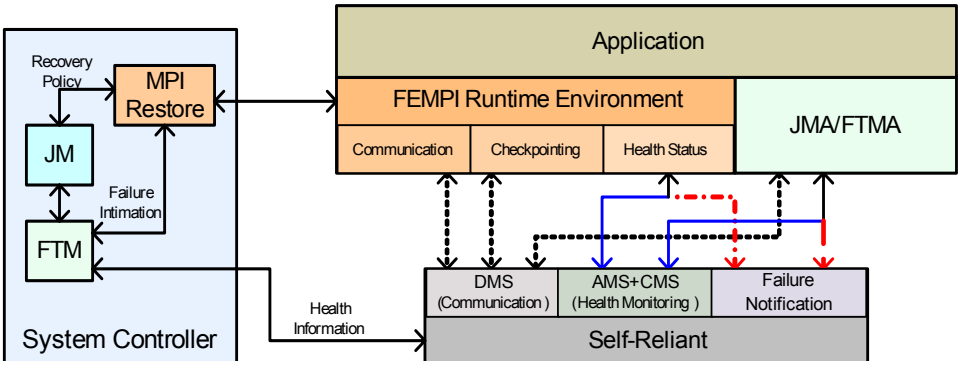

Figure 5. Interfaces for FEMPI and Related Software Components of the Dependable Multiprocessor.

library needs to maintain global awareness of the processes that collectively constitute a parallel application. An MPI library consequently emerges as a logical and suitable place to incorporate certain fault-tolerant features in order to enable legacy and new applications to meet the emerging high-availability and high-reliability requirements of HPC systems in space. Freeware implementations are also underway, but significantly lag commercial efforts, and both forms are generally too heavy and laden with overhead for embedded systems. Fault tolerance is absent in both the MPI (MPI-1 and MPI-2) standards, and to our knowledge no satisfactory products or research results offer an effective path to providing scalable embedded computing applications with effective fault tolerance. FEMPI is a fault-tolerant MPI design and implementation that provides process-level fault tolerance at the MPI API level.

Fault tolerance and recovery is provided through three stages including detection of a fault, notification of the fault, and recovery from the fault. As with other Dependable Multiprocessor software components, FEMPI is built on top of the HA Middleware. The services of the HA Middleware in conjunction with the FTM and JM are used to provide detection and notification capabilities. The HA Middleware allows processes to heartbeat through fault handlers, and hence has the potential to detect the failure of processes and nodes. The notification service is developed as an extension to this middleware. The HA Middleware also guarantees reliable communication between the nodes in the system through DMS as described in Section 4.1.

With MPI applications, failures can be broadly classified as process failures (individual processes of MPI application crashes) and network failures (communication failure between two MPI processes). FEMPI ensures reliable communication (reducing the chances of network failures) with all low-level communication through DMS. As far as process failures are concerned, the entire application fails or crashes on the failure of any process in regular fault-intolerant MPI designs. By contrast, FEMPI prevents the entire application from crashing on individual process failures. MPI Restore, a component of FEMPI, resides in the System Controller and communicates with the FTM to update the status of nodes. On a failure, MPI Restore informs all the MPI processes regarding the failure. The status of senders and receivers (of messages) are checked in FEMPI before communication to avoid attempts to establish communication with failed processes. If the communication partner (sender or receiver) fails after the status check and before communication, then a timeout-based recovery is used to recover out of the MPI function call.

FEMPI survives the crash of *n*-1 processes in an *n*-process job, and, if required, can re-spawn/restart them. However, it is still the responsibility of the HA Middleware to execute a recovery scheme (i.e. recover the data structures and the data on the crashed processes). A program written in conventional MPI can execute over FEMPI with little or no alteration.

4.6. FPGA Co-Processor Services

Using FPGAs for reconfigurable computing to accelerate scientific applications is still an emerging discipline within computer engineering. Until recently it has been confined to relatively few outside the computer science and engineering fields due to the complexity of the intrinsic hardware design. The RC discipline is fractured and populated with proprietary solutions. Universal standards that power the software industry, compile-time libraries, a universal run-time environment and reliable middleware, etc. all do not as yet exist. Several vendors such as Nallatech and SRC provide top-down solutions for FPGA development, but these are based around proprietary interfaces and closed-source APIs. Often, a specific RC platform must be targeted before application development can begin. This method is intolerable in the software industry where code written to language standards (e.g. ANSI-C) can be ported to multiple operating systems and instruction set architectures. Porting an application to another vendor's RC platform is often a major task, as substantial portions of the hardware and software need to be rewritten.

The USURP framework is being developed by researchers at the University of Florida as a unified solution for multi-platform FPGA development. A compile-time interface between software and hardware and a run-time communication standard have been developed to support the framework (Fig. 6). As described in [18], the compile-time

Reprinted from the *Proceedings of the 2006 IEEE Aerospace Conference*, Big Sky, MN, March 6, 2006

hardware/software interface is responsible for unifying vendor software APIs, standardizing the hardware interface to external components and the communications bus, organizing data for the hardware-accelerated kernels, and exposing the developer to common FPGA resources. The run-time communication standard handles determining whether the resources meet the application's requirements, configuring the FPGA, detecting and handling hardware faults and interrupts, and transferring data between the host PC and FPGA.

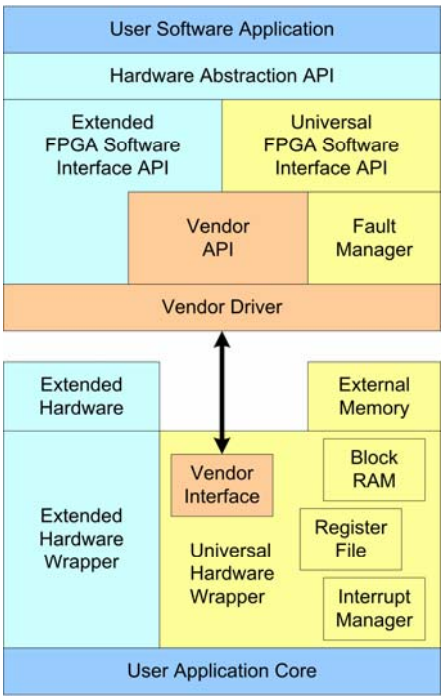

Figure 6. Hardware/Software Interface of USURP.

The Hardware Abstraction API [19] abstracts the FPGA from the application developer; the reconfigurable hardware becomes just another computing resource. To accomplish this goal, the USURP hardware/software interface and run-time communication standard are encapsulated in a familiar library of linear algebra and signal processing kernels. The Hardware Abstraction API is based on the GNU Scientific Library (GSL). GSL is an open-source library of numerical routines for scientific computing and remains popular in the science and engineering community due to its highly portable nature. RCGSL, our hardware-accelerated version of GSL, uses the same structures and syntax as GSL to provide the user with a familiar programming environment.

4.7. Checkpoint Interface

The checkpointing service provides a user-level, uncoordinated protocol for storing and recovering system state, application data, and any data transferred to or from Mass Data Store (MDS). The service comprises a server process that runs on the MDS and an API for the applications that want to communicate data.

The main server process facilitates all data operations between the applications and radiation hardened mass memory. DMS is used to reliably transfer data, using its many-to-one and one-to-one communication capabilities. Checkpoint and data requests are serviced on the Mass Data Store in parallel to allow for multiple simultaneous checkpoint or data accesses.

The application-side API consists of a basic set of functions that allow data to be transferred to the MDS in a fully transparent fashion. These functions are similar to C-type interfaces and provide a method to write, read, rename, and remove stored checkpoints and other data files. The API also includes a function that assigns each application with a unique name that is used for storing checkpoints for that particular application. This name is generated based upon the name of the application and a unique job identifier and process identifier defined by the central JM when the job is scheduled. Upon failover or restart of an application, the application may check the MDS for the presence of specific checkpoint data, use the data if it is available, and complete the interrupted processing. Checkpoint content and frequency is determined by the process that chooses to checkpoint.

4.8. Algorithm-based Fault Tolerance (ABFT) Library

The Algorithm-Based Fault Tolerance (ABFT) library is a collection of mathematical routines that can detect and in some cases correct data faults. Data faults are faults that allow an application to complete, but may produce an incorrect result. The seminal work in ABFT was done in 1984 by Huang and Abraham [20]. Subsequently, the JPL-led REE project developed a parallel processing ABFT library. The BLAS-3, ABFT-enabled library from JPL includes functions such as matrix multiply, LU decomposition, QR decomposition, single-value decompositions (SVD) and fast Fourier transform (FFT). This library is being ported to the Dependable Multiprocessor for use by application developers as a fault-detection mechanism. ABFT operations function by checking on linear algebraic computations by adding check-sum values in extra rows and columns of the original matrices and then checking these values at the end of the computation. The mathematical relationships of these checksum values to the matrix data is preserved over linear operations. An error is detected by re-computing the checksums and comparing the new values to those in the rows and columns added to the original matrix. If an error is detected, an error code is returned to the calling application. The appeal of ABFT over simple replication is that the additional work that must be done to check operations is of a lower order of magnitude than the operations themselves. For example, the check of an FFT is $O(n)$, whereas the FFT itself is $O(n \log n)$.

In Dependable Multiprocessor, ABFT-enabled functions will be used by the application developer to perform automated, transparent, low-overhead error checking on

Reprinted from the *Proceedings of the 2006 IEEE Aerospace Conference*, Big Sky, MN, March 6, 2006

linear algebraic computations. In the longer term, it is expected that other, non-algebraic algorithms will similarly be ABFT-enabled and added to the library. The user will determine, from the returned error code, whether and how to address the error. As part of this effort, we will add application-level APIs, which when called will inform the Job Management Agent (JMA) that a fault has occurred. The JMA will then inform the FTM, and the FTM will determine a course of action. A typical response would be to stop the application and restart from checkpointed values.

4.9. Replication Services

Replication and comparison is a well known method to detect errors in a system. One typical replication technique is hardware replication, wherein the application is replicated on one or more processing resources and the results of the computation amongst all the processors are compared. In Triple Modular Redundancy (TMR), if two or more results agree, that result is taken as correct. If two or more disagree, then an uncorrectable fault as been observed and additional action is needed. Another technique is process-level replication, in which multiple identical processes are instantiated on a single processing resource and their results compared for consistency

In this NMP experiment, since resources are limited, process-level replication is implemented where two identical processes are spawned on a single processing resource. The user will insert provided application-level API calls at locations in the program where results are exchanged. The results of the application replicas are then compared for consistency before forwarding. In the event of a miscompare, an error code is returned to the calling application. The user application will determine, from the returned error code, whether and how to address the error. Similar to ABFT, the user application will invoke an application-level API to inform the JMA that an error has occurred and that corrective action is required.

5. CONCLUSIONS

NASA's strategic plans for space exploration present significant challenges to space computer developers. Traditional methods and architectures fall short of the requirements for next-generation missions. The Dependable Multiprocessor (DM) technology addresses this need and provides the foundation for future space processors. The Dependable Multiprocessor is an integrated parallel computing system that addresses all of the essential functions of a cluster computer for spacecraft payload processing. A TRL4 prototype of the technology has been demonstrated, and a TRL5 prototype will be completed in Spring of 2006. The next step in the development of Dependable Multiprocessor includes a TRL6 prototype, scheduled for completion in 2007, followed by a TRL7 prototype validation flight experiment in 2009 [21].

REFERENCES

[1] J. Ramos, et. al., "Environmentally Adaptive Fault Tolerant Computing," IEEE Aerospace Conference, Big Sky, Montana, March 2005.

[2] "NASA 2003 Strategic Plan," NP-2003-01-298-HQ.

[3] R. Some and D. Katz, "NASA Advances Robotic Space Exploration," IEEE Computer, IEEE Press, Volume 36, Issue 1, Jan. 2003, Pages. 52 to 61.

[4] Home page of the New Millennium Program, http://nmp.jpl.nasa.gov/.

[5] J. Ramos, R. Sowada, and D. Lupia, "Scientific Computing in Space Using COTS Processors," Proc. International Conference on Military and Aerospace Programmable Logic Devices (MAPLD), Washington, DC, Sep. 7-9, 2005.

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

[7] E. Prado et al., "A Standard Approach to Spaceborne Payload Data Processing," IEEE Aerospace Conference, Big Sky, Montana, March 2001.

[8] G. Brown, "Radiation Hardened PowerPC 603e ™ Based Single Board Computer," 20th Digital Avionics Systems, Oct. 2001.

[9] IBM Corporation, "PowerPC 750FX Microprocessor User's Manual," Feb 2003 http://www-306.ibm.com/chips/techlib/techlib.nsf/products/PowerPC_750FX_Microprocessor.

[10] Xilinx Corporation, "QPro Virtex 2.5V Radiation Hardened FPGA," Xilinx Web site, http://www.xilinx.com/, Nov. 2001.

[11] J. Donaldson, "Push the DSP Performance Envelope," Xilinx Xcell Journal, Spring 2003.

[12] IEEE Standard 802.3ab, and IEEE Standard802.3z.

[13] Orion Technologies, Inc. CPC7510 Single Board Computer webpage http://otisolutions.com/cpc7510.html.

[14] Alpha Data Parallel Systems ADM-XRC-II webpage http://www.alpha-data.com/aDependable Multiprocessor-xrc-ii.html.

[15] GoAhead web site, http://www.goahead.com/.

[16] I. Troxel, A. Jacob, A. George, R. Subramaniyan, and M. Radlinski, "CARMA: A Comprehensive Management Framework for High-Performance Reconfigurable Computing," Proc. International Conference on Military and Aerospace Programmable Logic Devices (MAPLD), Washington, DC, Sep. 8-10, 2004.

[17] Message Passing Interface Forum, MPI: A Message-passing Interface Standard, Technical Report CS-94-230, Computer Science Department, University of Tennessee, April 1, 1994.

[18] J. Greco, B. Holland, I. Troxel, G. Barfield, V. Aggarwal, and A. George, "USURP: A Standard for Design Portability in Reconfigurable Computing," submitted to IEEE Symp. on FCCM, Napa Valley, CA, April 24-26, 2006.

[19] J. Greco, G. Cieslewski, A. Jacobs, I. Troxel, and A. George, "Hardware/software Interface for High-performance Space Computing with FPGA Coprocessors," Proc. IEEE Aerospace Conference, Big Sky, MN, March 4-11, 2006 (to appear).

[20] K. Huang and J. Abraham, "Algorithm-Based Fault Tolerance for Matrix Operations", IEEE Trans. on Computers, Vol. C-33, No. 6, pp. 518-528, June 1984.

[21] John R. Samson, Jr., et. al., "Technology Validation: NMP ST8 Dependable Multiprocessor Project," Proceedings of the 2006 IEEE Aerospace Conference, Big Sky, Montana, March 2006.

ACKNOWLEDGEMENTS

The authors would like to thank the following people and organizations for their contributions to the Dependable Multiprocessor effort: Brian Heigel, Paul Arons, Gavin Kavanaugh, and Mike Nitso from GoAhead Software, Inc. Other members of the team are Dr. Ravishankar Iyer and Dr. Zbigniew Kalbarcyk from the University of Illinois and Armored Computing Inc.

The Dependable Multiprocessor effort is funded under NASA NMP ST-8 contract NMO-710209.

BIOGRAPHIES

Jeremy Ramos earned the B.S. in Computer Science and Engineering, and is currently a Ph.D. student at the University of South Florida. Mr. Ramos has been a Honeywell Aerospace employee since 1999, and is presently a Technical Director and Systems Engineer. His most recent assignments at Honeywell included the Honeywell Reconfigurable Space Computer (HRSC) project, and the New Millennium Space Technology 8 Dependable Multiprocessor project. His numerous technical contributions have resulted in several patent applications, and a patent award. Prior to his engineering career Mr. Ramos served for 7+ years with the United States Army as a Technician in the Army Ordnance Core. Mr. Ramos' research interests include computer architecture, system simulation, and reconfigurable computing. He is a member of the IEEE.

John Samson is a principal engineering fellow at Honeywell Aerospace in Clearwater, Florida. Dr. Samson has 35+ years of experience in onboard processing for space and airborne applications and has published more than 40 papers in the area of onboard processing systems and architectures. He is a senior member of the IEEE and an associate fellow in the AIAA.

David Lupia came to Honeywell Inc. in 2005. He has over 10 years of experience in the design and development of space and military electronics systems, with his core expertise in Digital Signal Processing and Communication Systems. In 1993 he graduated from Ohio University with Bachelor of Science in Electrical Engineering, and subsequently joined Raytheon. He has received 3 patents in the field of satellite communication systems and advanced waveform development. He is a Lead Systems Engineer on the Dependable Multiprocessor project. His research interests include coding theory, advanced waveform development, and fault tolerant reconfigurable computing systems

Ian Troxel is a Ph.D. candidate in Electrical and Computer Engineering at the University Florida. He is a research assistant who co-leads the advanced space computing and the reconfigurable computing research groups at the High-performance Computing and Simulation Research Laboratory. His research interests include reconfigurable and embedded computing and he is a student member of the IEEE.

Rajagopal Subramaniyan is a Ph.D. student in Electrical and Computer Engineering at the University of Florida. He co-leads the high-performance computing and communication group and is also a member of the advanced space computing group at the High-Performance Computing and Simulation Research Laboratory. His research interests include high-performance computing, systems and networks.

Adam Jacobs is a Ph.D. student in Electrical and Computer Engineering at the University of Florida. He is a research assistant in the Advanced Space Computing and Reconfigurable Computing groups at the High-Performance Computing and

Simulation Research Laboratory. His research interests include fault-tolerant FPGA architectures and high-performance computing. He is a student member of the IEEE.

James Greco is a Ph.D. student in the Electrical & Computer Engineering Department at the University of Florida. He is a research assistant and member of the Advanced Space Computing and Reconfigurable Computing groups in the High-performance Computing & Simulation Research Laboratory. His research interests include reconfigurable computing in HPC and the hardware acceleration of signal processing applications. He is a student member of IEEE.

Grzegorz Cieslewski is a graduate student at the University of Florida where he is currently pursuing a Ph.D. degree in Electrical and Computer Engineering. As a research assistant he is a member of Advanced Space Computing and Reconfigurable Computing groups at High-performance Computing & Simulation Research Laboratory. His research interests include computer architecture, reconfigurable, fault-tolerant and distributed computing as applied to linear algebra problems and signal processing. He is a student member of IEEE.

John Curreri is a graduate student at the University of Florida where he is currently pursuing a Masters degree in Electrical and Computer Engineering. As a research assistant he is a member of Advanced Space Computing group at High-performance Computing & Simulation Research Laboratory. His research interests include parallel, reconfigurable and fault-tolerant computing. He is a student member of IEEE.

Michael Fischer is pursuing a Master's Degree in Electrical and Computer Engineering at the University Florida. He is a research assistant who is a member of the advanced space computing group at the High-performance Computing and Simulation Research Laboratory. His research interests include fault tolerance and availability of embedded systems.

Eric Grobelny is a Ph.D. student in Electrical and Computer Engineering at the University of Florida. He works as a research assistant at the High-performance Computing & Simulation Research Laboratory. His main focus of research is in performance prediction of parallel scientific applications for clustered and embedded systems through modeling and simulation. He is also the team leader of the Mission Assurance group which focuses on disaster recovery and business continuity in high-performance computing environments.

Alan D. George is Professor of Electrical and Computer Engineering at the University of Florida, where he serves as Director and Founder of the HCS Research Laboratory. He received the B.S. degree in Computer Science and the M.S. in Electrical and Computer Engineering from the University of Central Florida, and the Ph.D. in Computer Science from the Florida State University. Dr. George's research interests focus on high-performance architectures, networks, services, and systems for parallel, reconfigurable, distributed, and fault-tolerant computing. He is a senior member of IEEE and SCS, and can be reached by e-mail at *george@hcs.ufl.edu*.

Minesh I. Patel is a systems and software architect and consultant with Tandel Systems in Clearwater, Florida. He received his BSEE and BSCpE in electrical and computer engineering and his MSCpE and Ph.D. in Computer Science and Engineering from the University of South Florida. His research and technical interests include software and system fault tolerance, artificial intelligence and machine learning, embedded and real-time systems and high-performance, parallel and distributed computing. Dr. Patel is Lead Software Architect for the Dependable Multiprocessor project.

Vikas Aggarwal is a systems engineer with Tandel Systems in Clearwater, Florida. He received his B.Tech. degree in Electronics and Communications Engineering from G.G.S. Indraprastha University, Delhi, India. He then moved over to United States and received his MS degree in Electrical and Computer Engineering from University of Florida. His research interests include reconfigurable and embedded computing and system fault-tolerance, high-performance, parallel and distributed computing.

Raphael Some is a program technologist at JPL for the New Millennium Program. He has served as Contract Technical Manager and Leader of the Technology Review Board for the ST8 Dependable Multiprocessor project. Prior to his involvement with the NMP ST8 project, Mr. Some was the Chief Engineer for the Remote Exploration and Experimentation Project at the Jet Propulsion Laboratory. Previously, at JPL, he formulated and managed the Smart Sensor project. His experience prior to JPL includes the development of fault tolerant space based supercomputers as well as a variety of avionics and signal processing systems for both commercial and military applications. He holds a BSEE from Rutgers University.

Reprinted from the *Proceedings of the 2006 IEEE Aerospace Conference*, Big Sky, MN, March 6, 2006