# Reliable Multicore Processors for NASA Space Missions

Carlos Villalpando, David Rennels, Raphael Some
carlos@jpl.nasa.gov, rennels@cs.ucla.edu,
rsome@jpl.nasa.gov
Jet Propulsion Laboratory
4800 Oak Grove Dr. Pasadena, CA 91109

Manuel Cabanas-Holmen
manuel.f.cabanas-holmen@boeing.com
The Boeing Company
PO Box 3707, MS 42-57, Seattle, WA 98124

*Abstract*—The current trend in commercial processors of moving to many cores (30 to100 and beyond) on a single die poses both an opportunity and a challenge for space based processing. [1] [2] The opportunity is to leverage this trend for space application and thus provide an order of magnitude increase in onboard processing capability. The challenge is to provide the requisite reliability in an extremely challenging environment. In this paper, we will discuss the requirements for reliable space based multicore computing and approaches being explored to deliver this capability within NASA's extremely tight power, mass, and cost constraints.

Topics include: i) discussing the salient issues in achieving fault tolerance in a many-core chip, ii) describing the architecture of an existing commercial many-core processor (Tilera Tile64), iii) using it for an examination of the design issues needed for increasing levels of reliability, and iv) a discussion of how the Tile64 is being adapted for space as the Maestro processor and of the tradeoffs involved in making it a practical space design.

The OPERA Maestro processor based on the Tilera TILE64 architecture shows potential to give high processing performance at an error rate equivalent to current space deployed uniprocessor systems. We will discuss potential enhancements to the Maestro processor to address NASA specific performance and reliability requirements.

**TABLE OF CONTENTS**



## 1. INTRODUCTION

Although current multicore processors with two to four cores have become widespread, it is becoming clear that the next generation of this technology will replicate a much large number of smaller, more basic processors, creating what is termed many-core" processors. Examples of multicore machines include the HyperX from Coherent Logix, the 64-core TILE64 from Tilera [1] and a new experimental 80 core processor from Intel [2]. These processors consist of tens to low hundreds of homogenous processing elements, connected by a high speed mesh or crossbar grid network. The parallelism that exists on these processors can improve not only system level performance, but the granularity available from a large number of tightly integrated cores can speed up individual tasks or applications as well. Various NASA programs and missions are in need of reliable and flight qualifiable high performance processing for both critical applications and on-board science processing.

By using many-core processors, high performance can be obtained at drastically reduced mass and volume, and compared to an equivalent multicomputer with computers on separate chips and boards. This paper focuses on two salient design issues critical to achieving a high degree of dependability in these systems: (1) Redundancy for Permanent Fault Recovery and (2) Mechanisms for Detecting and Recovering from Transient Errors. We start with a brief discussion of the issues involved then describe the functionality of a real multicore space computer and discuss how these issues might be dealt with in that context.

Our discussion is anchored on one such manycore system that is under consideration. The Maestro processor developed by the OPERA program is a radiation hardened by design processor based on the TILE64 processor by Tilera, We will discuss the TILE64 and the changes made by OPERA and how it addresses fault tolerance requirements.

*Redundancy for Permanent Fault Recovery*

There is a great deal of inherent redundancy in large multicore chips (typically with dozens of processor cores, redundant I/O and memory ports, power supply pins, etc) to allow degraded recovery from a wide range of permanent faults. Thus upon failure of a processor core, interconnection switch, I/O or memory port, the computing load can be shifted to other resources on the chip. This approach is imperfect since there are some faults that can disable the whole chip, but it can lead to considerably improved overall reliability, i.e., the length of time that the computer can be expected to provide required performance in space. However, the effectiveness of this approach depends upon a multicore chip design that is optimized to minimize the probability of single failures that disable the

[1] 978-1-4244-7351-9/11/$26.00 ©2011 IEEE.
[2] IEEEAC paper #1163, Version 5, Updated November 22, 2010

Aerospace Conference, 2011 IEEE , vol., no., pp.1-12, 5-12 March 2011
doi: 10.1109/AERO.2011.5747447

whole chip, and an architecture that can efficiently work around failed cores, memory and I/O elements. Redundancy can be implemented as fine grained structures such as triplicated modules, robust latches, and structured code, or it can be implemented as gross methods such as duplicating effort across multiple processing elements managed by software checking

*Detection and Recovery from Faults and Errors*

In order to recover from a permanent fault or transient error, it is necessary to detect that an error has occurred and to provide an automated recovery action to restore computations. Key issues here are: i) how effective is the error detection provided? ii) to what degree can computations be correctly restored after the error has been detected? and iii) how is the recovery mechanism itself protected? Since the processors on many-core chips are unlikely to have comprehensive fault tolerance at the hardware level, this must necessarily depend upon software-implemented fault tolerance. Software implemented fault tolerance (SIFT) depends upon a reliable messaging system and isolation of faults to individual processors so that protective redundancy can be effectively employed – especially in preventing errors from causing data damage that escapes virtual memory protection boundaries. These present real challenges for multicore chip designs. Before exploring these issues it is useful to explore alternatives.

*Gross Detection and Recovery Mechanisms*— If very gross detection mechanisms are employed (such as looking for a heartbeat or crash from a machine and activating a spare), recovery consists of trying a rollback recovery and if it is not successful, throwing out existing computations followed by reloading and rebooting the machine. In this case, errors are likely to have propagated before detection and may have produced incorrect outputs or damaged system state – making recovery of computations problematic. This approach is typical of many existing spacecraft where few errors are expected during a space mission due to radiation hardening of hardware parts. In this case, another spacecraft computer (often a smaller, extremely radiation hardened processor) can intervene, or specially designed logic circuits can be used to provide "safe hold modes". These architectures provide a hierarchical level of protection above that of the standard operational spacecraft avionics. The applicability of these gross techniques may be adequate for non-critical applications depending on the effectiveness of radiation hardening techniques and the mission requirements, e.g., if a mission only expects a handful of SEUs during a mission and the probability of correct recovery using these techniques is around 99%. Validating the adequacy of this approach depends, in turn, upon having thorough radiation testing for the parts.

*Comprehensive Detection and Recovery Mechanisms*— These are designed to detect errors before an incorrect output is generated and before error propagation reduces the chance for recovering correct computations. When an error is detected, recovery is implemented using well-known checkpointing and rollback techniques. Here computations are checked at specified test points before outputs are generated, and state is updated using redundancy built into the computations. Highly structured computations can use software methods such Algorithm Based Fault Tolerance (ABFT) quite efficiently (these are really algorithm based error checking techniques), but unstructured computations must use replicated computations with comparisons. The software that implements the detection and recovery functions is preferably triplicated and voted so that it cannot be disrupted by a single fault.

This approach has been demonstrated in current multicomputer cluster based systems that depend upon independence of faults in different circuit processors. But to use this for a many-core architecture i) the probability of faults in common circuits such as the clock and overall control must be greatly minimized, ii) a dependable communication mechanism must allow a processor tile to know the source of a message and that received messages are correct, and iii) strong virtual memory protections must be provided to prevent errors in one processor core, communications link, or peripheral device from affecting other cores.

## 2. THE TILE64 ARCHITECTURE

*Overview*[1]

The Tile64 system is a multi-core processor system on a chip containing an array of processing elements and data routers, called switch engines, connected in a Manhattan style grid as illustrated in Figure 1. Each tile in the array contains a 32 bit 3-way VLIW general purpose Processing Element (PE) and a data router with five data channels. Tilera has released updated versions of this architecture in the TilePro and TileGX series. This paper is limited to the architecture as it exists in the Tile64 series only. There are enhancements that are designed into the the TilePro and TileGX series that increase fault tolerance performance above that of the TILE64, but are outside of the scope of this document.

The PEs are designed to operate independently, and each contains its own Level 1 and Level 2 caches, with a TLB and DMA engine. Each PE is capable of running a full featured operating system such as Linux or VxWorks, or running bare applications. The L1 and L2 caches are designed so that shared memory programming is fully supported, with either hardware supported or user managed cache coherency.

At the core of the Tilera architecture is a mesh network that interconnects the Tilera tiles, and I/O subsystems. There are five parallel networks in the mesh. For each, the path between modules is 32-bits wide, the same as words and addresses in the tiles. Of the five networks, four are designated dynamic networks. They are packet switched,

and wormhole routed, and each is dedicated to a particular function, i.e.,. memory access requests and memory data transfer, messaging between applications programs, and OS messages. The fifth is called the static network**.** It does not provide packet address decode and routing, but instead provides fixed, pre-programmed routing. It can be programmed as part of a parallel application to move data in user-specified static patterns.

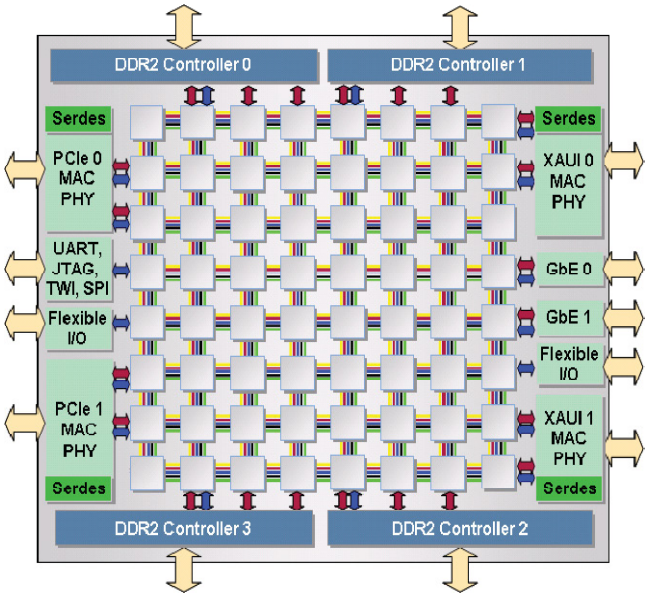


**Figure 1: General architecture**

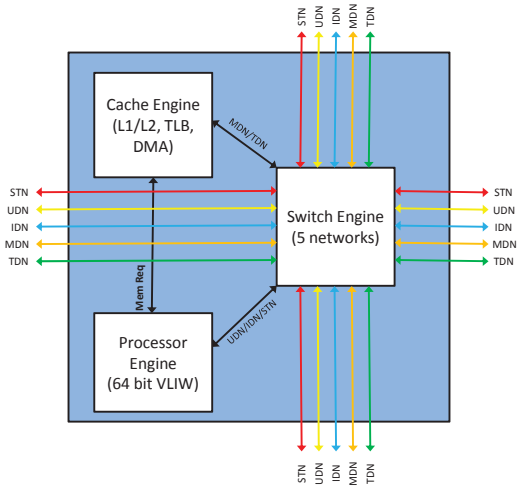


**Figure 2: Single Tile Overview**

The five networks are a Static Tile Network (STN), a User Dynamic Network, (UDN), a Memory Dynamic Network (MDN), a I/O Dynamic Network (IDN), and a Tile Dynamic Network (TDN). A simplified block diagram of an individual tile is shown in Figure 2 and a summary of its functionality is listed in Table 1.

The Dynamic networks are packet based, are "fire and forget" and use "wormhole routing." Their routes do not need to be set up ahead of time. Each tile Switch Engine can route multiple packets per network at the same time. Upon receipt of a packet header, the Switch Engine examines the header to determine which direction to route the packet. The packet is routed using a dimension ordered routing policy. The packet is first routed along the X dimension until the destination column is reached, and then it is routed along the Y dimension until the destination tile is reached.

A simplified diagram of the Switch Engine is shown in Figure 3.

*Dynamic Networks:*

*UDN (User Dynamic Network)* —The UDN is a user accessible routing network. It is directly accessed by the processor via direct reads and writes to special purpose registers. Because it is tightly integrated with the ALU, low latency blocking sends and receives are available. In this mode, if the network is unable to accept an outgoing packet, or an incoming packet is not yet available, the processor will sleep until the network is ready.

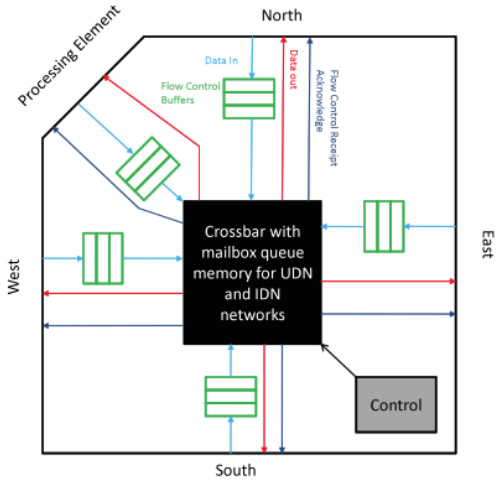


**Figure 3: Switch Engine unit, each incoming direction has a multiple word queue for flow control. Each tile contains one switch unit per network.**

A field in the dynamic header tag word specifies a demux or mailbox queue. Each processor can send to or receive from 4 queues on each tile. An incoming packet that is meant for the receiving tile examines the tag and places the packet in the appropriate queue. These queues are accessible by the user through special purpose registers on each PE that are allocated for each queue. This allows differently tagged messages to be serviced out of order. Non tagged messages are placed in a "catch-all" queue, and are serviced in the order they are received. Each queue is a part of a larger packet memory that is shared with the IDN. Partitioning of the shared packet memory is performed by system level software and is not modifiable by the user. When a packet arrives at a destination, an interrupt is triggered at the

destination processor and a flag is set to indicate which queue has data available. The presence of a queue buffer allows multiple incoming packets to be stored in the switch engine and allows the PE to process the incoming packets at its own pace, and allows the network to be freed up for routing other packets.

There is no data protection mechanism described for the packet queue memories or the queue management. There is program access protection for each network at each PE. Access to the networks can be limited to privileged instruction levels.

To prevent sending messages across a user defined boundary, the Tile64 utilizes "hardwall" protection scheme. The hypervisor running on the local PE specifies, for each output port, if message traffic is allowed to leave that port. This hardwall is implemented at each tile such that if a message is to be routed to a protected port, the message is not sent through that port, and instead an interrupt will be triggered on the tile's local PE. A similar hardwall mechanism exists for the Static Data Network.

*MDN (Memory Dynamic Network)*—The MDN is used to transfer data between external memory and the processor caches and for responding to requests on the TDN. It operates as a dynamic network, but does not have the hardwall mechanism of the UDN and STN networks. *It is not user accessible and is accessed only by the L2 cache engine.*

*TDN (Tile Dynamic Network)*—The TDN is used to initiate tile-to-tile memory requests, i.e., to move data between tile caches. The responses for requests on the TDN are delivered on the MDN. *The TDN is only accessible by the L2 cache engine.*

*IDN (I/O Dynamic Network)*—The IDN is the primary network for system level software such as a hypervisor or guest operating system to control and coordinate messages and system level operations such as task spawning, processor allocation, and device management. It also is the primary method of communicating with the various I/O devices on the chip. It operates as a dynamic network with two multiplexed channels per processor similar to the four queues of the UDN. The queues are part of a larger packet memory shared with the UDN and operate identically to the UDN with the exemption of only having 2 queues instead of 4. The IDN is a privileged network and is not accessible to user level code. The IDN, along with the STN, is also used by boot level code to distribute PE configuration and initial boot code to each processing element. The hardwall protection mechanism is also implemented for the IDN and operates identically to the UDN's hardwall protection.

*The Static Network:*

*STN (Static Network)*—The STN provides low overhead, low latency tile to tile communication to transfer operands and is meant for repetitive and well known communication. Data flows through the network a word at a time with no concept of discrete packets or any higher level of abstraction. Low overhead is attained by programming a routing processor in each tile ahead of time with a fixed route map. The switch engine in each processor is programmed with a destination direction for each source direction. The network remains configured for that pattern until the switches are changed to another routing configuration. A message can be routed to one or more output ports. A message can be transferred from tile to tile as fast as 1 clock per word per node traversed. There is also flow control on a per tile hop basis.

Each link buffers three words of storage, and the sender therefore begins with three credits. A sender decrements its credit count when it sends a word, and increments the credit count when it receives acknowledgement from the receiver. The send/receive flow control handshaking is handled entirely in hardware by the switch engine. A sender can only send when its count is non-zero.

The STN incorporates hardwall protection. There is no deadlock prevention for this network as it is not needed since routing is determined by software. If a failure of a routing processor is detected, it is possible to re-program the routing of the static network to avoid using the failed routing processor.

Table 1 summarizes the data networks.

**Table 1. Network types and function**

| Network | Access type | Routing type | Hardwall | Used for |
|---|---|---|---|---|
| STN | User | Static | YES | User messaging |
| UDN | User | Dynamic | YES | User messaging |
| MDN | System | Dynamic | NO | Memory access |
| TDN | System | Dynamic | NO | Memory Access |
| IDN | System (USER if not running an O/S) | Dynamic | YES | I/O and System Messaging |

*Network Fault Tolerance Issues:*

None of the Switch Engines employ internal redundancy (e.g. parity) in hardware for error detection. This is a potential reliability problem in a space environment as will be discussed later.

*The Memory System*

*Main Memory*—The memory system allows four channels of RAM each connected to the switch network by a DDR2

controller. The DDR2 controllers run at 400MHz and are 64 bits wide, with an optional 8 bit ECC with single bit error correction, double bit error detection, (SEC/DED) protection. Each memory controller is connected to the internal data network by three MDN ports, and one IDN port.

The DDR2 controllers accept read and write memory packets from the tiles' cache controllers or from the DMA engines in the I/O interfaces via the MDN -- returning data packets upon receiving reads and returning an acknowledgement packet for writes. A complex queue controller can reorder requests for increased performance. Control packets are also communicated via the IDN.

*Caches and Virtual Memory*—Each PE contains two caches; a split L1 cache and a combined L2 cache. The L1 cache contains an 8KB write-through data cache, and an 8KB instruction cache. The L2 cache is a 64KB combined instruction and data write-back cache, and includes a DMA engine to handle cache misses and write backs. The cache and TLB properties are shown in Table 2 – along with a summary of their error protection.

**Table 2: Data on Cache and TLB**

| | Size | Type | Line Size | Write Policy | Protection |
|---|---|---|---|---|---|
| L1 Instruction | 8KB | Direct | 64B | NA | 64-bit parity |
| L1 Data | 8KB | 2-way associative | 16B | Write through | 8 bit parity |
| L2 | 64KB | 2 way associative | 64KB | Write back | 8 bit parity |
| Instruction TLB | 8 entries | Fully associative | | | |
| Data TLB | 16 entries | Fully Associative | | | |

*Memory-Cache Fault Tolerance Issues*

Although the caches are protected using an error detecting code, the tags and the TLBs are not. The TLBs on the TILE64 consist of processor flops and logic cells, so from an error rate point of view, errors in the TLB would be lumped in the same category as processor errors. Furthermore, although the L2 cache is write-back, there is no single bit error correction, so error recovery will be complicated and time consuming. Hardware-based error detection is not provided for the TLB, cache control sequencers and the DMA controllers, so errors may violate virtual memory protection boundaries

*Input/Output*—There are several I/O options available on the Tile64. As well as the 4 DDR2 controllers, there are 2 10Gb/s XAUI ports, two 10/100/1000 Mbit Ethernet MACs, an HPI interface, 4 banks of 16 bit General Purpose I/O, a 2 wire UART, an I2C port, a Serial ROM port, and 2 4-lane PCIe ports. Each of these I/O options are connected to the internal PE mesh by I/O "shims." The shims provide a protocol translation between the I/O device and the internal data networks. The shims can perform DMA to and from on chip caches and external memory.

*I/O Fault Tolerance Issues:*

Since none of the intercommunications networks use hardware based error detection, end-to-end protection of data presents a reliability problem as does protection of local hardware controllers (e.g. state machines) and DMA controllers in the and DDR2 interfaces, cache's cores, and I/O devices.

*Processing Elements*

All the processing elements on the chip are identical. They are a 64-bit, VLIW processor. Each 64 bit instruction word is called an instruction bundle and can encode two or three instructions. Each bundle can handle two ALU and one Load/Store instruction.

*Processing Element Fault Tolerance Issues*

In order to achieve low power and high performance in the commercial marketplace, it was not to be expected that the PEs implement concurrent error detection. Software-Based Fault-tolerance can be expected to address this problem. However, it must be noted that when a tile runs the hypervisor or an underlying OS computing errors can violate virtual memory boundaries or corrupt system state tables. This provides a challenge in implementing fault tolerance.

## 3 TILERA FAULT-TOLERANCE CHALLENGES

The TILE64 has used the standard low cost redundancy techniques common for commercial parts to improve its reliability by adding error detection codes to caches and main memory. Since the majority of the active area on the chip lies in these areas, most errors are covered, and the undetected error rate should be greatly reduced. In addition a hardwall protection scheme for the switch network and virtual memory protection is provided to limit error propagation. There is a great deal of redundancy on the chip with a rudimentary way to work around tile faults. Although its design point is well chosen for the terrestrial market (its intended application), where if a processor fails one simply replaces a board, considerably more must be done to achieve adequate dependability in space. In space the transient error rate is likely to be three to four orders of magnitude higher than on Earth due to ionizing radiation, and many missions require long unmaintained life.

Returning to Section 1 above we examine potential deficiencies of the Tile64 for space use and in the following Section 4 we will examine what has been done to address some of these problems in the new radiation hardened Maestro Chip. This is not a fault of the TILE64, as with typical commercial parts, fault tolerance for space applications is not a priority or design requirement. The OPERA program was tasked to build on the TILE64 architecture to add fault tolerance.

*Redundancy for Permanent Fault Recovery Revisited*

Due to the dimension ordered routing scheme of the Tile64, when a tile router fails the other tiles in the row and column in which the failure occurred are blocked in some of their communications. This can be worked around by a rearrangement of the programs in all of the various tiles (possibly with software relaying at a significant performance cost), but it requires shutting the system down and restarting after redesigning the communications patterns. An increase in flexibility of the routing algorithm could make this recovery easier and more efficient. By providing a capability to automatically route around a failed tile, a spare tile anywhere could be used to replace it. Furthermore, when implementing software-based fault tolerance, it is useful for a tile to be able to send a message over two disjoint paths to gain a high degree of assurance that it has been delivered – even in the face of a new permanent fault. It appears that this could be done without overly complicating the Tile64 routing design possibly with an "avoid row/column" feature addition to the current dimension ordered routing policy. If only the processing element of the tile fails, the router is still functional, capable of routing packets, however functionality may be slightly degraded. Answering hardwall interrupts on that tile and/or managing static network routes on that tile may be difficult depending on the failure mode of the processing element.

A more vexing problem is that of single faults or failures in critical shared logic that can disable the whole chip or cause errors in multiple tiles that can only be recovered by a system restart.

*Detection and Recovery from Faults and Errors Revisited*

The Tile64 chip has caches that use hardware based error detection, but much of the rest of the chip has little or no hardware support for error detection. It is possible to partially compensate for this by using software-based fault tolerance techniques, but some errors will go undetected damage system state and require a reboot/restart to restore functionality. For example, it is possible to add redundancy to some packets using software to allow concurrent error detection in some of the dynamic networks and the static network (IDN, STN, and UDN). This cannot be done for the Memory Dynamic Network (MDN) and the Tile Dynamic Network (TDN) because they are totally hardware controlled. Since they do not provide hardware checking, they remain a serious problem. A single bit error in these networks can i) go undetected, ii) and damage a virtual memory space that is different than that of the currently executing process. Similarly the various DMA controllers in the caches, processors or I/O devices may have undetected errors that can create similar damage. This makes Comprehensive Detection and Recovery Mechanisms impossible - resulting in incorrect outputs or the inability for a successful rollback recovery.

Furthermore, since individual computations are being carried out in largely unprotected processors, these processors may have undetected errors while running the Hypervisor. These errors may, in turn, create errors in the DMA controller settings or in shared tables for the hypervisors. This can make error recovery impossible, - corrupting critical state and requiring a reboot/restart to re-establish computing.

*Reducing the Effective Error Rate by Radiation Hardening*

If the error rate in space can be dramatically reduced then Gross Detection and Recovery Mechanisms (see Section 1) may be adequate to meet the needs for some spacecraft applications. The commonly used technique for doing this is to redesign chips to be more resilient using Radiation Hardening by Design. If transient errors can be reduced to a very small number during the life of a mission, then most (perhaps 90%) of them will be automatically recovered, and a very rare reboot/restart may be an acceptable price for the orders of magnitude increase in power-performance provided by these processors. This is the approach taken by the OPERA Program described in section 4.

## 4. MAESTRO

Under the OPERA program, Boeing has developed the Maestro processor, a 49 core version of the Tile64 for space applications.[3] The main approach Boeing used when designing the Maestro processor was to make the new design as functionally equivalent as possible to the original Tile64 design, with the addition of a floating point unit on each PE, and I/O options customized for the Maestro application. To that end, they received a full RTL description of the Tile64 from the Tilera Corporation and worked from that dataset. Boeing used their 90nm bulk CMOS RHBD libraries to increase the radiation tolerance of the processor.

In addition to using their RHBD libraries, Boeing has modified the various I/O devices on the Maestro to meet radiation tolerance requirements, and to tailor I/O options to OPERA program requirements. PCIe was removed for additional XAUI ports, and additional JTAG features were added.

*Floating Point Unit*

Boeing added the Aurora FPU used in Sun Microsystems Sparc chips for the FPU. It is an IEEE-754 single/double precision FPU with multiply-accumulate capability. It is interfaced to the PE via special purpose registers in the PE and custom interrupts for control, data transfer, and synchronization. Interface to the FPU via special purpose registers was chosen in order to avoid having to modify the processor to add extra instructions and exemptions. The FPU instead operates as a "coprocessor" instead of an integrated submodule of the main processor.

*Memories*

The on chip memories such as caches and I/O buffers were modified for additional fault tolerance capability. Artisan/ARM SRAM cells were used and each memory bank was rearranged in order to interleave bits. Bit interleaving physically spreads out logically adjacent bits in order to reduce the probability of multi-bit errors in a single word.

There are three types of SRAM banks on the Maestro: L1/L2 cache memory, DDR memory shim memory, and Ethernet shim memory. Error Detection and Correction (EDAC) was added to all three types of memory. Single-bit Error Correction/Double-bit Error Detection (SEC/DED) is the primary method of protection for those memories, with some exemptions.

Also included in the Maestro design is an enhancement of the protection of the L1 cache. On the Tile64, a parity error causes a processor exemption. On the Maestro, the intention was that a parity error instead causes a cache miss, causing a re-load from L2 cache, which corrects the error.

*On-chip registers*

There are many on-chip configuration registers, memory FIFOs, and other memory cells. These are protected by Boeing RHBD standard cells. Although these storage devices are not EDAC protected in Maestro additional protection is provided by software routines that compare their values against a known good copy of their values stored in EDAC protected memory.

In addition to the Single Event Upset (SEU) resistance of the Boeing standard RHBD cells, the MAESTRO includes additional mitigation for Single Event Transients (SETs) by incorporating temporal filtering on the data, scan and clock inputs. Depending on the timing requirements and criticality of the node, the temporal filter may be included on all, some or none of the inputs.

*External DDR*

The Maestro includes a new DDR module that is both radiation hardened and adds DDR1 functionality as well as the existing DDR2 functionality. The new DDR controller uses the existing internal data mesh shim, but adds the ability to run external Built-in-self-test on DDR memories as well as a loopback test for radiation testing without needing external memories.

## 5. FAULT TOLERANCE DISCUSSION

This paper is most concerned with the fault tolerance issues of a potential enhancement of the Maestro architecture, henceforth referred to as "Maestro-enhanced." Three approaches for implementing fault-tolerance on the enhanced processor are outlined for purposes of discussion and to illuminate possible tradeoffs. The baseline starting point is the Tilera/Boeing-Maestro architecture; the suggested three approaches propose adding software and hardware to achieve improved fault tolerance. In order that the fault tolerance enhancements be implementable with minimal changes to the design, and thus minimal cost and risk, this paper focuses on "low hanging fruit", i.e., relatively straightforward approaches to improving reliability through easily attainable fault tolerance improvements requiring minimal modification of the hardware, system software and associated development tools. The suggested approaches represent incremental changes of increasing complexity for increased fault tolerance – hopefully exposing what can be done, what is gained, and how much it costs.

*Maestro – The Starting Point*

From a fault tolerance standpoint, Maestro has greatly reduced the Tile64 Single Event Upset (SEU) rate by Radiation Hardened By Design (RHBD) circuits in its logic design and by adding error codes in cache memories. The SEU rate that has been achieved has not yet been fully determined, but it is expected that the SEU error rate of a processor core or other subsystem of equivalent complexity has been reduced to somewhere in the neighborhood of typical radiation-hardened uniprocessors previously used in space applications. The overall Maestro chip SEU error rate is probably higher than earlier single radiation hardened spacecraft uniprocessors due to its higher overall complexity of 49 cores and larger physical size. The permanent fault rate from reliability wear-out mechanisms is also probably higher than earlier chips, but there is sufficient circuit redundancy on chip to recover from many of the permanent faults – at a somewhat reduced level of performance. Permanent fault recovery algorithms (on-chip redundant workarounds) are still being developed for the Maestro processor.

Although error correcting codes have been used in memories and caches, most of the rest of the Maestro chip (49 CPUs, five intercommunication networks, DMA controllers, finite state machines, etc.) have no hardware redundancy to allow local error detection or correction, since its design philosophy depends upon the RHBD circuit design to reduce SEUs to a negligible rate. The SEU rate reduction techniques may not be sufficient, so we look for adding additional fault-tolerance in Maestro-enhanced.

Before discussing possible fault-tolerance alternatives it is useful to review a selected set of typical spacecraft dependability requirements.

*Typical use cases for on board processing*

Some of the possible uses of processing on spacecraft are as follows, from least critical to most critical:

*On board instrument processing (aka payload processing)*—This use is typically has the lowest reliability and fault tolerance requirement set, but may have the

maximum requirement on throughput and power vs. performance ratios. It involves non-real time processing of instrument data for analysis or downlink. It does not perform the main Command and Data Handling (C&DH) processing function. The aim is to process at the best of its ability, detect and report any uncorrected errors, and solely operate as a subsystem support role. Lifetime use is often quite lengthy, i.e., the sensor may be actively used for long periods of mission time, thus the associated processor has a concomitantly long mission life requirement.

*On board navigation sensor processing*—This function has an increased performance requirement. This use typically involves processing of navigation sensors such as inertial units, radar sensors, LIDARs and navigation cameras. Typically this function adds a tighter real-time processing requirement and higher reliability, fault tolerance and error handling ability. The processing must be fast enough for the navigation control loop, be reliable during the performance period, and report correct results, or report when a result is suspect. Depending on specific mission use, this function may have a relatively short lifetime, e.g., minutes to days for an entry descent and landing operation, or it may have a significantly longer life time, e.g., years, for rover autonomous navigation operations.

*On board robotic C&DH*—This function typically has a lower processing throughput requirement, but still retains the real-time requirements, and adds greatly to the availability requirement. This function manages spacecraft health and operation. Availability requirements will dictate whether a single processor or multiple redundant processors are required. In future systems, it is expected that this function will grow in required throughput as model based onboard mission and spacecraft health management technologies are matured and incorporated into next generation missions.

*Human Life Critical Systems*—This is the most stringent of all use cases with respect to reliability and fault tolerance. Any function that affects critical aspects of a human flight will add heavy requirements for availability and reliability. Single processor solutions, regardless of the reliability, performance, or dependability of the device will never be sufficient. Multiple levels of redundancy will be required at the system level in a fault tolerance hierarchy. Counter intuitively, because these systems require this high level of redundancy regardless of the device level capability, a somewhat less reliable computing component may be utilized. In other words, inasmuch as the system must pay a penalty in mass, power, and volume anyway, less reliable parts can be used! This approach has been used to significant advantage in the Constellation Orion CEV design, where commercial (non-radiation hardened) components are used with massive redundancy to provide high performance computing in human life critical systems.

*General Requirements for Use on Spacecraft*

*Typical*

Most of these requirements are specified in rates in order to make them more relevant to potential users and to reflect the effects of RHBD already employed in Maestro.

(1) Maestro-enhanced shall recover from errors and faults autonomously –without intervention from the ground. Mean time between failure to recover events, $\lambda_{fra}$ , might be specified e.g., values of 5 to >100 years depending upon the criticality of the application and the availability of ground support.

(2) The rate of incorrect outputs shall be less than one every $N_{ce}$ years. (This variable may range from 0.5 to >100 for different criticality of applications.)

(3) Computational Integrity is the probability that when a computation is recovered after an error or fault that it can be correctly completed, i.e., no computations were lost and the end state of the computation is correct. This is the opposite of situations where a computer must be rebooted in order to recover. A recovery without computational integrity may occur when state data is lost through latent errors, input buffer overflows during recovery due to excessive recovery delays, etc.

(4) The availability is $N_{av,}$ the probability that the system is "up" and ready to use at any given time. Typical values of $N_{av}$ should be in the range of 0.9995 or higher.

More Stringent Requirements for Real Time Control Systems for Unmanned Spacecraft

(5) Typically real-time applications must be synchronized, and worst case timings must deliver correct and timely results in the presence of all worst-case: i) behavior of programs running, ii) data traffic in the shared communication system, and iii) any reasonably expected error/fault recovery being conducted in the system.

(6) Recovery time from errors must be bounded. Typically critical hard real time programs must be run redundantly in order to "operate through" errors.

(7) These redundant copies must be run in fault-containment regions to prevent correlated errors/faults from disabling their function.

(8) Reliability is the probability that correct operation continues throughout the mission. Typical required values are > 0.9995.

Human Life Critical Systems cannot be implemented without redundant copies of Maestro-enhanced with independent power supplies, and I/O because there is always

the risk of catastrophic failure of a single chip and its supporting circuits. Severe requirements on containment regions and execution of redundant computation in these regions MUST be maintained. However a single Maestro-enhanced may be sufficiently reliable for low-cost unmanned spacecraft if some modifications are made. This is especially true for unmanned rovers.

*Key Implications of two typical spacecraft requirements*

(1) In light of general requirement (1) for autonomous on-board recovery a considerable amount of software and possibly some support hardware will need to be developed for Maestro-enhanced to implement this recovery process.

(2) In order to meet acceptable levels of general requirement (2) for correct outputs**,** additional error detection must be implemented in hardware and software. It is not sufficient to use the standard OS and timeout checks that come with Maestro because of their low coverage and long latency. Erroneous outputs will occur before an error is detected. Therefore SIFT (Software Implemented Fault Tolerance) is needed to provide acceptable error detection and autonomous recovery as described in the next section.

(3) To provide high computational integrity, it is necessary to prevent hypervisor state and system control tables from being damaged by a single error or fault. These situations are likely to result in a system re-boot and loss of both state and computation results as well as considerable delay. This in turn may require more complex and expensive spacecraft designs to compensate for these situations. *This may imply replicating execution of hypervisor functions and redundancy of critical state tables.*

(4) For more critical real-time control functions, it will be necessary to provide time synchronization and run multiple voted copies. Here it is important to minimize the probability of a single error or fault disrupting more than one copy at a time. This implies reducing the probability of common failures with added hardware support to prevent violating virtual memory protection boundaries and minimizing the probability of common failures by splitting up the chip into fault containment quadrants.

*SIFT – Software Implemented Fault Tolerance (necessary for Implication 1,2,3 above.)*

Software Implemented Fault Tolerance goes back over forty years with perhaps the first SIFT system being implemented for uniprocessors at SRI under NASA sponsorship. About a decade ago, these types of systems were implemented to provide fault tolerance in computer clusters. One of the early ones was Chameleon at the University of Illinois [4]. A software system, named "Ghidra" has been developed at UCLA and partially funded by NASA for multicomputers that provides a framework for Software Implement Fault Tolerance (SIFT) in multicomputer clusters [5,6]. It is an example of a software architecture that should be adaptable to the Maestro or Maestro-enhanced multicore processors in a fairly straightforward fashion and at a relatively low cost. Three processors run three identical copies of a triplicated Control and Fault Management (CFM) program in different computers that is responsible for scheduling, collecting error check results, and activating error/fault recovery in applications running on themselves and other computers in a cluster. The CFM controls software agents in all of the processors that do scheduling, timeout checking and error collection locally and report to the three CFM processors. The agents circumvent failure of a CFM replica by voting. The CFM is capable of scheduling simplex, duplex or triplicated application processes in different processors. These application processes do comparison, or voting of other redundant copies or checking of simplex ABFT-checked processes and send the results via the agents to the CFM.

This type of software can be used in multicore processors if protection of address spaces can be provided and unexpected interactions between cores is suppressed. If not, then additional system re-boots and an external recovery agent may be necessary for a degraded level of fault tolerance.

The three suggested fault-tolerant approaches for Maestro-enhanced are outlined below:

*1. Basic Maestro-enhanced FT*

a) SIFT Software is added, and three tiles serve as the CFM that manages the cores, collects error messages, and restarts failed applications. It should be noted that in the Ghidra system, these three cores can be arbitrarily chosen and can "float" amongst the processor array. Also note that the CFM functionality is relatively light weight, occupying a relatively small number of cycles, thus these cores need not be dedicated to the CMF and CMF cores may execute other codes, including applications.

b) The three CFM tiles send heartbeats to an external hardened restart state machine where they are voted. If two CFM tiles fail to deliver an OK heartbeat, the Restart sequencer re-boots the whole system.

c) Error checking and reporting is done by applications that either run simplex with ABFT acceptance test checking, or in Replicated Mode (two or more copies are executed and results compared)

The SIFT software was developed for multicomputers under the assumptions that each computer could be viewed as a fault containment region. It assumes reliable message passing that uses source coding for message authentication

to assure that messages are correct and came from the right place to maintain interactive consistency.

In a shared memory multicore machine like Maestro, there are many opportunities for single errors to affect more than one processor, and circumvent the software barriers for error propagation provided by messages (e.g., a transient on the common clock, or errors that violate virtual memory boundaries, or errors in cache coherency).

In Maestro, single bit errors can do major computational damage. The mesh networks are not protected against errors (i.e., parity), allowing wrong data to be sent to memory, and destination addresses are not protected, allowing reads and writes to wrong places – including into other virtual memory spaces. The source tile address is not included in packets, making authentication impossible. The TLB and many DMA controllers are similarly not protected, allowing violation of virtual address boundaries.

Although the latches and flip flops are protected by RHBD standard cells, there are tens of thousands of flip-flops in the Maestro mesh network, thousands of latches and flip-flops in DMA controllers and at least hundreds of latches in state machine controllers that are not protected by secondary data integrity checks. All of these can lead to unexpected error propagation or violation of virtual memory protection boundaries.

*So What Does This Mean?*

If SIFT is implemented on the existing Maestro, it can greatly reduce the number of incorrect outputs, but there will be a significant number of error cases where state recovery cannot be achieved and the system must be reloaded and re-booted, and the loss of computations may require ground intervention. It may even hang under some circumstances.

By i) improving the error detection capabilities, ii) providing better capabilities for implementation an underlying reliable communications layer, iii) providing greatly improved fault isolation and reducing the probability of catastrophic single failure, and iv) providing a better mechanism for isolating permanent faults and quickly and automatically working around them, the reliability, and overall error recovery time can be improved.

Maestro is only partially fault-tolerant, so it is important to ask the question "can additional fault tolerance be added to a future Maestro-Enhanced design that will make a machine that is more suited to critical applications?" Some suggested fault tolerance improvements, derived from the discussion above, are listed below:

(1) Add error detection (e.g. parity) to all word buffers of switches to detect address and data errors in packets and reduce error-induced violations of virtual spaces.

(2) Provide traceability in the switch network to see where an error occurred to allow location of intermittent faults.

(3) Add error detection (e.g., parity) to Registers in DMA controllers and to the Translation Lookaside buffer to prevent virtual address boundary violations.

(4) Use Error Correcting Code in the write back second level cache to allow error correction of level 1 and level 2 cache errors without interrupting the processor or requiring a program rollback.

(5) The source ID should be included in all packets to allow authentication and creation of a reliable communications layer for SIFT.

(6) Interleave memory scrubbing rather than requiring a processing halt when this occurs.

(7) Augment the routing algorithm to i) allow packets to be re-routed through an intermediate destination to work around failed tiles and links and not isolate communications between tiles, ii) to allow a tile to send a message redundantly through disjoint paths

(8) To prevent error propagation, partition tiles into regions that memory writes cannot cross – reads and message passing are unrestricted. Partition voted real-time processes across regions.

(9) Augment the chip design to minimize the probability of a single fault from disabling the whole chip, or a transient error from affecting multiple sites.

Adding parity protection to the TLB, to the networks, and to DMA controllers greatly reduces the probability that an error will cause violation of virtual address protection boundaries. This prevents the accumulation of latent errors that often make recovery difficult and can bring down the CFM, thus requiring a reboot.[3]

In order to recover from an intermittent fault in the switch network, it is necessary to identify where errors occur and keep track of whether certain errors in certain switches are occurring frequently in order to isolate and work around the faulty hardware. By adding error detection to internal packets, it is possible to trace to the switch where an error occurred.

The suggested additional packet routing mode allows a tile to use a physically different path in order to work around permanent tile or link failures. This allows some computing to continue after such a fault has occurred and thus the

[3] Some packet coding could be done in software, as could addition of a source ID. However the two internal networks are hardware controlled so a hardware fix is necessary. Adding parity on words in a uniform fashion is advantageous for all networks, and would be faster than requiring extra software for each message. Message re-routing requires hardware to be efficient.

possibility of continuing after a permanent fault without bringing down and reprogramming the Maestro-enhanced.

The suggested changes above will improve error coverage by detecting many errors sooner than they would be detected otherwise by software (ABFT/Replicated.), make recovery actions more effective, and make the SIFT more stable.

This approach greatly reduces the probability of errors causing violation of physical and virtual boundaries, but when a core runs the hypervisor, the hypervisor is unprotected. Therefore it is possible for an error that occurs (when the hypervisor is running in unprotected mode) to set up wrong TLB pointers or to damage the system tables on which all tiles and I/O devices depend. Additional work is needed on protecting the hypervisor.

*Maestro-enhanced for Real Time and Protection of Control and Fault Management*

The idea (see 8 above) of separating the tiles into regions where memory writes cannot cross a region partition is especially useful in implementing triplicated and voted processes such as real-time applications, Control and Fault Management (CFM), and possibly protecting the hypervisor. There real time processes are block redundant with each copy implemented in a different region. This prevents a processing error from affecting more than one copy.

This hardwall can be implemented dynamically under control of the hypervisor or a Ghidra type fault tolerant cluster manager middleware, and/or it may be hard-designed into the basic architecture. A potentially advantageous Maestro partitioning, for example, would be to partition the array into four quadrants with each quadrant 'owning' a memory interface a gigabit Ethernet port and a XAUI port.

This goes as far as possible in matching, from a fault tolerance standpoint, the multicomputer environment for which SIFT was designed. CFM replicas are separated into fault containment regions where erroneous software (including the hypervisor) cannot damage other quadrants. Message authentication is made possible by the changes in (2) Augmented Maestro-enhanced above.

The philosophy here is to make a relatively inexpensive change to the architecture that makes stronger fault-containment domains to allow real time control and to improve dependability of the CFM – which is at the heart of SIFT.

Since the tiles, memory, and I/O of Maestro-enhanced would share common logic for clocks and overall chip control, as well as a common piece of silicon, package, board and power supplies, there are still single faults that can completely disable the system. The design of such a system should strongly protect these circuits by use of extended RHBD and hardware implemented fault tolerant techniques to minimize the probability of those single point failures occurring

## 6. CONCLUSION

The Boeing Maestro development has demonstrated a low cost methodology of radiation hardening a commercial many-core processor. The SEU rate that has been achieved has not yet been fully determined, but preliminary results indicate that the SEU error rate of a processor core or other subsystem of equivalent complexity has been reduced to somewhere in the neighborhood of typical radiation-hardened uniprocessors previously used in space. It should enable very high performance on-board processing for a number of upcoming space missions, and it may have a revolutionary impact.

The success of the Maestro development contrasted with the discussion in this paper also points out an essential conundrum in achieving fault-tolerance in these complex technologies. To put them into space at all requires building upon the intellectual property, expertise, and tens, if not hundreds of millions of dollars of investment of private industry for high volume commercial applications. For their market, dependability is valued, but fault tolerance must be traded off against performance. The degree to which their designs can be modified, by an outside agent that translates them to radiation-hardened chips, is limited by cost, computer architecture expertise, and knowledge of the original design.

If this is done, it is apparent that a modern, high performance computer, providing orders of magnitude improvement in spacecraft onboard computing can be developed at a relatively low cost (1s to low 10s of millions of dollars), providing concomitant improvements in spacecraft performance, mission science return, and reduced mission cost and risk across a broad range of science, exploration, defense and commercial applications.

## ACKNOWLEDGEMENTS

The work described in this publication was performed at the Jet Propulsion Laboratory, California Institute of Technology, under contract from the National Aeronautics and Space Administration. Portions of this work were also performed by the Boeing Corporation under the OPERA program.

## REFERENCES


[1] Wentzlaff, D.; Griffin, P.; Hoffmann, H.; Liewei Bao; Edwards, B.; Ramey, C.; Mattina, M.; Chyi-Chang Miao; Brown, J.F.; Agarwal, A.;. "On-Chip Interconnection Architecture of the Tile Processor", IEEE Micro, September-October 2007, pp. 15-31.

[2] Saurabh Dighe, et. al, "Lessons Learned From The 80-Core Tera-Scale Research Processor," Intel® Technology Journal | Volume 13, Issue 4, 2009

[3] Malone, Michael, "OPERA RHBD Multi-core" Presentation at Military and Aerospace Programmable Logic Devices (MAPLD). 2009

[4] Z. Kalbarczyk, R. K.Iyer, S. Bagchi, and K. Whisnant, ''Chameleon: A Software Infrastructure for Adaptive Fault Tolerance,'' IEEE Transactions on Parallel and Distributed Systems, vol.10, no.6, pp.560-579 (June 1999)

[5] Daniel Goldberg, Ming Li, Wenchao Tao, and Yuval Tamir, "The Design and Implementation of a Fault-Tolerant Cluster Manager," Computer Science Department Technical Report CSD-010040, University of California, Los Angeles, CA (October 2001).

[6] Ming Li, Fault-Tolerant Cluster Management, Ph.D..Dissertation, University of California at Los Angeles, 2006.


## BIOGRAPHY

***Carlos Y. Villalpando*** *is a Senior Member of Technical staff in the Advanced Computer Systems and Technologies group at the Jet Propulsion Laboratory. He is currently a digital designer for advanced computing techniques for machine vision applications in FPGAs as well as system designer and programmer for machine vision tasks on multicore processors. He earned his Bachelor of Science degree in Electrical Engineering, Computer Block at the University of Texas at Austin in 1996 and a Master of Science in Electrical Engineering-VLSI at the University of Southern California in 2003. He has been a member of the JPL community continuously since 1993 and has worked primarily on Technology development tasks.*

***Raphael Some*** *manages the AAPS High Performance Computing task at JPL. His other duties at JPL include: avionics technologist for the New Millennium Program; leader of the Technology Review Board for the ST8 Dependable Multiprocessor project; and leader of JPL's Advanced Avionics Research and Technology Initiative. Previously at JPL, Raphael was the Chief Engineer for the Remote Exploration and Experimentation Project and Principle Investigator of the Smart Sensor Web technology development 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.*

***Dr. David Rennels*** *received the B.S.E.E. degree from Rose Hulman Institute of Technology in 1964, the M.S.E.E. from Caltech in 1965, and the Ph.D. in Computer Science from UCLA in 1973. He has been principal investigator of research projects in fault-tolerant computing sponsored by the Aerospace Corporation, NSF, ONR, Hughes Aircraft, and TRW at UCLA. He has also been responsible for the design and experimental validation of several fault- tolerant computers at the Jet Propulsion Laboratory. Dr. Rennels was the general chairman of the 12th International Symposium on Fault-Tolerant Computing in 1982. He is a member of the IFIP Working Group 10.4 on Reliable Computing and Fault-Tolerance, and was chairman of the IEEE Computer Society Technical Committee on Fault-Tolerant Computing in 1988 and 1989. His research interests are computer architecture and fault tolerant computing. Professor Rennels was Vice-Chair of Undergraduate Affairs for the Computer Science Department.*