Enabling Future Robotic Missions with Multicore Processors

Wesley A. Powell[1], Michael A. Johnson[2], and Jonathan Wilmot[3]
NASA Goddard Space Flight Center, Greenbelt, MD 20771

Raphael Some[4], Kim P. Gostelow[5], Glenn Reeves[6], and Richard J. Doyle[7]
Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA 91109

Recent commercial developments in multicore processors (e.g. Tilera, Clearspeed, HyperX) have provided an option for high performance embedded computing that rivals the performance attainable with FPGA-based reconfigurable computing architectures. Furthermore, these processors offer more straightforward and streamlined application development by allowing the use of conventional programming languages and software tools in lieu of hardware design languages such as VHDL and Verilog. With these advantages, multicore processors can significantly enhance the capabilities of future robotic space missions. This paper will discuss these benefits, along with onboard processing applications where multicore processing can offer advantages over existing or competing approaches. This paper will also discuss the key artchitecural features of current commercial multicore processors. In comparison to the current art, the features and advancements necessary for spaceflight multicore processors will be identified. These include power reduction, radiation hardening, inherent fault tolerance, and support for common spacecraft bus interfaces. Lastly, this paper will explore how multicore processors might evolve with advances in electronics technology and how avionics architectures might evolve once multicore processors are inserted into NASA robotic spacecraft.

I. Introduction

Robotic spacecraft, regardless of mission, generally conform to a generic architecture, which can be divided into the spacecraft infrastructure and the spacecraft payload. As illustrated in Fig. 1, the spacecraft "bus" provides the basic infrastructure of the spacecraft and consists of a mechanical structure, a power generation and distribution subsystem, a propulsion and attitude control subsystem (including guidance/navigation sensors), a radio communication subsystem, and a command and data handling subsystem (including the flight control computer, memory storage, and data acquisition for a suite of "housekeeping" sensors). Attached to this bus infrastructure is the payload, consisting of the science and exploration instruments which are the spacecraft's raison d'etre.

In this paper, we examine the spacecraft computing system, which, to date, on most spacecraft, is a relatively low performance, but extremely high reliability computer. Its task, for the most part, has been to execute carefully crafted sequences provided by a team of experts on the ground that is responsible for mapping out the spacecraft's minute-to-minute activities and uploading these sequences on a periodic basis via the radio communication system. This paradigm, however, is extremely costly, limiting in mission capabilities, and greatly reduces the potential science and exploration return on mission investment. In most cases, what can be done with the limited computing resources available in current spacecraft has been done, and we are reaching the limit of mission complexity and spacecraft capability achievable with standard spacecraft control computer technology.

---

[1] Assisitant Chief for Technology, Electrical Engineering Division, Code 560.
[2] Chief Technologist, Applied Engineering and Technology Directorate, Code 500.
[3] Software Engineer, Flight Software Systems Branch, Code 582.
[4] Technologist, Avionics Section, M/S 3450.
[5] Flight Software Engineer, Software Guidance & Control and Flight Software Validation, M/S 321-150.
[6] Chief Engineer, System and Software Division, M/S 301-230.
[7] Deputy Manager, Solar System Exploration Technology Office, Solar System Exploration Directorate, M/S 321-550, and AIAA Associate Fellow.

Demands for onboard computing will be significantly increased for many future National Aeronautics and Space Administration (NASA) robotic space missions. This is largely driven by the fact that increases in instrument sensor data rates are not being matched by increases in downlink bandwidth. In addition, future mission concepts call for autonomous decision making which further drives onboard processing needs.

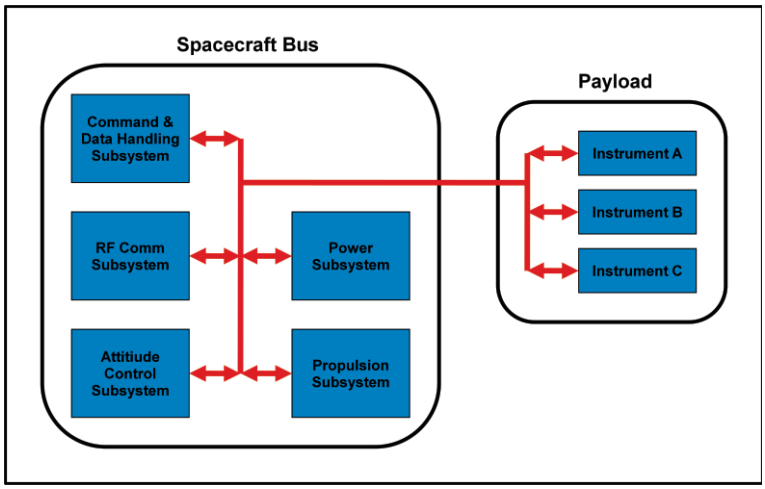

Figure 1. Robotic Spacecraft Architecture.

Several options currently exist for onboard computing, offering varying levels of performance. Radiation hardened spaceflight processors such as the BAE RAD750 are based on PowerPC processors and offer performance up to 240 MIPS[1] (million instructions per second) range. Other vendors offer multiple unhardened PowerPC processors with various redundancy schemes, which offer increased performance at the expense of power and programming complexity. For applications requiring less performance and lower power, processors embedded with a System-On-a-Chip (SOC) are a viable approach. With this approach, moderate performance processors such as LEON and ARM variants, or lower performance processors such as MISC, can be implemented either in radiation hardened application specific integrated circuits (ASICs) or field proigrammable gate arrays (FPGAs). Applications for these processors are typically coded with high-level software languages, and with the exception of the lower performance processors operate on real-time operating systems such as VxWorks or RTEMS (Real-Time Executive for Multiprocessor Systems).

For high performance onboard digital signal processing applications, processing can be implemented directly within the logic of a radiation hardened FPGA such as the Actel RTAX series or the Xilinx Virtex-5QV or, alternatively, an ASIC. With this approach, performance in excess of 300 GOPS (billion operations per second) is possible on a single FPGA device[2]. While several high-level tools exist that allow rapid modeling of these applications, the actual coding is typically done using a hardware description language such as VHDL (Very High Speed Integrated Circuit Hardware Description Language) or Verilog.

In general, across all processing options, there is a trend where one can trade performance for application development complexity. Current radiation hardened spaceflight processors are limited in performance, but offer a relatively simple application development process using common software programming languages and operating systems. Radiation tolerant processors in redundant configuration offer higher performance and can use these tools and operating systems, but require added complexity to handle voting mismatches and resynchronization. FPGA or ASIC based computing provides very high performance, but at the cost of a complex application development process using hardware description languages.

A radiation hardened multicore processor is an attractive alternative to these approaches that could provide increased performance for future processing requirements and allow application development using standard software languages and tools. With the emergence of commercial multicore architectures and radiation hardened integrated circuit design libraries, such a device is now feasible.

Commercial industry adopted multicore processor architectures after it proved difficult to push operating frequencies above a few Gigahertz. To circumvent this limitation, these processors use multiple cores operating at a lower frequency. Multicore architectures should be viable well into the future as improvements in semiconductor fabrication technologies (reducing feature sizes) will allow an increasing number of processing cores to be implemented on a single device. As a point of reference, the Tilera TilePro64 multicore processor offers a theoretical maximum of 384 GOPS[3] (billion operations per second). However, multicore processors do present some challenges as developing software that can fully utilize the processing resources can be difficult. There is, at present, a lack of parallelizing compiler support and tools that can optimally allocate threads across multiple processors. Furthermore, the achieved processing performance can be very sensitive to memory locality as off-chip memory access can incur significant performance penalties.

II. Classes of Multicore Applications

Multicore processors are well suited for applications that allow the computation to be distributed among multiple processing nodes. We call out one category – general spacecraft computing – and three special application classes— Short-Duration Real-Time Burst Calculations, High-throughput Science Data Processing, and Intensive Search-Based Reasoning— that seem particularly amenable to future multicore processor-based spaceflight systems. The latter three also seem well-suited for implementation via other processing technologies (e.g. FPGAs). Characteristics of these application classes are documented in Table 1. The following discussion identifies examples of spaceflight systems within each application class.

Table 1. Classes of computation for spaceflight multicore applications.

Class of Application	Mission Applications	Objective of Computation	Flight Computing Concept
Short-Duration Real-Time Burst Calculations	Science/Exploration: Entry, Descent & Landing, non-cooperative Autonomous Rendezvous and Docking, real time reaction to internal or external science or engineering stimuli	Achieve most robust results within available time constraints as input to control decisions	High peak power needs; significant margin for remainder of mission; stringent fault tolerance and real-time requirements
High-throughput Science Data Processing	Science: High resolution sensors, e.g., Synthetic Aperture Radar (SAR), Hyperspectral	Downlink images and products rather than raw data	Distributed, dedicated processors at sensors; perhaps less stringent fault tolerance
Intensive Search-Based Reasoning (may be Non Real-Time)	Science/Exploration: Mission planning, fault management, model-based reasoning	Accomplish opportunistic science; mitigate execution failures via contingency planning; detect, diagnose and recover from faults and unanticipated events	Multicore may be needed to enable onboard capabilities due to computational demands
General Spacecraft Computing	Core spacecraft functionality: Telecom; file system; commanding and sequencing; attitude control; …	All the functions required to run a spacecraft.	Utilize the capacity offered by multicore; message-passing and partitioning to achieve fault containment regions. Redundancy in cores allows for fault recovery, multiple algorithms to achieve the same result.

A. Short-Duration Real-Time Burst Calculations

Gamma-ray bursts (GRBs) are intense flashes of gamma rays that occur several times daily and typically last for only a few seconds. The Swift spacecraft is dedicated to the study of gamma-ray burst science and exemplifies a scenario that requires autonomous detection of, and real-time reaction to, external (or internal) stimuli.

As illustrated in Fig. 2, Swift is comprised of three instruments, each operating in different wavelengths— the Burst Alert Telescope (BAT), the X-ray Telescope (XRT), and the Ultraviolet/Optical Telescope (UVOT). Once the wide field-of-view detectors on BAT detect a GRB, the spacecraft must be repointed to align the other two narrower field-of-view telescopes with the event. Short GRB durations and their unpredictable times of occurrence render a human-in-the-loop architecture ineffectual. Instead, a high-throughput onboard data processing system determines

the location of the burst and autonomously reorients the spacecraft to bring the burst area into the XRT and UVOT fields-of-view[4].

Depicted in Fig. 2, entry, descent, and landing (EDL) is another autonomous spaceflight capability that requires significant burst processing capability. Mission success hangs in the balance of a successful EDL sequence, and for destinations such as Mars, light-time delay forces the question of autonomy. Increasingly, EDL algorithms are vision-based algorithms that involve, e.g., identifying and tracking terrain features to avoid hazards and achieving pinpoint landing accuracy. Future proximity operations concepts for primitive bodies missions also are expected to make heavy use of vision-based algorithms. As a rule, these algorithms are computationally intensive and push beyond a general-purpose processor level of capability. The Autonomous Landing and Hazard Avoidance Technology (ALHAT) project within the Exploration Technology Development and Demonstration Program (ETDDP) has benchmarked a set of Lidar-based algorithms for a lunar lander as requiring approximately two more orders of magnitude flight computing capacity than had been baselined.

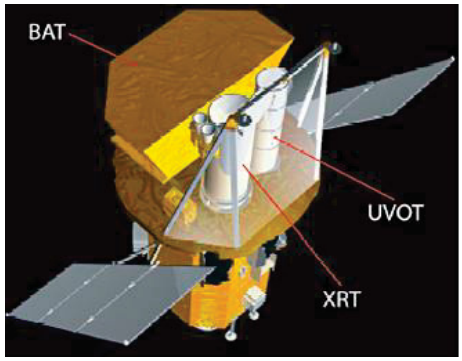

Figure 2. Swift Spacecraft.

Applications in the burst calculation class are typically mission-critical. To the extent they are performing calculations within a limited timeframe as input to control decisions, they have the most stringent fault tolerance requirements.

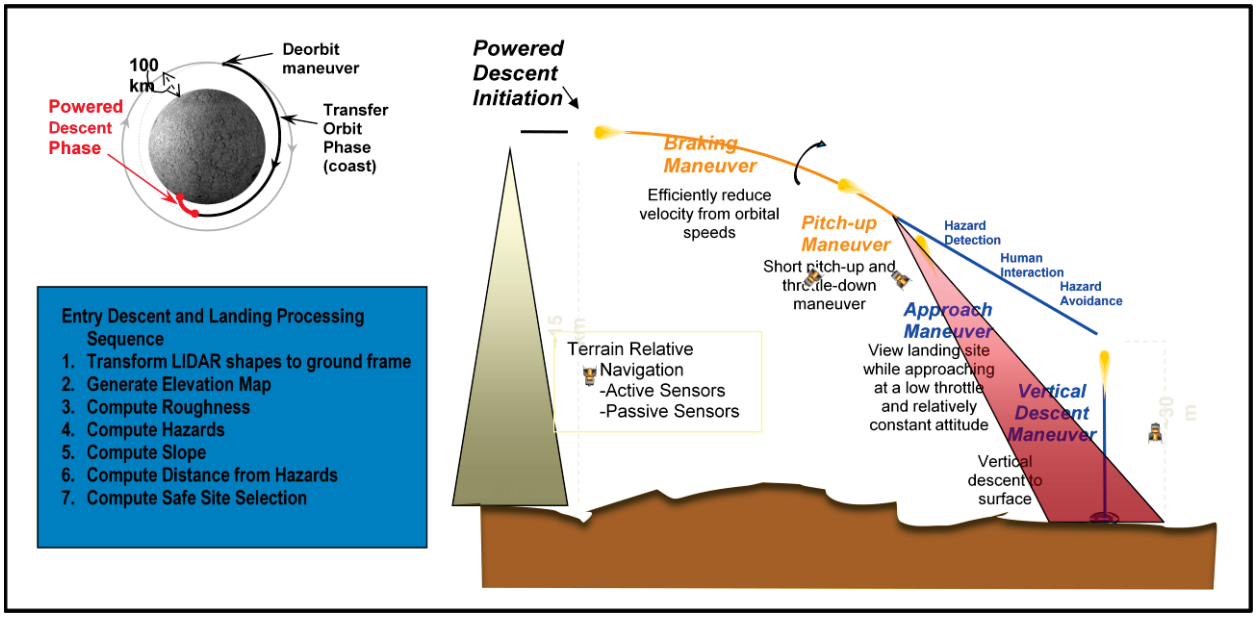

Figure 3. Entry, Descent, and Landing Concept of Operations.

B. High-Throughput Science Data Processing

Long-standing race conditions exist between capabilities for sensors and instruments to collect data and capabilities to efficiently communicate and meaningfully process that data. The throughput of modern science instruments can be astounding. The HyspIRI (Hyperspectral Infrared Imager) instrument is projected to collect 3.2 terabytes per day while the DESDynI (Deformation, Ecosystem Structure, and Dynamics of Ice) synthetic aperture radar (SAR) instrument is projected to collect 4.9 terabytes per day, to cite just two examples. Such high data throughput can challenge downlink capabilities even for Earth orbiters. However, multicore offers the possibility to trade downlink capacity with onboard computing capacity; specifically, to generate appropriate onboard science data products (e.g., images, advance statistical summaries of raw data) with smaller data footprints that can fit within downlink limitations. Figure 4 depicts a notional onboard processing flow for hyperspectral image data that could

be implemented via multicore computing. Multicore support for this application class would likely involve decentralized, dedicated processing at the instrument, with less stringent fault tolerance requirements than for core spacecraft functions. To summarize, multicore opens up the trade space to offer more flexibility in finding suitable system solutions among flight computing, science instrument data throughput, and downlink capabilities.

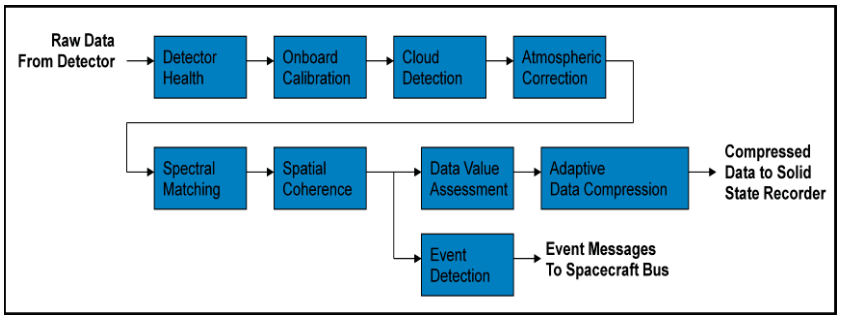

Figure 4. Hyperspectral Image Processing

C. Intensive Search-Based Reasoning

Autonomous functions on future space missions can be expected to become more common and necessary, especially for deep space missions, as flight systems venture further into remote environments that are at least partly unknown. Multicore flight computing will have a role to play to support certain onboard model-based, search-intensive reasoning capabilities such as planning, scheduling and resource management, as well as fault management. As an example, multicore can increase the very pace of a mission by improving the so-called thinking-to-driving ratio of Mars rovers. Currently, for *Spirit* and *Opportunity*, that ratio has been approximately 15:1. In other words, during mobility periods, the Mars Exploration Rovers are "thinking" much more than they are moving. With multicore, that ratio can approach 1:1, or ideally, the rover "thinks" while it moves. Such onboard computing capacity has implications for science: processing navigation images for interesting science signatures while traversing; and for fault management: more frequent collection and processing of navigation images for hazards or fault conditions. More generally, future flight systems will increasingly have need for fail-operational fault management, rather than traditional fail-safing, to grapple with challenging operational environments and/or severe mission duration constraints. Multicore can support the implied computations..

D. General Spacecraft Computing

This class of software currently comprises the majority of software on a spacecraft and includes functions such as telecom, resource and device control, attitude control, power switching, file system management, commanding and sequencing, and a myriad of other activities necessary to keeping a spacecraft operating. In many instances, these activities are programmed as threads that communicate and coordinate through message passing interfaces, allowing them to function easily in a multicore system. Among the non-real-time applications, we should expect several would share a given core. But of special note are the real-time applications, such as attitude control, which require predictable, rate-group style organization with strict adherence to deadlines. Elsewhere in this paper we refer to *partitions* that may be established within a multicore chip, and certainly a real-time partition is required if we are to host attitude control and similar applications on multicore. Given a partition capability, including timely access to spacecraft sensors and actuators, and thus, a means to establish fault containment regions and memory partitions, real-time and non- real-time applications supplying the spacecraft's core functionality should operate well in a multicore environment..

E. Multicore Application Considerations

Software architectures used to instantiate specific applications will vary considerably. How well the software architecture maps to the multicore architecture directly impacts how efficiently the application will be executed. The following questions are particularly relevant to the issue of efficiency. Some of these issues are addressed in the following discussion on multicore architectural features.

1) Does the application require frequent and fast access to a large off-chip data volume or can the application largely be executed on-chip (in place)?
2) How well can the application be parallelized?
3) Are the processes distributed among the nodes largely independent or is significant inter-processor communication necessary?

4) Are these processes identical or heterogeneous?
5) Does the application require a dynamic processing (e.g. tasks and memory allocation) model?
6) How much processing latency is acceptable?
7) Is floating-point processing necessary?

III. Key Multicore Architectural Features

Commercial multicore processor architectures are proliferating at a rapid pace. Applications and processing paradigms drive architectures. As the range of processing applications and requirements expands, so does the span of architectures. In space-based systems, as in any multiprocessing system, they key elements are:

1) Processor core instruction set architecture (ISA)
2) Heterogeneity vs. Homogeneity of cores
3) Number of processor cores
4) On-chip Inter-core communication network
5) On-chip memory (cache) architecture, size, distribution and access method
6) Off-chip memory architecture and interface
7) Off-chip bus/network interconnect interface

The specific characteristics of the above key elements will determine the applicability of the processor to real-time systems, scientific parallel processing, large space searches, and model-based analytical processing. Similarly, the inherent robustness of the computer and suitability for application of different fault tolerance techniques is directly determined by these architectural features.

Many currently popular, commercially available multicore processors comprise a small number (2 to 8) of cores based on previous generation unicore processors. The primary advantage of these types of machines is that they require minimal new development both in hardware and software, thus providing a rapid path to market and a familiar software suite at the cost of potential enhancements in efficiency and throughput that might be achieved using newer ISAs and interconnects more tailored to the multicore environment. Provision of a relatively straightforward entry path to multicore computing for both developers and users has given researchers time to begin the process of understanding the potential of multicore computing and to experiment with different approaches. While most of these multicore processors are built around general-purpose ISAs, there are also examples of Digital Signal Processors (DSPs) in this category. DSP ISAs and unicore architectures, having a long history of accommodating multiprocessing architectures such as parallel and systolic arrays, are more easily adapted to the multicore environment and highly efficient multicore DSPs are readily available.

Current state of the art general-purpose multicore processors, such as the Tilera product line, offer many tens of processors with the latest generation machines providing up to 100 cores on a chip. In this case, a custom ISA was developed as well as custom on-chip interconnect. This level of customization enabled high efficiency for the target application area, (video processing) while standard off-chip memory and I/O interfaces allowed relatively straightforward board level product development. As the family moves towards more general-purpose computing, changes in ISA, interconnect, and hardware support for previously software-implemented functions are required, and thus the family architecture is evolving. Other computing system developers are making similar trades as they evolve their product lines to penetrate the growing range of applications and markets.

As users are quickly finding out, the use of multicore machines for unintended applications results in significant, and often unacceptable, losses in efficiency. We have recently experimented with a range of commercial multicore computers; porting a variety of operating systems and applications to these machines and measuring power efficiency and compute throughput and scalability while concurrently analyzing the architectures for fault tolerance capabilities and suitability for use in mission critical and extreme environment applications.

As will be discussed in the following sections, space-based applications require additional unique features at the fundamental semiconductor and circuit technology levels, as well as, at the ISA and architectural levels. Designing multicore processors for space will entail unique architectural trades to accommodate a broad range of unique and general-purpose applications a well as high reliability in extremely challenging environments.

IV. Desired features of a spaceflight multicore processor

NASA has been looking at multicore processors as a means to enable advanced mission capabilities and as a way of reducing the size, mass and power of the avionics systems. Currently, mission designers must deploy federated computing systems to provide the sufficient computational power and fault detection, isolation, and recovery (FDIR) functions. Using this approach with standalone processors has significant negative impacts to system complexity, size, weight, and power (SWaP) and increases overall mission cost. Multicore systems have the potential to consolidate computing functions, reduce the number of special purpose FPGAs and maintain fault FDIR functionality, all while dramatically improving computational performance for next generation capabilities.

During an early phase of the Altair crewed Lunar Lander development, the development team investigated deploying a fault tolerant multicore processing architecture. By using the high performance computing capabilities for general automation, vision processing, landing hazard avoidance, Automated Rendezvous and Docking (AR&D), as well as normal Guidance Navigation & Control (GN&C) and Command & Data Handling (C&DH) functions, the team was able to reduce significantly the overall SWaP and provide for future commonality with other systems within the Constellation program. And while this study focused on a crewed mission, these benefits would apply equally to robotic space missions. Currently, several vision based autonomous landing systems providing pinpoint landing, hazard recognition and avoidance, and real time navigation based on terrain analysis are under development for both human and robotic missions. In addition, there are multiple efforts under way aimed at developing autonomous science capabilities and autonomous onboard mission planners for robotic missions based on multicore processor architectures. Using these projects as a basis, the following general features have been defined to be highly desirable in future general-purpose multicore computers. It should be noted that the following paragraphs relate to general-purpose computing solutions, not digital signal processors or other types of specialized

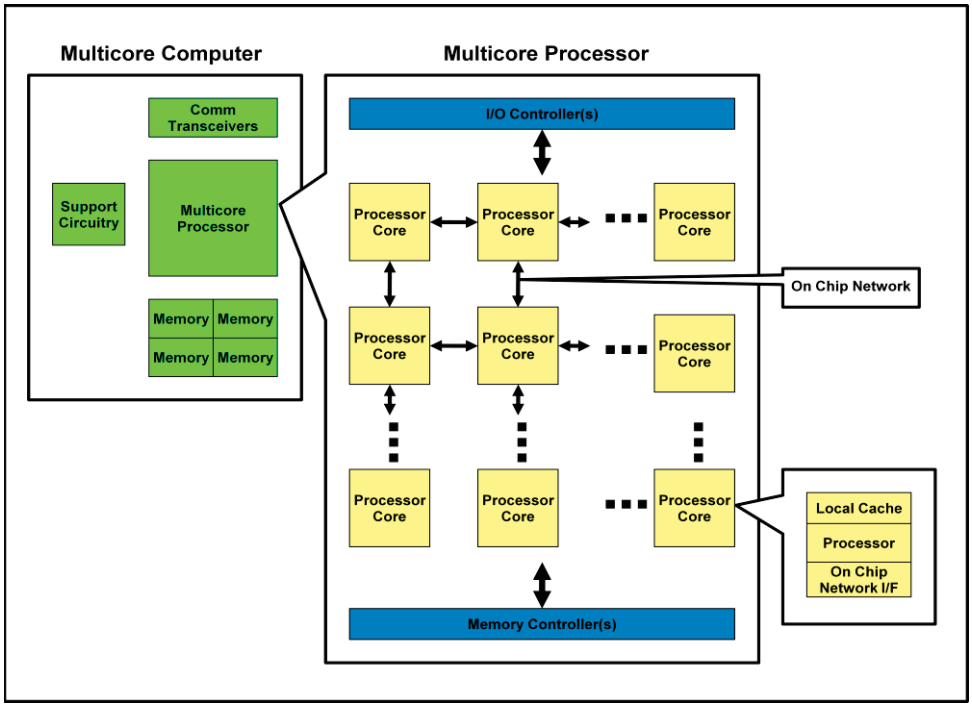

Figure 5. Notional Multicore Computing System.

computing architectures, and that the perspective of the authors is to provide guidance to computing system developers utilizing current or emerging technologies for products to be fielded in the next few (3-6) years. Figure 5 depicts a notional multicore computer system at the board, multicore processor device, and processor core levels. The following discussion explores the desired feature set by type of feature (e.g., performance, fault tolerance) architectural allocation (e.g. processor, memory, I/O), and level of implementation (e.g., processor, system).

A. General Performance

While onboard processing performance requirements can vary substantially from mission to mission, the desired features described below would accommodate most foreseeable applications.

1) Processing Throughput

The processor should be capable of providing at least 40GOPS and 20GFLOPS (billion floating-point operations per second) throughput to support these highly compute-intensive real time applications. While computing platforms of lesser capability will be of interest to some missions, this high level of computational capacity will provide the most generically useful, cross-mission capability. The computing system design should provide sufficient memory and I/O bandwidth to support this throughput capability.

2) Power Management

The processor should be capable of scaling its power and energy consumption with required throughput. For many applications, the required throughput will vary dramatically throughout the mission either due to pre-planned mission phasing or because of unexpected events that require immediate response, e.g., real time execution of a model based mission planner due to discovery of a science opportunity or potentially mission threatening hazard. Two modes of power/performance scaling should be made available: clock control and core power up and down. To accommodate real-time events, these mechanisms should respond rapidly, i.e, on the order of microseconds to milliseconds, not seconds. To provide maximum utility in extremely energy constrained missions, the adjustability should be fine grained (i.e., at the level of individual cores and a few megahertz clock rates). Typical spacecraft processors nominally dissipate approximately 5 watts while spacecraft computers will typically dissipate approximately 10 to 15 watts maximum. While somewhat higher power levels are acceptable for some earth orbiting missions, deep space missions are extremely power constrained and expected to become more so in many future missions. Thus, in addition to power scaling as discussed above, the processor design should provide a generally useful operating configuration (at least 1GOP, external memory interface and one high speed external I/O) for not more than 5 watts.

3) Processor Core Implementation

The system should, in general, be homogeneous in nature, with the possible exception of cores controlling off-chip I/O. Processor cores should all implement the same instruction set. This provides ease of programming, fault tolerance, and space qualification. While custom heterogeneous architectures can provide significantly higher efficiency, the specific application must be very well known and ubiquitous to justify special purpose heterogeneous designs. In the longer term, heterogeneous architectures may become advantageous, but at present this does not seem a winning strategy.

4) Internal Interfaces

It is extremely difficult, at this point, to define the optimum architectural tuning point of processing core throughput, memory capacity per core, inter-core network speed or topology, on or off-chip memory, and I/O bandwidth. The optimum architecture will vary dramatically with application. The driving requirement at this time is to enable support of a broad range of applications, and thus, a broad range of architectural tuning points within a given processor and computing system design. In the future, benchmark applications will need to be developed to accurately assess the adequacy of specific architectures against anticipated mission needs. At present, we can, however, state that each processor core should have sufficient local memory, implemented as either local memory or cache, to ensure that the need to access off-board memory for both typical spacecraft applications and for scientific parallel computations (such as are typically performed by cluster computers for ground base science data analysis) is minimized. Similarly, the inter-core interconnect should be sufficiently fast that the need to access off-board memory or I/O does not unduly delay processing in worst case real time scenarios, which might include multiple cores re-loading local memories with new instructions and data. Additionally, we anticipate increasing inter-core message traffic among computations spread over the chip which need to be supported by the interconnect. We expect message-passing to become increasingly important compared to shared memory for general-purpose programs for fault containment reasons as well as simplicity. Strategies such as network partitioning to minimize contention and collisions may be beneficial in some architectures. Use of emerging technologies such as through-silicon-vias and chip stacking may provide

considerable flexibility in both local core memory capacities and core interconnect topologies. By similar reasoning, it would seem that provision of multiple memory ports and multiple high speed I/Os distributed about the multicore processor chip would provide the best overall throughput and the highest probability of being able to achieve a balance of processor throughput, memory bandwidth, and I/O for a broad range of applications.

5) External Interfaces

As described above, the processor should provide high speed I/O, but it is difficult at this time to specify optimum or prioritized I/O protocols. At the computer system level, however, it is clear that multiple 10Gb/s interfaces, compatible with standard spacecraft and commercial standards, should be provided. The current standards such as 1553B, SpaceWire, and TTGbE (Time Triggered Gigabit Ethernet) will not be sufficient to support future spacecraft requirements, so that while they must be accommodated by some means, a next generation protocol at the 10Gb/s level is needed and the computer design should accommodate easy addition of new interface standards as they become available.

6) Time Management

Distribution of "system time" is also a hard requirement for space-based computing systems, both for fault tolerance and for normal operation. To accommodate this requirement, the processor should provide a low latency, deterministic mechanism for distributing system time across the array of processor cores and provide an external interface to synchronize with other local computing systems.

B. General Reliability

The system needs to be fundamentally reliable for long missions. Typical missions can last 10 years or longer, and some missions have lifetimes of 30 years or more. The processor chip and the computer system must provide long life time and high reliability in the space environment. Typical mission environments include low earth orbit, geosynchronous orbit, the lunar and Martian surfaces, and deep interplanetary space. Due to the long life requirements, device packaging, board designs, and packaging implementations, and semiconductor component design and quality are critical.

1) Packaging

Spaceflight electronics packaging requirements are far more demanding than those for most commercial off-the-shelf devices. The packaging must first be able to withstand mechanical stresses during launch, which can generally be handled by good mechanical design. Once on orbit, the thermal environment of space demands that packaging withstand wider temperature extremes and frequent thermal cycles throughout a mission, as well as, provide adequate thermal dissipation in vacuum. Even prior to launch, electronics packaging must also allow detailed inspections during board-level assembly. For large integrated circuits, such as microprocessors, these requirements typically are addressed by using hermetically sealed ceramic column grid array packages.

2) Thermal

While typical ambient or cold plate temperatures are normally within the 0^{O} to 40^{O}C range, due to the difficulty of establishing good thermal paths, junction temperatures can reach well over 100 ^{O}C. However, cases also exist (i.e. after start up) where a computer must operate near 0^{O}C. Component and board level designs must accommodate these wide temperature variations.

3) Ionizing Radiation

The ionizing radiation environment can vary dramatically from mission to mission and can even be highly variable within a single mission. To operate reliably in their intended environment, spaceflight electronics must mitigate both the long term and transient effects of this radiation. Long term effects result from total ionizing dose (TID), which can cause threshold shifts and increases in leakage current. With the possible exception of the Jovian system, the Van Allen Belts and a few other special cases, total ionizing dose has not been a

significant issue. However, transient single event effects (SEE) can manifest in many ways and can be a major contributor to the unreliability of space computing systems. Single Event Latchup (SEL) is a potentially destructive high current state and, in general, all spaceflight electronics must have immunity to this. Other SEEs include Single Event Upset (SEU), Single Event Transient (SET), and Single Event Functional Interrupt (SEFI), which are non-destructive but can cause either data errors or operational interruptions. Depending on the criticality of the application, a spaceflight computer must also be either prevent or mitigate these effects. While a variety of fault tolerance approaches have been developed to handle these errors, the components themselves should provide high levels of radiation tolerance thereby minimizing reliance on fault tolerance techniques.

4) Part Qualification

The system level reliability of spaceflight missions is significantly determined by the reliability of the individual electronic parts. To ensure the high reliability of these parts, qualification programs are required to ensure that the parts are fabricated, assembled, and tested in a controlled manner. These qualification steps call for extensive lot-level testing and inspection steps that go well beyond the standard practice for commercial fabrication processes.

C. Fault Isolation and Recovery

While reliability is preferable to an over-reliance on fault tolerance, the reality is that no component or system is immune to fault induced errors. Highly reliable systems must implement at least some fault tolerance. The fault tolerance measures should, therefore, be inherent in the design of both the processor and the computer system, while imposing minimal overhead in throughput, mass, power, and volume. It is our experience that, if fault tolerance is built into the system from the processor and components through the system level, it is possible to achieve high efficiency levels. But it is also our experience that the converse is true: if fault tolerance is not designed in at the outset, it will be impossible to achieve highly efficient, high coverage fault detection, isolation, location, and recovery.

One of the most important desired features is the ability to hardware partition functions from one another on different cores. This applies to all of the use cases discussed in section II. With the support of super/hypervisor level software, hardware partitioning would allow individual cores to be reloaded, restarted, crash, and contain software errors all without affecting other cores. Mixing science data processing software from different vendors or even loading critical Guidance Navigation & Control software into an adjacent low criticality science processing core would be possible and certifiable. This partitioning allows the software system composability (separation of concerns) benefit of a federated system to be retained while still meeting future mission needs for high performance and lower SWaP avionics. Without the ability to run-time partition cores and system resources, a multicore processer may become just a niche product and not be generally applicable to a wide range of missions.

The processor chip should be designed with fault tolerance of the processor cores and external I/O in mind. This can be done in various ways, but it should be done power efficiently and with minimal operational overhead. At a minimum, the following issues should be considered in developing a fault tolerant machine.

1) Error Detection and Correction

Basic fault tolerance mechanisms such as error detection and correction codes should be provided at the processor level, i.e., at the internal chip interconnect and on-chip memories.

2) Memory Partitioning

The system should optionally provide the capability to partition memory and I/O such that it is not possible for a core to corrupt memory or I/O that it does not “own”. The processor internally, and the computer at the system level, must be able to guarantee this partitioning in the presence of faults. However, there may also be special cases where a set of processors may need to share local memory to optimize performance.

3) Scrubbing

The processor and the system should provide some means of detecting and scrubbing latent faults or errors to preclude a buildup of undetected errors that could result in system failure due to an accumulation of errors beyond the ability of the built in fault tolerance to handle.

4) Core-to-Core Communication

The processor should be capable of fault tolerantly and efficiently enforcing a message passing interconnect paradigm. If it were possible to achieve high efficiency in a purely message passing interconnect, this would be preferred, but it is not clear that this is the case, especially for certain applications that naturally map well into a shared memory computing model. Thus, a general-purpose processor, in the near term, should provide both modes of communication but be able to guarantee message passing only (i.e., that a processing core cannot, even under fault conditions, write into another processor's memory area) within specified regions of the core array. Similarly, this mechanism should also be extended to external memory and I/O.

5) Internal Redundancy

It is often desirable (or necessary) to implement a multiply redundant fault tolerance scheme such as triple modular redundancy. To support these fault tolerance modes, the processor should provide the ability to tightly synchronize processing cores.

6) External Synchronization

It will, for some applications, be necessary to implement redundancy at the system level. Synchronization to an external timing source is therefore also desired at least at the processor level.

7) Redundant I/O

Fault tolerance at the spacecraft level requires the spacecraft computer to provide multiply redundant I/O ports. This is at least a computer system level requirement, but depending on architecture, may be required at the processor level as well.

8) External Memory

The computer memory system will require error detection and correction of computer memories. Toleration of whole memory chip failures and multiple bit errors is needed as well.

V. Software Considerations

The efficient and cost-effective development of the software that is to run future multicore systems requires tools. Of course, we still need the usual compiler-linker-loader with standard binary formats for the hardware architecture at hand. But with multicore, we have the additional issue of distributing the programs to the cores that will use them – a possible wrinkle in the loader part of the problem, or possibly addressed through some other facility that communicates with the cores. But whatever the solution, the programmer expects that getting the program into the correct memory cells and moving the code from one core to another will be done handled by functions supplied with the system. We also note that other tools that have become standard, such as language-sensitive editors and platform-aware debuggers, are also expected. These are the minimum.

In particular, the job to be done by debuggers is more complex, and more necessary, than has previously been the case. Simply stated, multicore makes possible more paths through a given collection of code than equivalent code run on a single-core, multi-tasking system. One reason is because task priorities in a single-core system often constrain the order in which a given set of tasks will be allowed to run, while with multicore, priorities among tasks hold only within a single processor. So, if two tasks happen to be running on separate cores, they may interact and their executions interleave in ways unanticipated by the programmer.

Most debugging takes place in a sequential environment. This means that the programmer wants to control, in a repeatable way, the order in which cores as well as tasks will run. The programmer will want to run, and re-run, the

same sequence of execution over and over again, to discover his bug. Debuggers are a proven time-saving tool, and in multicore, the situation is certainly no less complex.

Another important tool for code testing is simulation. Multicore simulators, both binary compatible and not, have their costs and uses. But a simulator is essential and is especially useful if it is capable (for the task at hand) of sufficiently accurate performance prediction. More and more, programmers and projects employ simulation for testing code and for testing entire systems. The cost of simulation is much lower than the cost of testing on actual hardware and has become an expectation of developers and managers alike.

Our real challenge is applications with a real-time component. Of course, a real-time operating system is essential, and with libraries for:

1) Communication Among Cores, On and Off Board
2) Interrupt Handlers
3) Memory Allocation (in a distributed environment)
4) Device Driver Frameworks
5) Time and Timers
6) File System Framework
7) Logging Support

This list not meant to be exhaustive and is, in many respects, no different than such a list for a single-core system. Additional elements, depending on hardware architecture, are support for on-chip communication networks and associated communication protocols. And of course, the test of a real-time system is performance: do the tasks meet their deadlines? To answer this question, performance measurement tools will be required.

Finally, standard tools and platforms, such as Eclipse, allow additional tools, languages, and compilers, and all manner of extensions to tools within an interface known to many suppliers. In fact, not adhering to or not utilizing standards where they exist is a negative in the world of software and of computers in general.

VI. Long Term Impact of Multicore Processors to Future Missions

In the longer term, provision of high performance multicore based computers onboard spacecraft, as discussed in section II, will revolutionize robotic (and eventually crewed) space missions. As with terrestrial robotic systems, providing a space robotic system with intelligent autonomy allows the robot to carry out missions on its own with minimal direction from humans. Thus, it can react in a timely manner to unexpected events, integrating goal directed mission planning with situational awareness to quickly determine a course of action without terrestrial intervention. Not only is this capability greatly enhancing of certain missions, improving spacecraft reliability and robustness, hence, science and exploration return, but it is also enabling of whole classes of missions in which it is impractical or impossible to engage human intelligence in time to direct the mission.

Multicore computing will also shift the paradigm for how science information is provided by spacecraft assets. With the availability of increased onboard processing capabilities, future mission concepts can evolve where higher level science data products are returned to earth instead of compressed sensor data, and science of opportunity is pursued as the situation is warranted.

How multicore computing will ultimately be embedded in the spacecraft bus and in the payload is yet to be determined. The computing architectures and paradigms that will emerge over the next 20 years are similarly unknown and will evolve, probably in several directions simultaneously. What is clear, however, is that this is an exciting time for mission designers, scientists, space explorers, and spacecraft system and computer engineers. For the first time in decades, robotic spacecraft will have computing capabilities similar to those enjoyed by terrestrial systems. How creative we are in developing the form of those computing systems and how we put them to use, will determine the extent of the science and exploration we can do in the next 20 years.

VII. Conclusion

Multicore computing offers a significant advance in capabilities over the current state of the art for onboard processing. Several potential application classes have been identified for multicore computing including; (a) high throughput science data processing, (b) short duration real-time burst calculations, (c) intensive search-based reasoning, and (d) general-purpose computing. Several multicore architectural features can influence how well a processor will perform for a given application. However, this paper has identified sets of desired features (general performance, general reliability, and fault isolation and recovery) for a spaceflight multicorer processor that would

be broadly applicable for future NASA onboard processing applications. Beyond these features of the multicore processing device and computer, software tools were identified that will be necessary for application development. With these features and software tools, onboard multicore processing can enable autonomy and advanced science data processing that can lead to entirely new classes of robotic space missions.

Acknowledgments

This research was carried out both at NASA Goddard Space Flight Center and at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. The authors wish to acknowledge the NASA Exploration Technology Development and Demonstration (ETDD) Autonomous Systems and Avionics (ASA) Project, under which multicore processing architectures have been studied for spaceflight applications.

References

[1]Berger, R., Bayles, D., Brown, R., Doyle, S., Kazemzadeh, A., Knowles, K., Moser, D., Rodgers, J., Saari, B., and Stanley, D., "The RAD750™- A Radiation Hardened PowerPC™ Processor for High Performance Spaceborne Applications", *IEEE Aerospace Conference*, CP849, Vol. 1, IEEE, Big Sky, MT, 2001, Pages 2263 - 2272 vol.5.

[2]Richardson, J., Fingulin,S., Raghunathan, D., Massie, C., George, A., Lam, H., "Comparative Analysis of HPC and Accelerator Devices: Computation, Memory, I/O, and Power", *Fourth International Workshop on High-Performance Reconfigurable Computing Technology and Applications*, New Orleans, LA, 2010, Pages 1-10.

[3]Bell, S., Edwards, B. , Amann, J., Conlin, R., Joyce, K., Leung, V., MacKay, J., Reif, M., Liewei Bao , Brown, J., Mattina, M., Chyi-Chang Miao , Ramey, C., Wentzlaff, D., Anderson, W., Berger, E., Fairbanks, N. , Khan, D., Montenegro, F., Stickney, J., "TILE64 - Processor: A 64-Core SoC with Mesh Interconnect", *IEEE Solid-State Circuits Conference*, San Francisco, CA, 2008, Page 88.

[4]Gehrels, N., and Swift Science Team (1999), "Swift - The Next GRB MIDEX Mission", *Bulletin of the American Astronomical Society*, Vol. 31, Page 1512.