Multicore Considerations for Legacy Flight Software Migration

Kenneth Vines
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive, Pasadena, CA. 91109
kenneth.w.vines@jpl.nasa.gov

Len Day
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive, Pasadena, CA. 91109
len.day@jpl.nasa.gov

Abstract— This study examines the issues that should be considered when migrating legacy flight software implementations from a single processor to a multi-core environment. With the increasing availability of space qualified multicore processor single board computers, the flight software community must be prepared to utilize this new technology. Most existing flight software created at the Jet Propulsion Laboratory (JPL) is running on single core RAD750 processors with the Wind River vxWorks real-time operating system. Projects including the Mars Exploration Rovers, Mars Science Lab, Aquarius and many others utilize this platform. As always, it is desirable to reuse as much of the existing software architecture and source code as possible. This holds true for migration to multicore as well. We initiated this study to explore the most significant software migration issues to consider and the potential for innovations that were previously not realizable with a single core processor. A potentially significant benefit of adopting a multi-core implementation is improved system reliability through multi-core fault recovery mechanisms. Several options are available for implementing improved fault tolerant software systems using multicore. These may include dedicated cores for fault monitoring, redundant instances of critical functions or combinations of these with associated recovery mechanisms.

Multicore implementation of existing real-time sequential applications does not necessarily improve processing performance. An existing real-time application utilizing a single core processor architecture already executes "fast enough" with some amount of margin. A multicore implementation may *increase the performance margin* for real-time requirements of an application as well as reducing development cost associated with a "tight fit" and enabling more complex applications. This depends on how much of the application can be parallelized. According to Amdahl's Law an 8-core processor will generally double the execution speed of an application that has a parallelizable portion of 50%. An application having a 95% parallelization may see a speed increase by a factor of six. However, it is possible that tightly coupled applications may not fully benefit from symmetric parallelization due to added communications overhead between cores. Wind River has recently marketed a new product called the Hypervisor, which enables multiple instances of one or multiple operating systems to run on assigned cores. This provides a virtualized asymmetric multiprocessing capability allowing existing software targeted at a single processor to be migrated into a multicore environment gradually while freeing the developer from explicitly managing cores. Several vendors will be releasing space qualified multicore single board computers within a couple of years, with development systems that are currently available and this should be taken into account in architecture and design today.

In this paper we will discuss potential benefits and pitfalls when considering a migration from an existing single core code base to a multicore processor implementation. The results of this study present options that should be considered before migrating fault managers, device handlers and tasks with time-constrained requirements to a multicore flight software environment. Possible future multicore test bed demonstrations are also discussed.

Table of Contents

1. Introduction 1
2. Key Considerations for Multicore Flight Software Migrations 2
3. A Flight Software Migration Path 4
4. Conclusions 6
References 6
Acknowledgements 6
Biography 6

1. Introduction

The benefits of multicore implementation of space qualified processor platforms are expected to enable future space missions to incorporate far greater autonomy and fault recovery capabilities than current single core architectures. Spacecraft operations, instrument control and onboard image processing could all exist within one multicore platform, minimizing size, weight and power (SWaP) requirements. For this to occur, careful consideration must be given as to how these functions should be distributed among the available cores, how the memory, interrupts, I/O resources and even multiple instances of the operating system are to be managed.

It is desirable to utilize legacy flight software at JPL that is developed to run on a single core processor such as the BAE RAD750. Since the time critical components of the software have been proven to meet their performance requirements, these components should be able to meet these requirements with greater margin given a dedicated core and interfaces with equal or greater capabilities. But to take full advantage of what multicore can offer, we need to consider potential benefits of increased functionality by utilizing multiple cores, improved fault recovery mechanisms and integrated instrument flight software functions.

Several space qualified multicore platforms will become commercially available in the near future. Prototype versions will be available even sooner. Planning a path for migration to multicore can begin by considering some key issues regarding architectural choices that are available.

2. Key Considerations for Multicore Flight Software Migrations

When analyzing the feasibility of migrating legacy flight software to a multicore platform, several key issues should be considered. Does the legacy software architecture easily lend itself to a distributed processor implementation? Should software modules having time-critical performance requirements be allocated their own core or can they coexist with other modules? Does a single core have the capability to meet these performance requirements? What are the I/O bandwidth requirements for each of the assigned cores? Does the effort required for a multicore migration of this application warrant the advantages of its implementation? What parts of the system lend themselves to operating system or hypervisor allocation of resources and what parts may require their resources to be explicitly allocated (as in a dedicated core)? How can separation of software components increase system reliability and perhaps reduce development cost of less critical components? This section addresses some of these concerns.

Multicore Aware Operating Systems and Hypervisors

A Symmetric Multiprocessing (SMP) operating system manages all processor cores by assigning tasks to available processor cores and moves tasks among the processors to provide load balancing. A software component having significant processing requirements may be written to take advantage of multiple cores by multithreading, but could require a substantial development effort to implement. Operating systems that support unsupervised Asymmetric Multiprocessing (AMP) enables the user to configure each core with it's own operating system and functional software components. It is possible to distribute different operating systems among the cores. The user must also assign cores such that the required resources are provided. Functional software components of an existing software architecture need to be partitioned and assigned an affinity to a specific core. Time critical components will most likely require a dedicated partition, whereas multiple non-time-critical components may be assigned to a shared partition, while keeping in mind the resources available to the assigned core. An advantage of an unsupervised AMP implementation versus SMP is the legacy software components scale well with minimal modification.

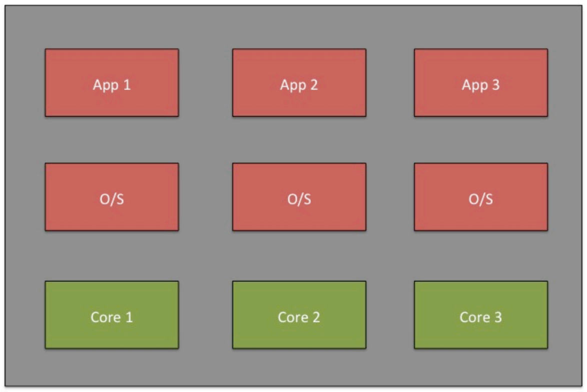

Figure 1 – Unsupervised AMP

Supervised AMP provides dynamic core assignment of a partition containing an O/S and some number of functional components. As with unsupervised AMP, one partition is assigned per core. A hypervisor is a hardware virtualization manager that provides a virtual board (VB) interface. Virtual boards share one or more cores during execution. A hypervisor usually schedules virtual boards that run on a shared core based on priority assignment or round-robin.

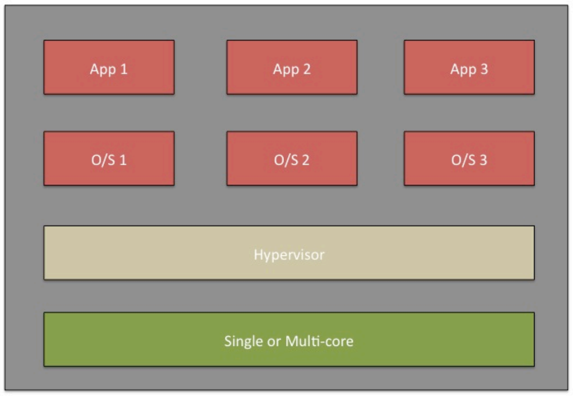

Figure 2 – AMP with Hypervisor

The developer may assign multiple components of a legacy flight software system to a partition. The O/S residing in the partition schedules the component within the partition. The supervised AMP implementation allows different types of operating systems to be dynamically assigned. Alternatively, running no O/S within a partition is possible (known as bare metal implementation). Therefore, as the supervisor assigns various functional components to a core during operation, it is possible various operating systems are being run on a core at any given time.

Analyzing Legacy Software for Multicore Distribution

Existing legacy flight software most likely consists of some number of functional components. These components should have well defined time-constraints. A component may execute one or more tasks at a specified priority. Application program interfaces (APIs) should also be well defined among the software components. The API should define all messages, events and memory specifications for the component. In addition, hardware interfaces required for use by the component should be defined. Given this information, a software designer can begin to consider how to most effectively distribute these components onto a multicore processor platform.

The existing flight software architecture was developed to run on a single processor core and has proven to meet the project system requirements for which it was designed. Migrating the legacy code to multicore can improve the system performance margin and allow for additional system functionality and robustness. It is assumed here that the processor performance for each core in a given multicore device is at least equivalent to the legacy processor. For an unsupervised AMP implementation, the legacy software components can be separated into partitions that will each run on a dedicated core. As one would expect, the most time-critical components may need to be assigned individually to a dedicated core. Some external devices may be directly accessible to only one core or a subset of cores depending on the platform. Core assignment of the functional components should take this into consideration to avoid unnecessary data traffic between cores. Device driver components will be assigned to the core that will interface to the specific device. Devices that need to be shared among cores may require updates to the driver to provide multi-access capability. Alternatively, a device handler in the application may be developed to provide an API for shared access to the device. Ideally, components that require common device access will be assigned to a common core.

Legacy fault monitoring mechanisms can be upgraded to take advantage of the capabilities of a multicore processor architecture. As an example, software components may be swapped between cores if a fault monitor determines a core or its interfaces are non-functional. On an unsupervised AMP multiprocessor system, the fault monitor may command a functional component to transfer to an alternate partition, where the active component on the failed core is deactivated and the corresponding component on an alternative core is activated. Other possible multicore fault recovery mechanisms are discussed later.

Other than a possible fault recovery scheme as mentioned above, the component and O/S assignments remain static for most unsupervised AMP implementations. The effort required to migrate to an unsupervised multicore environment will depend on the coupling of resources and components in the legacy system. Careful consideration of partitioning the functional components will improve the probability of a successful migration. A hypervisor implementation can reduce the implementation effort because the legacy software components can be distributed to any number virtual board partitions. The dynamic virtual board environment can be configured for optimum performance and flexibility, minimizing legacy code modification effort. As with unsupervised AMP, time-critical components should be assigned to a dedicated virtual board and core. Also, care should be taken to configure the hypervisor to map the components requiring devices to cores that provide an interface to the device. Some hypervisors provide options for device access among cores. A direct access option to a device provides native core performance. Shared access may be provided, but with some performance reductions. An example migration to a hypervisor based multicore system is shown in section 3.

Interprocess Communication

Migration of existing frameworks such as those for Interprocess Communication (IPC) to a multicore platform is needed. Fortunately multicore aware operating systems provide mechanisms that reduce the complexity of migrating some of these frameworks. Multi-OS Interprocess Communication (MIPC) and the Multicore

Communications API (MCAPI) provide an API that routes messages between cores and virtual machines. Once an existing framework is modified to utilize the mechanism provided by the O/S, reusable libraries can be generated for future project use.

Interrupt and Device Handlers

An interrupt source from a device or a discrete signaling event is handled by an interrupt service routine running on a core that is assigned to service the event. Software components residing on a processor core that require access to device or external event that is serviced by another core may utilize IPC communication with the device or event handler if throughput response meets the performance requirements. In a virtualized hypervisor environment, multiple virtualized components may access shared devices. The hypervisor manages the scheduling of device access among multiple virtualized software components as shown in Figure 3.

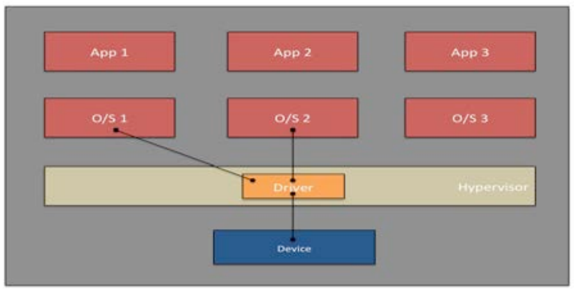

Figure 3 – Virtualized Device Access

Fault Handling Considerations

Fault monitoring mechanisms for legacy flight software can be enhanced to take advantage of multicore. As an example, the Mars Science Laboratory (MSL) implemented a fault recovery software component named Second Chance that executed on a second string processor during Mars entry, descent and landing (EDL). This component ran in parallel to the primary system and consisted of a subset of critical tasks to be performed in the case of primary processor reboot during EDL, which did not occur. This mechanism could possibly be modified to run on a dedicated core, in addition to alternate processor strings. If the O/S running a critical component such as EDL experiences a failure or reboot, a fault monitor could activate a Second Chance component on a dedicated reserved core. Another possibility is to activate the critical component itself on the reserved core if it is determined a hardware failure has occurred in the primary core.

A fault monitor in a multicore processor system may be distributed among multiple cores. In an unsupervised AMP configuration, functional components are grouped along with an O/S and assigned a core on which to reside. Fault monitors can be included in the grouped components assigned to each core. System status is communicated among cores providing more robust fault monitoring. Critical system state can be monitored by multiple cores, where inconsistent state is determined by a voting scheme. A hypervisor implementation can reduce the effort needed to implement fault protection on multicore. Each software component can run on a virtual board partition. A fault on a virtual board partition can be contained and the virtual board restarted on an alternate core autonomously by the hypervisor.

3. A Flight Software Migration Path

It may be preferable to migrate an existing traditional flight software system to multicore using an incremental path approach. Initially some of the software service frameworks can be retooled for a multicore testbed environment. The representative flight software architecture can then be modified for multicore and perhaps additional fault monitoring functionality can be implemented. Initially the O/S or hypervisor can be used for resource allocation with minimal impact on an existing implementation. A partitioning of the legacy software into functional components that coexist on a virtual board will minimize the effort needed for a multicore migration. System performance can be monitored on the testbed as the configuration parameters for the hypervisor are modified for optimization.

Application Partitioning Description and Modeling

A Synthetic Aperture Radar (SAR) instrument controller provides an example of a real-time legacy software system that has been used on several projects at JPL. The software resides on a PowerPC based single board computer running the Wind River vxWorks operating system. The Radar Instrument Controller consists of nine functional modules, each executing one or more vxWorks tasks. An example of a possible multicore migration of the legacy flight software to a hypervisor based multicore system is shown in Figure 4. An analysis of the existing functional components was performed based their known performance and interface requirements. It was determined that the components could be grouped into six partitions. Each partition resides on a hypervisor virtual board (VB). Since the hypervisor handles core scheduling, the legacy software components do not require modification other than some possible API updates for inter-process communication. The multicore architecture provides additional processing capability and therefore a new software component is added for onboard SAR processing (OBP). OBP does not require real-time response, therefore it may run on a non-real-time operating system, such as Linux. Some existing components of the example software system do not necessarily require real-time response and my also be migrated to a non-real-time O/S if desired.

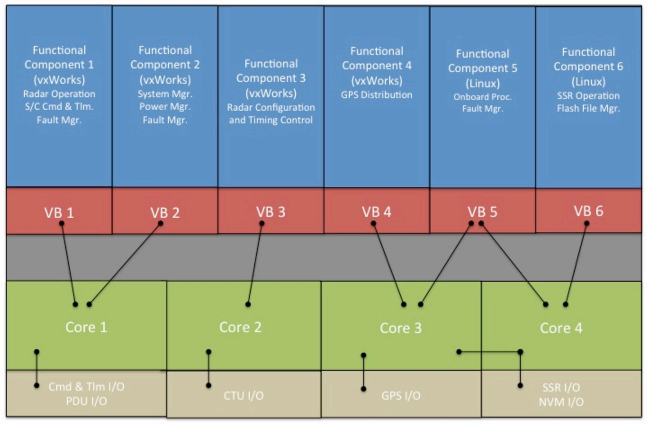
Functional Component 1 (vxWorks) Radar Operation S/C Cmd & Tlm. Fault Mgr.
Functional Component 2 (vxWorks) System Mgr. Power Mgr. Fault Mgr.
Functional Component 3 (vxWorks) Radar Configuration and Timing Control
Functional Component 4 (vxWorks) GPS Distribution
Functional Component 5 (Linux) Onboard Proc. Fault Mgr.
Functional Component 6 (Linux) SSR Operation Flash File Mgr.
VB 1
VB 2
VB 3
VB 4
VB 5
VB 6
Core 1
Core 2
Core 3
Core 4
Cmd & Tlm I/O PDU I/O
CTU I/O
GPS I/O
SSR I/O NVM I/O

Figure 4 – A Hypervisor-based multicore migration.

This migration example shows four cores allocated to the Radar Instrument Controller. Other available cores may be allocated to spacecraft specific functions or other instruments. The hypervisor is configured to allocate the virtual boards to the available cores as shown in Figure 4. The most time-critical software component resides on VB3 and is assigned an affinity to Core 2. No other VB is allowed to run on this core during nominal operation. Components assigned to VB1 and VB2 are assigned an affinity for Core 1, VB4 is assigned to Core 3 and VB6 assigned to Core 4. VB5 contains the OBP is allowed to run on both Core 3 and Core 4 as a SMP component. The hypervisor handles scheduling of the virtual boards among the cores. VB1 and VB2 implement prioritized scheduling for Core 1, VB4 and VB5 for Core 3, VB5 and VB6 for Core 4.

External devices are assigned to device drivers directly residing on the cores shown in the figure. In the example, access to data residing on the Solid State Recorder (SSR) and Non-volatile Memory (NVM) is required by multiple components. The hypervisor is configured to share access to these devices. Some modification of the device driver may be required. An alternative implementation is to provide a device handler service at the application level to share device data among functional components, depending on the throughput limitations.

The Radar Instrument Controller has a fault manager that monitors system state and telemetry. For this multicore migration, the fault manager is divided among 3 virtual boards. Fault monitoring functionality is distributed. As with the original single core implementation, each component periodically sends status to the fault monitor. If a component reports invalid status or fails to report any status, the fault handler may reconfigure virtual board assignments as deemed necessary. In Figure 5, Core 4 has failed. VB5 and VB6 are not responding. The fault detection mechanism instructs the hypervisor to assign VB5 and VB6 to Core 3. Now three virtual boards are running on Core 3 in with prioritized scheduling. The device drivers for the SSR and NVM that were assigned to Core 4 are now assigned to Core 3.

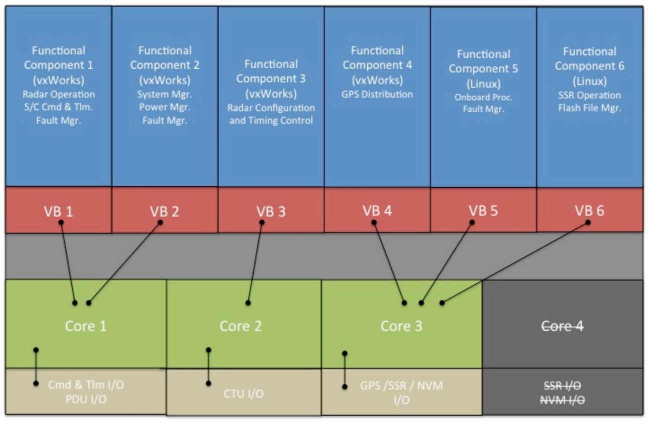
Functional Component 1 (vxWorks) Radar Operation S/C Cmd & Tlm. Fault Mgr.
Functional Component 2 (vxWorks) System Mgr. Power Mgr. Fault Mgr.
Functional Component 3 (vxWorks) Radar Configuration and Timing Control
Functional Component 4 (vxWorks) GPS Distribution
Functional Component 5 (Linux) Onboard Proc. Fault Mgr.
Functional Component 6 (Linux) SSR Operation Flash File Mgr.
VB 1
VB 2
VB 3
VB 4
VB 5
VB 6
Core 1
Core 2
Core 3
~~Core 4~~
Cmd & Tlm I/O PDU I/O
CTU I/O
GPS /SSR / NVM I/O
~~SSR I/O~~ ~~NVM I/O~~

Figure 5 – Core failure fault recovery

The ability to dynamically change virtual board core assignments will depend on the capability of the hypervisor. However, a fault recovery manager could have the capability to enable and disable functionality per virtual board if the hypervisor does not provide this functionality.

Testbed Implementation

A testbed platform containing a prototype multicore flight computer architecture provides an invaluable path for a project to gain confidence and experience in the ability to eventually utilize multicore in space based platforms. Several vendors currently provide or will soon provide radiation-hardened (>100 krad) versions of multicore single board computers for use in spacecraft platforms. These rad-hard SBCs contain multicore Power-PC or X86 processor architectures with Rapid I/O, SpaceWire and other I/O options. Many of these vendors offer up to 8 processor cores. The boards also provide hardware assisted processor buffer management and device memory access management. Although some hypervisor implementations provide multicore buffer management in software when needed, a hardware implementation is preferable. Prototype versions of these boards are currently or soon will be available and can provide a test platform to migrate existing software components to a multicore environment.

4. CONCLUSIONS

This paper has discussed many of the issues that should be considered when migrating legacy flight software to a multicore platform. We presented an example path to migrate existing legacy software to multicore. Several commercially developed space qualified multicore platforms are currently or will soon be available. Migration of existing functional software components onto a prototype multicore processor platform will enable flight software to take advantage of multicore space platforms in the near future.

REFERENCES

[1] Wesley Powell, Michel Johnson, Jonathon Wilmot, Raphael Some, Kim Gostelow, Glen Reeves, Richard Doyle, "Enabling Future Robotic Missions with Multicore Processors," American Institute of Aeronautics and Astronautics, 2011.

[2] Berger, R., Bayles, D., Brown, R., Doyle, S., Kazemzadeh, A., Knowles, K., Moser, D., Rodgers, J., Saari, B., and Stanley, D., ―The RAD750TM- A Radiation Hardened PowerPCTM Processor for High Performance Spaceborne Applications‖, *IEEE Aerospace Conference*, CP849, Vol. 1, IEEE, Big Sky, MT, 2001, Pages 2263 - 2272 vol.5.

ACKNOWLEDGEMENTS

The research described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

BIOGRAPHY

Kenneth Vines *is a senior software engineer at the Jet Propulsion Laboratory in Pasadena, CA. He has developed and/or led development and testing of flight software for various suborbital SAR instruments, Cassini Radar and the Shuttle Imaging Radar programs. He has a BS and MS degree in Computer Science from California Polytechnic State University Pomona.*

Len Day *is a senior software engineer at Jet Propulsion Laboratory in Pasadena, CA. He has been involved in many space missions going back to Viking and including Seasat, Milstar, Cassini, MSL and others. Additionally he has substantial industry experience in areas of operating systems, networking and low level software in general.*