Solving Multicore Interference for Safety-Critical Applications

Contents

Executive Summary

The use of multicore Arm®, Intel®, and Power Architecture® processors in safety- and security-critical systems, such as integrated modular avionics (IMA), presents challenges for determinism and safety due to substantial variations in worst-case execution times resulting from contention for access to shared processor resources. The effects of such multicore interference can be significant, and mitigation is best done with support from the operating system. A general solution is needed that enables changes to the applications without needing to retest every application. One such solution is fine-grain control of the system bandwidth allocated to each processor core. When combined with time and space partitioning, such interference mitigation speeds development, testing, verification, and certification of multicore safety-critical systems.

Introduction

Safety-critical systems for airborne applications are gradually adopting multicore processors to realize the benefit of increased throughput while decreasing size, weight, and power of the computing solution. Together these benefits enable consolidation of multiple functions in an integrated modular avionics (IMA) solution. It is important to note that a goal of IMA is to enable efficient and reliable sustainment and growth of the systems' functionality via software upgrades, improvements, and additions at an affordable cost. Achieving that goal can be very difficult because of the increased variability that occurs when multiple processor cores try to access the same shared resources concurrently. In safety-critical applications, the principal concern is how such shared resource contention can cause an application running on one core to interfere with a different application running on another core, negatively affecting determinism, quality of service, and, ultimately, safety. Without a general solution for mitigating such multicore interference, any software changes or additions will require extensive retesting and analysis of the entire system, contrary to the goals of IMA.

The Single Core Solution

In a single-core processor, multiple safety-critical applications may execute on the same processor by robustly partitioning the memory space and processor time between the hosted applications. Memory space partitioning dedicates a non-overlapping portion of memory to each application running at a given time, enforced by the processor's memory management unit (MMU). Time partitioning divides a fixed time interval, called a Major Frame, into a sequence of fixed sub-intervals referred to as partition time windows. Each application is allocated one or more partition time windows, with the length and number of windows being factors of the application's worst-case execution time (WCET) and required repetition rate. The operating system ensures that each application is provided access to the processor's core during its allocated time.

Sources of Multicore Interference

On a multicore processor, applications are time-partitioned on each core but can be running concurrently with applications on other cores. The problem is that each of the concurrent applications needs access to the processor's shared resources. All multicore processor architectures include shared resources, such as memory controllers, DDR memory, I/O, shared cache, and the internal fabric that connects them (Figure 1). Contention for these shared resources results when a processor core tries to access a resource that is already servicing another processor core. The resulting delay is a form of interference that could prevent a high-criticality application from performing its intended function within its required timeframe. Because the DMA engine can execute in parallel with the cores, it presents an additional source of interference.

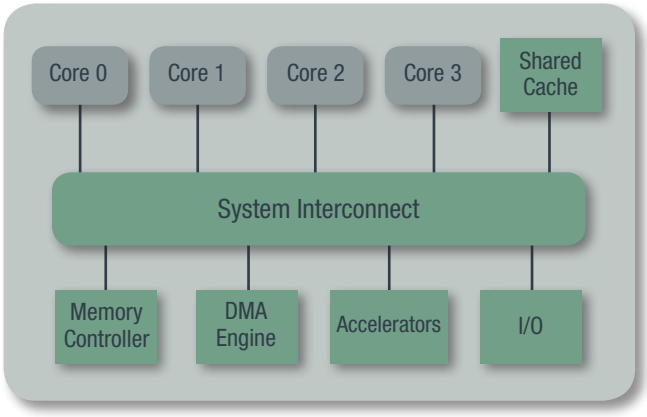

Figure 1: *Separate processor cores (**gray**) share many resources (**green**) ranging from the interconnect to memory and I/O.*

Directly addressing the issue of multicore interference, the Certification Authority Software Team (CAST), supported by the FAA, EASA, TCCA, and other aviation authorities, has published guidance for multicore systems in a position paper called CAST-32A. CAST-32A includes 10 objectives that need to be satisfied to address the concerns with the use of multicore processors. One of the objectives that directly addresses multicore interference requires the applicant to identify all the interference channels that could affect the software applications hosted on the processor cores and verify the effectiveness of the chosen means of mitigation.

To truly identify all the sources of multicore interference requires a deep understanding of the multicore processor subsystems, how they operate, and how they interact. Such understanding can include hardware details such as DDR geometry, memory controller scheduling priorities, cache replacement policies, the multicore interconnect's arbitration schemes, and hardware initialization & configuration options.

Gaining an understanding of multicore interference and mitigation is not just a concern for safety but also security. If the availability of one application can be affected by another, then it is neither safe nor secure. Digging a little deeper, the sources of covert timing channels required for many types of security attacks are often the same sources of interference with respect to availability concerns. The bottom line is that multicore interference needs to be tightly mitigated.

Effects of Multicore Interference Can Be Significant

This concern over multicore interference can appear to be excessive given that individual access times to shared resources are usually much less than 1 microsecond. These small increments can add up quickly, however, and repeated interference can act as a temporary denial of service. In the simplest case with fair arbitration of resource access, two cores trying to access the same resource simultaneously in the same way, would each get half the of the bandwidth of that resource. Similarly, four cores would each get a quarter of the bandwidth. Having an application at the highest design assurance level (DAL A) take four times longer is far from desirable. To make matters worse, the accesses are not always the same type, and arbitration is not always fair in precisely the way one would hope.

Consider multiple cores accessing DRAM. Due to shared resource arbitration and scheduling priorities in the DDR controller, fairness is not guaranteed, and interference impacts are often non-linear. The type of operations or combination of operations (e.g. reads, writes, coherency traffic, broadcast operations, etc.) and/or which specific core they are executing on, plays a significant role in the interference impact. For example, write operations to memory can generate a disproportionate amount of interference over read operations on certain architectures. Although some companies have claimed that multicore interference causes worst-case execution time (WCET) to double or triple at the most, tests on Power Architecture® cores show that a single interfering core can increase WCET on another core by a factor of 8x. With multiple sources of interference from multiple cores, increases in WCET of over 12x have been observed in a quad-core system just from the cores accessing DDR memory over the on-chip interconnect. Those increases do not factor in the impact of I/O accesses or the DMA engine running simultaneously.

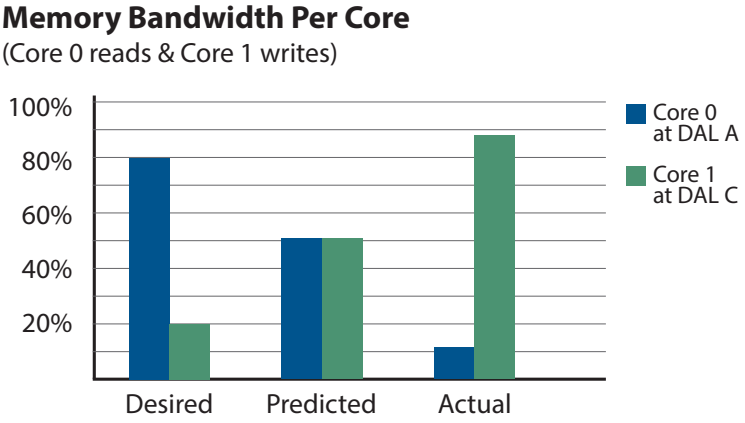

***Figure 2**: Example memory access bandwidth for DAL A and DAL C applications on different processors trying to access DRAM simultaneously.*

Figure 2 shows example memory access bandwidth for DAL A and DAL C applications on different processors trying to access DRAM simultaneously. The desired outcome is for the DAL A application to get the vast majority of the bandwidth. Without any interference mitigation, one might predict that the bandwidth would be split evenly. Actual observed results on Power architecture cores show that the DAL C process gets the vast majority of bandwidth if it is doing certain types of operations, up to 8X the bandwidth of the DAL A process for merely a single interfering core.

Alternative Solutions for Interference Mitigation

There are a variety of possible approaches to reducing and mitigating multicore inference. In theory, it would be desirable to reduce the interference as much as possible to make the job of mitigation easier. In practice, most methods to reduce interference require greatly restricting the applications or significantly reducing efficiency and throughput of the overall solution. For example, restricting all the applications running on each core to be at DAL B or higher would ensure that only well tested and reliable applications are present, and those presumably would be less likely to cause runaway or unknown interference. Another approach could be to take all the I/O and move that to a single core with time partitioning, but that could

require rewriting existing applications and reducing the utilization of the other cores. Even if that approach worked for I/O accesses, it doesn't address the more significant problem of interference for DDR memory. All applications need to access DDR memory unless they are so small as to run out of local core cache memory.

The DMA engine is one resource that benefits from more careful management. By managing the timing of I/O and when it causes DMA transfers, the amount of interference caused by the DMA engine can be reduced.

The vast majority of multicore interference cannot be reduced, only mitigated. Mitigation approaches can vary greatly from "just deal with it" to automated fine-grained control of access to shared resources that monitor and strictly enforce the use of shared resources. A simple but inefficient approach is just to estimate the new WCET given maximal interference, set the partition time window to that new WCET, and test extensively to verify that application consistently completes within the window. The biggest drawback to that approach is the vast underutilization of multicore throughput if the new WCET is 10x the non-interfering time when a more active mitigation strategy might require only 1.5x. Additionally, this could lead to increased sustainment costs as applications are added or updated on the system and the WCET changes, thus requiring further regression testing.

A better approach is to monitor the access to shared resources and regulate access if a core exceeds a predetermined threshold. This can be accomplished using the hardware counters available on most multicore processors and is relatively straightforward at a macro level. Each application gets a threshold set for its partition time window, and if that threshold is exceeded, then the application is suspended. Such a coarse-grain approach can work well to catch a misbehaving application but does not address the more common case of a lower criticality application taking bandwidth from a higher criticality application. For that, fine-grain control is required where the threshold applies to a time slice that is many times smaller than the partition time window.

With such fine-grain control, bandwidth to shared resources for a lower-critically application is throttled back to the desired threshold rate, and the higher criticality application gets its full allocation of bandwidth from the beginning of its partition time window.

The INTEGRITY-178 tuMP Multicore RTOS

The INTEGRITY®-178 tuMP™ multicore RTOS is explicitly designed to meet the challenge of high processor utilization, application portability, and true IMA operation while mitigating multicore interference and providing DO-178B/C DAL A safety-critical operation. Available on Arm, Intel, and Power Architectures, INTEGRITY-178 tuMP delivers on the key tenets of IMA: consolidation, portability, sustainment, and reuse to enable higher functionality and lower life-cycle costs.

The INTEGRITY-178 tuMP RTOS is a high-assurance operating system certified for both safety and security. It provides partitioning of memory space, processing time, and processor resources to ensure safety-critical applications meet their real-time deadlines. In 2002, INTEGRITY-178 became the first commercial partition-enforcing RTOS certified as complying with DO-178B Level A objectives. With a high degree of code reuse, the INTEGRITY-178 tuMP multicore RTOS was delivered to customers in 2010. Today, it is the only RTOS certified conformant to the latest version of the Future Airborne Capability Environment (FACE™) Technical Standard, Edition 3.0. Superior utilization of all cores on a multicore processor can be enabled by providing several options for a multi-processing software architecture that are supported by the RTOS to run applications across multiple cores simultaneously. The INTEGRITY-178 tuMP multicore RTOS provides the system software architect with the flexibility of full multicore support for all combinations of Asymmetric Multi-Processing (AMP), Symmetric Multi-Processing (SMP), and Bound Multi-Processing (BMP) in a

time-partitioned manner. BMP is an enhanced and restricted form of SMP that can statically bind an application's tasks to a specific set of cores, allowing the system architect to more tightly control the concurrent operation of multiple cores. The AMP, SMP, and BMP operational requirements for each of the individual IMA applications are defined at build time. These definitions, as well as when and how often they execute within a partition schedule, are managed by the system's software architect using statically defined system configuration files. Support for BMP operation is required to meet the latest revision of ARINC 653 Part 1 Required Services (Supplement 4), and INTEGRITY-178 tuMP is the only ARINC 653 RTOS that complies with Supplement 4.

Fine-Grained Bandwidth Allocation Mitigates Multicore Interference

As described above, mitigation of multicore interference is best done by allocating shared resource bandwidth to each core and then enforcing that allocation in very small time intervals. Such fine-grain monitoring and enforcing allocations of bandwidth to shared resources can currently only be done at the operating system level.

INTEGRITY-178 tuMP includes a Bandwidth Allocation and Monitoring (BAM) capability to observe interference channels and mitigate them. Based upon more than 50 staff-years of research and development into multicore interference analysis and mitigation strategies, BAM monitors and enforces the bandwidth allocation of the chip-level interconnect to each of the cores. Because the chip-level interconnect is at the center of interactions between the cores and other shared resources, it is the ideal place to observe and enforce limits on the use of shared resources. Green Hills Software has implemented an internal mechanism for INTEGRITY-178 tuMP' bandwidth allocation and monitoring that uniquely uses an extremely small time quantum to enforce the cores' use of shared multicore resources as opposed to the typical approach using high-level fault detection. In that way, BAM throttles the bandwidth of any resource hog throughout the partition time window, and the other processes do not have to wait until the resource hog has consumed all of its allocation for its partition time window.

The system architect decides how much bandwidth to allocate to each core based on the functional requirements of the applications or design assurance levels. When applications on a particular core reach the threshold bandwidth for a given BAM time quantum, that core is cut off from consuming shared resources until the next BAM time quantum. Using this mechanism, a DAL A application running on core 0 can be allocated a set amount of resources, such as 60% of the total bandwidth, while the other 3 cores could be allocated only 15%, 15%, and 10% respectively (see Figure 3). BAM is developed to DO-178C DAL A objectives, and it allows integrators to mitigate interference issues.

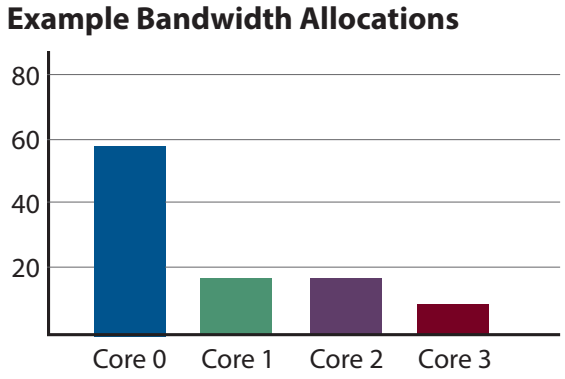

***Figure 3**: Example bandwidth allocations*

Setting the proper bandwidth allocation requires analysis and testing of the application. To aid in that analysis, Green Hills Software provides interference generating libraries, DMA generating libraries, and bandwidth reporting libraries. The interference and DMA generating libraries are tailored to each processor architecture and contain hundreds of interference profiles to simulate interference for worst-case scenarios. Running the interference and DMA generating libraries on all cores not used by a particular application concurrently with the application execution provides the new multicore

worst-case execution timing (WCET). Without the ability to generate worst-case interference from both the DMA engine and the other cores, it would be easy to vastly underestimate the multicore WCET.

The bandwidth reporting library uses the interference and DMA generating libraries to get a measured assessment of the total amount of bandwidth available after accounting for the DMA interference. Knowing the total bandwidth available aids in setting the bandwidth allocation thresholds in BAM. The bandwidth reporting library runs the interference and DMA generating libraries across a configurable number of cores concurrently. Specific subsets of the hundreds of interference profiles can be selected in order to tailor the evaluation more closely to the expected applications, and custom interference profiles can be created. The available bandwidth depends not only on the processor model but also the memory type, clock speed, configuration registers, and which interference profiles were selected to approximate the final application configuration.

Multicore Interference Mitigation in Action

The following figures show an example that illustrates the improved timing characteristics when BAM is enabled. The data used is taken from results measured on a quad-core Power Architecture processor, and the application execution times are relative to the application running on a single core while the other cores are idle. The application under test is running on core 0, with the interference generating libraries running on the other cores. Figure 4 shows how the execution time of the application running on core 0 increases as the number of interfering cores increases. When one other core runs the inference library, the execution time of the application under test skyrockets over 7x. With a second and third interfering core added, the execution time increases further to over 13x. Note that this worst-case behavior is much worse than the linear 2x, 3x, 4x that might be expected for average cases.

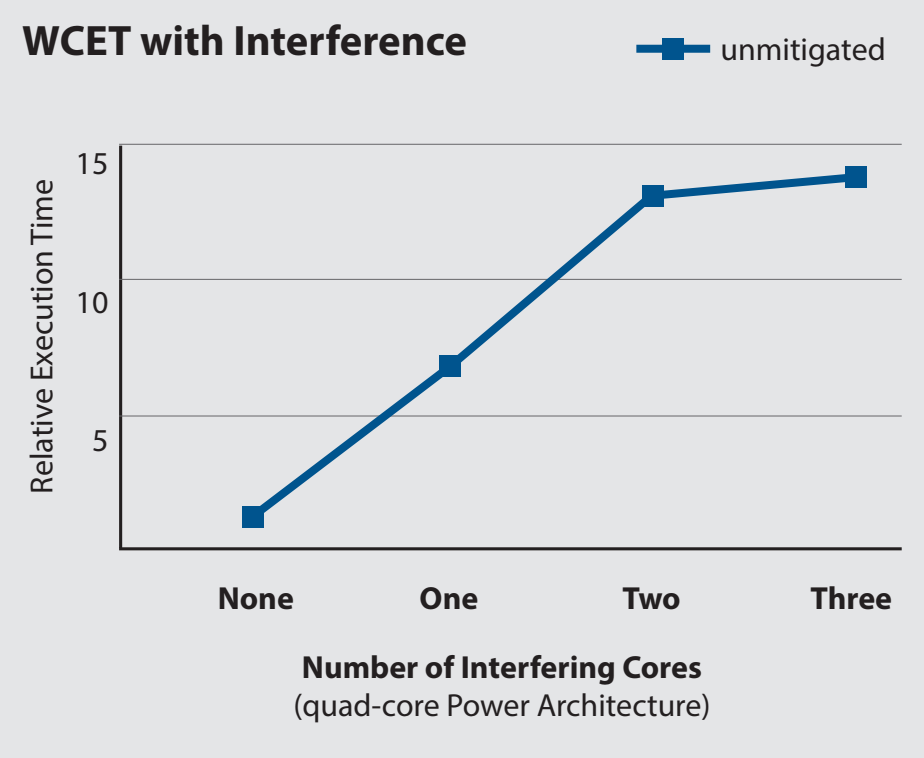

***Figure 4**: Worst-case execution timing varies with number of interfering cores*

To mitigate that interference, the system architect uses BAM to set a maximum bandwidth allocation for each core. Once the bandwidth from the interfering cores is limited, the execution time of the application under test comes back under control and remains nearly constant as the number of interfering cores increases. When the application under test is allocated 80% of the total shared-resource bandwidth, execution takes only 1.4x longer than when it runs by itself and maintains approximately that execution rate when one or more interfering cores run simultaneously. Figure 5 shows that 1.4x execution rate for 80% bandwidth allocation, as well as the performance for different allocations at 50% and 25% bandwidth. Note that even with only a 25% bandwidth allocation, nominally an equal share of the bandwidth, the application still runs 2.5x faster than without any interference mitigation.

Taken together, the interference and DMA generating libraries, bandwidth library, and BAM runtime mechanism provide the tools necessary for a system integrator to determine multicore worst-case execution times, mitigate interference, and certify multicore systems. These interference mitigating capabilities provided by Green Hills Software reduce certification risk and enable faster time-to-market by simplifying the verification and analysis activities while also significantly lowering the cost of long term sustainment and growth of the system. Without such multicore interference mitigation capabilities, system reverification after adding or modifying applications incurs a very high risk of needing to retest every application. As a result of such risks, system level analysis techniques are insufficient to support the intent of IMA systems. For multicore-based IMA systems, functional capabilities like bandwidth allocation and monitoring are a fundamental requirement if that system is to provide the full set of benefits of an IMA architecture.

Looking at the overall safety-critical system, BAM reduces risk and simplifies the development, integration, deployment, and sustainment of critical systems. BAM enables optimal core utilization in critical systems yielding superior size, weight, and power as well as spare computing capacity. This bandwidth allocation and monitoring capability is critical for IMA OEMs and developers.

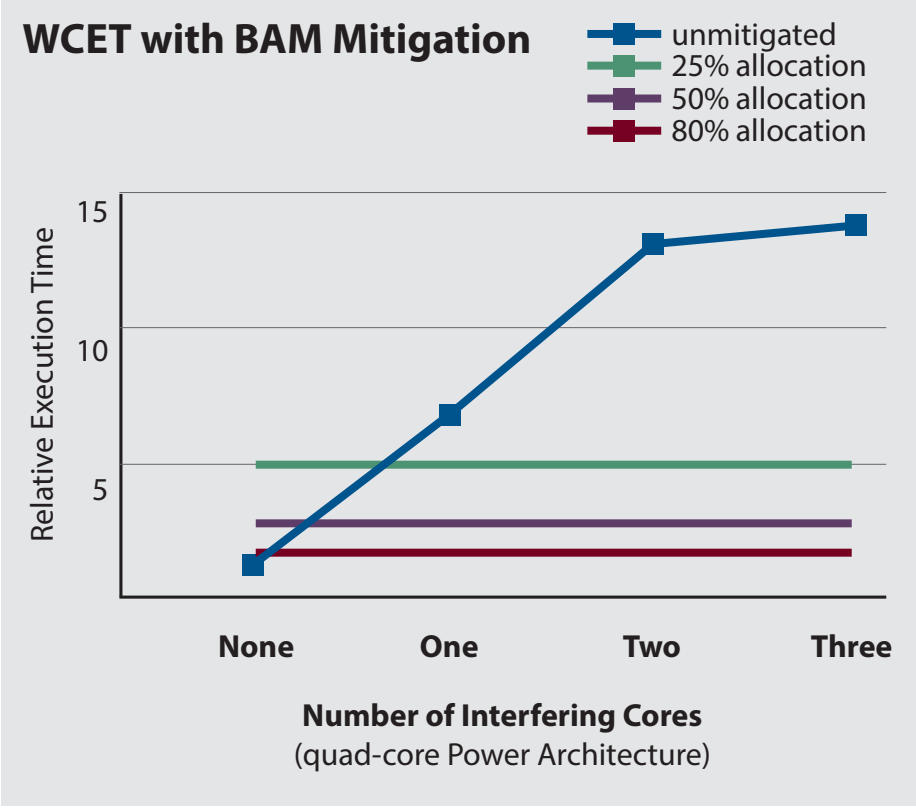

***Figure 5**: Worst-case execution timing stays steady after applying BAM*

Corporate Headquarters

30 West Sola Street • Santa Barbara, CA 93101
ph: 805.965.6044 • fax: 805.965.6343 • email: info@ghs.com • www.ghs.com

European Headquarters

Fleming Business Centre • Leigh Road • Eastleigh • Hampshire S050 9PD • United Kingdom
ph: +44 (0)2380 649660 • fax: +44 (0)2380 649661 • email: info-emea@ghs.com

Safety & Security Critical Products

34125 US Hwy 19 North • Suite 100 • Palm Harbor, FL 34684
ph: 727.781.4909 • fax: 727.781.3915 • email: info-sscp@ghs.com

Green Hills, the Green Hills logo, INTEGRITY, and tuMP are trademarks or registered trademarks of Green Hills Software in the US and/or internationally.
All other trademarks (registered or otherwise) are the property of their respective owners.
© 2019 Green Hills Software. v0719