SSC22-P2-12

Space-operating Linux: An Operating System for High Performance AI Computation on Commercial-Grade Equipment in Low Earth Orbit

Eric Miller
University of Georgia, Institute for Artificial Intelligence
1023 D. W. Brooks Drive, Athens, GA 30605
ericmiller@uga.edu

ABSTRACT

With recent exponential advances in AI—particularly with respect to the tremendous power and efficiency accessible for data processing—there is now a countless array of applications for aerospace missions and space exploration, even in experimental CubeSats. However, with the large volume of data acquisition required for satellite missions, downlinking presents an increasingly expensive bottleneck that drastically reduces mission efficiency. At the University of Georgia Small Satellite Research Laboratory, the Multi-view Onboard Computational Imager (MOCI) employs an NVIDIA Jetson TX2i GPU module to process data in situ and downlink only final products. As is the case for many commercial off-the-shelf (COTS) devices carried in CubeSats, the TX2i does not come radiation-hardened, as it houses a vulnerable eMMC disk. MOCI will employ hardware shielding to help counteract this issue, but there is nevertheless a possibility of single event effects (SEEs) reaching the TX2i, calling for software-level mitigation as a final line of defense. Using Yocto, we create a minimized operating system with built-in redundancy to reduce reliance on flash memory, particularly with a custom bootloader utilizing a triple modular redundancy (TMR) partition scheme and a RAM-based file system available upon boot.

INTRODUCTION

Motivation for AI in Space Exploration

On February 18, 2021, the Mars 2020 mission's Perseverance rover touched down on Martian terrain, equipped with a state-of-the-art imaging system. In supplement to the cameras used to document the rover's descent and its collection of samples, the vehicle houses two cameras for navigation and four for hazard avoidance. Each camera is 20 megapixels, requiring buffered image transmission into the rover's flight software, as well as lossy compression for downlinking. However, when coupled with the rover's enhanced AutoNav features since NASA's MSL and MER missions, these imaging subsystems enable semi-autonomous driving and arm movements, reducing both the amount of computation required by and the time of transmission to human engineers.[1,2]

Mars 2020 highlights two key points regarding the current and future states of scientific missions in space. Primarily, due to advances in sensor technology, as well as larger mission scopes, extraterrestrial endeavors are becoming highly data-intensive, with image and/or raw data transfers becoming a bottleneck to mission progress. Moreover, Perseverance among its predecessors illustrates the ever-expanding breadth of humanity's scientific exploration, requiring vehicles to reach further into space, where response times to and from Earth become slow and unreliable.

These two factors motivate projects to have a higher capacity for on-board data processing and autonomy. The feasibility of various imminent projects, such as a human mission to Mars, will require a decreased reliance on spacecraft-to-Earth communication. Accordingly, onboard applications of Artificial Intelligence in data analysis, distributed systems, swarm intelligence, and fault tolerance can permit spacecrafts to autonomously make necessary responses to their environments.[3]

The Advent of CubeSats

As Poghosyan and Golkar (2017) detail, another principal factor in the advancement of space technologies has been the industry's expansion into the private sector. Conventional satellite missions, since 1957 saw the launch of Sputnik I, have been restricted to government-funded agencies, as the sheer size of the satellites (meant to house multiple instruments to optimize cost) required both a large team and budget. However, with the technical challenges of engineering large spacecrafts, as well as the availability and reduced size of commercial-off-the-shelf

(COTS) products in recent decades, the space industry has inclined towards smaller satellites.

With small satellites' trending popularity, Stanford and California Polytechnic standardized the CubeSat as a composition of 1U volumetric units, each $10 \times 10 \times 10$ cm^3 and up to 1.33 kg in mass. Since their inception in 1999, CubeSats have ranged in application from the original proof-of-concept and educational technology demonstrations to full-scale scientific discovery missions. Their ease of development and low cost have enabled smaller countries and even educational institutions to devise and launch spacecrafts with state-of-the-art COTS technology. In particular, CubeSat missions have become a low-risk avenue for developers to test incremental updates in scientific payload technology.[4]

GPUs and their Applicability in Spacecrafts

Vuduc and Choi (2013) discuss the history of graphics processing units (GPUs) from their earliest applications, which, as their name suggests, was for 3D graphics modeling, particularly in gaming systems. As GPUs began to greatly outperform CPUs (by factors of 10–100) in parallel computational tasks required for graphics rendering, their capabilities captured the interest of developers in other fields. Due to the difficulty of developing GPUs for tasks outside of the graphics realm, NVIDIA developed the Compute Unified Device Architecture (CUDA) in the early 2000s as a high-level platform for developers to interface with GPUs. This gave rise to a new software-level paradigm, the general-purpose GPU (GPGPU), which allows programmers to utilize GPU hardware to explicitly parallelize tasks of their choosing. In particular, GPU hardware has proven useful for single instruction multiple data (SIMD) designs, where one operation can be designed to perform simultaneously over a large set of data values. CUDA has now grown to include a user base in the millions and is largely used in GPU-based research.[5]

GPUs are commonly utilized for AI applications, particularly in deep learning and computer vision, where intensive linear algebra computations that would heavily bottleneck CPU programs can be refactored into SIMD designs and accelerated via GPGPU programming. Hence, for spacecraft missions implementing such AI applications, such as those involving large volumes of image data, GPUs are often used for on-ground processing. However, due to the constraints of data transmission discussed earlier, teams have recently become motivated to incorporate GPUs onboard. GPUs' ease of programming and commercial availability make them suitable low-cost candidates for such missions, but the available hardware is primarily designed for terrestrial operation—that is, agnostic of the radiation environment that exists outside of Earth's atmosphere.[6]

Radiation-Induced Effects

The majority of satellites operate in low Earth orbit (LEO), where three main sources of radiation are of relevance:

1. galactic cosmic radiation (GCR)
2. trapped radiation belts
3. solar energetic particles (SEPs)

GCR includes particles of a wide range of energies originating from beyond the solar system, and whose flux inversely correlates with solar energy. Trapped radiation belts have minor effects in low altitudes, except at high inclination (near the poles). SEPs are directly correlated with the solar cycle, as the events originate from the sun (e.g., solar flares).[7]

Radiation in such an environment has two forms of effects on electronic devices: total ionizing dose (TID) and single-even effects (SEEs). The former refers to a build up of trapped charges that, over time, lead to functional hardware shifts. SEEs may take the form of, among others, single-event upsets (SEUs) such as bit-flips that cause corruption or single-event latch-ups (SELs) that lead to circuitry-level failures. Radiation mitigation measures, on a physical and hardware level, include device shielding, hardware-level redundancy, and error detection and correction (EDAC). The latter two tend to carry budgetary and performance-related overheads, generally creating an inverse relationship between computational efficiency and reliability.[6]

The NVIDIA Jetson TX2i

The NVIDIA Jetson TX2i is a state-of-the-art System-on-Module (SOM) that integrates a 64-bit ARMv8 multi-processor (Dual-Core Denver 2 and Quad-Core Cortex-A57) and a 256-core CUDA-compatible NVIDIA Pascal GPU. The module also contains a memory-controller providing error-correcting code (ECC) on LPDDR4 SDRAM, as well as an eMMC flash memory card. A block diagram of the module is shown in Figure 1.[8]

Jetson series GPUs have notably been used in various embedded computer vision and robotics tasks, due to their small form factor and low power

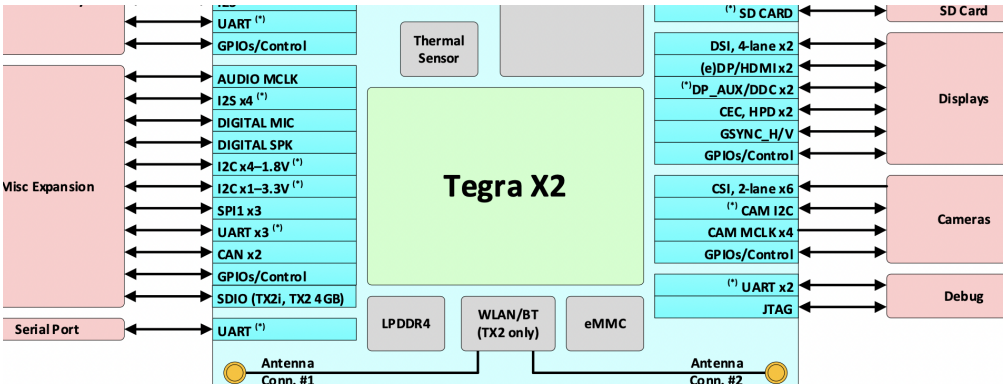

Figure 1: Block diagram of the TX2i module.

consumption, the latter of which is offered by a shared memory space between the GPU/CPU. Although virtually no commercial GPUs are intended for use in space—and such use cases void their warranties—the TX2i contains upgrades over its TX2 counterpart that permit its use in industrial grade environments, rendering it a relatively suitable candidate for on-board processing.[9]

With recent interest in flying a Jetson GPU, various radiation tests have been presented and published on the TX2i and related modules over the past two years. Slater et. al. tested the Jetson Nano under various TIDs (around 20 krad) and concluded that, with a 1/10 inch of aluminum shielding, the SOM could be expected to last through at least a 1.5–2 year mission.[6]

Heistand et. al. from the Johns Hopkins Applied Physics Laboratory (APL) conducted both TID and SEE tests on the TX2i module. The former test revealed that the TX2i could handle moderate radiation dosage (23 krad) before failure while powered on, a notably higher dose when off (45 krad), and lower doses (9.7–20 krad) during soft and hard power cycles—the latter of which had extremely high susceptibility. Proton SEEs caused multiple forced reboots before eventually resulting board failure. From the collected data, the APL team predicts a 57% and 73% survivability rate in LEO at solar minimum and maximum, respectively, without shielding or mitigation techniques. Additionally, while the majority of logged software events were CPU and/or memory errors, the majority of TX2i failures—including all permanent failures—were linked to flash errors (read/write).[10]

In orbit, while many of these issues can be mitigated through physical shielding, the TX2i still remains a weak point of future missions. This calls for software-level tolerance as a final line of defense against SEEs.

OPERATING SYSTEM DESIGN

Out of the box, the Jetson TX2i runs Linux for Tegra (L4T), a fork of the commercial Linux distribution Ubuntu 16.04. In collaboration with the DART team at APL, we develop Space Operating Linux (SOL): an open-source, space-faring alternative operating system to L4T to run on future Jetson-bearing missions with a primary focus on TX2i missions. SOL will first fly on the University of Georgia Small Satellite Research Laboratory's (SSRL) Multiview Onboard Computational Imager (MOCI) mission. The SSRL intends for MOCI to be a proof-of-concept for the feasibility of GPU-based AI and computer vision in space, and the mission is particularly motivated by the reduction of data downlink that is afforded by onboard data processing. The mission will provide integral data on GPU performance in the conditions of LEO and an open-source payload software suite, both of which may be used in the payload design of future, larger-scope spacecraft missions.

The primary goal of SOL is to increase the payload's reliability in orbit by providing software-level radiation mitigation and real-time scheduling, while also maintaining the functionality necessary for conducting AI-based missions. In particular, we make efforts to decrease reliance on flash eMMC memory, which has been shown in radiation tests to be the most vulnerable

Yocto and OS Minimization

SOL is primarily developed under the Yocto framework, an open-source project used to construct custom embedded Linux distributions from the ground-up. The fundamental units for Yocto

development are BitBake recipes, each of which provides instructions on retrieving, patching, building, and installing a particular software package for the target device.[11] Related recipes are grouped into layers, such as `meta-tegra`, which contains L4T-based software recipes intended for Jetson devices.

The `meta-sol` layer is developed on top of `meta-tegra` to provide SOL's custom software packages and patches. By doing so, SOL contains only the Tegra software packages that are essential in flight, as well as any necessary modifications to the existing software. Additionally, many existing Unix utilities are replaced by their alternatives in BusyBox, which provides lightweight implementations of such commands. These measures inherently minimize the size of the operating system and file system, hence reducing our usage of flash memory space.

Bootloader Redundancy

While a major goal of SOL is to reduce flash memory usage, the eMMC card is the device's only form of non-volatile memory and hence must be used for any permanent storage. Therefore, redundancy measures are necessary to ensure reliability of essential sectors of flash storage, particularly from corruption caused by accumulated SEEs such as bit-flips. At the most granular level, NVIDIA provides A/B redundancy, where all partitions are duplicated so that, on-boot, if the "A" side fails, the "B" side can be loaded instead.

SOL seeks to emulate triple modular redundancy (TMR) on the single eMMC card for the kernel and root file system. This method was originally proposed by Adams et. al.[12] but was since expanded to include redundancy measures beyond the kernel, motivated by the fact that corruption to the file system itself could hinder the kernel image from being accessed altogether. To achieve this, essential files for boot that are conventionally stored in the root file system—the kernel, device tree, and initial RAM disk image, as well as a tarball of the root file system itself—are directly written as binary large objects (BLOBs) into specified points of the main file partition, which is then triplicated. Each BLOB is stored along with an associated MD5 checksum to easily check for corruption, and there is a fifth, 60-byte `info` BLOB at the start of the partition that lists the exact file sizes of the following four. An example configuration is outlined in Figure 2, although exact size allocations are dependent on the build and determined within BitBake configurations. Aside from essential software correction, this partition is intended to be updated irregularly, and any temporary data or configuration files needed across boots shall be written to other data partitions.

File	Offset (# Blocks)	Max Size (# Blocks)
info	0	1
info (hash)	1	1
kernel image	2	90,000
kernel image (hash)	90,002	1
device tree BLOB	90,003	2,000
device tree BLOB (hash)	92,003	1
initial ram disk	92,004	5,000
initial ram disk (hash)	97,004	1
root file system TAR	97,005	1,500,000
root file system TAR (hash)	1,597,005	1

Figure 2: Example layout of one of three identical partitions.

The final stage bootloader for NVIDIA Jetson devices, U-Boot, is responsible for loading the kernel and booting into the initial RAM disk. U-Boot is conveniently open-source,[13] and patches can be applied within the Yocto source tree. Hence, SOL is able to apply custom TMR operations on-boot beginning at this stage of the boot process, with a patch that makes use of the custom partition layout. To load each BLOB—aside from the root file system, which is not needed at this stage—U-Boot checksums each of the three versions across the triplicated partition. If any of the checksums match the stored sum, the BLOB in that partition is assumed not to be corrupted and can be loaded. In the case that all three checksums are mismatched, bit voting is used to reconstruct the BLOB from the three corrupted versions, a method that is probabilistically resilient to bit-flips and potentially minor sector failures. A bit voting logic gate is illustrated in Figure 3. Note that, in order to speed up the boot process, which as discussed earlier is a vulnerable state for the SOM,[10] the reconstruction is only used in volatile memory in U-Boot—not yet written back to flash—for the sake of time. Once this process has been completed for each of the four BLOBs, U-Boot boots into the initial RAM disk, using the RAM disk image, the device tree, and the kernel image.

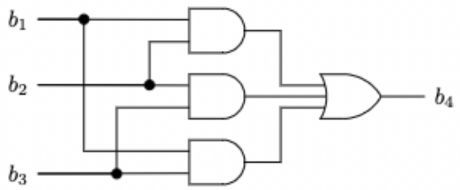

Figure 3: A simple bit voting logic gate using AND and OR blocks. The output is the majority "vote" of the three input bits.

Once booted into the initial RAM disk, the same logic is executed with a few key differences. First,

this uses all five BLOBs, as the root file system is mounted at this stage. Additionally, instead of stopping checksums once a valid BLOB is found, all three checksums are performed so that any corrupt BLOBs can be corrected in flash. With the kernel loaded at this stage, this can be sped up via multi-processing, which is discussed further in a later section.

RAM-based File System

Upon entry to the initial RAM disk, a temporary file system (tmpfs) is initialized in RAM. Then, after the emulated TMR process detailed above, the root file system is extracted from its tarball into the RAM disk. Hence, all reads from and writes to the root file system occur in volatile memory, greatly reducing flash usage post-boot.

Real-Time Linux Patch

In spacecraft processors, real-time scheduling is generally necessary to ensure that tasks are executed when expected. The `meta-sol` layer includes the `PREEMPT_RT` patch that enables the real-time scheduling paradigm on the Linux kernel. Hence, software commands may be sent over the mission's payload interface without disrupting the OBC's real-time operations.

CONCERNS & POTENTIAL SOLUTIONS

Volatile Memory

There are two main drawbacks to using a RAM-based file system. First is the fact that data is stored in volatile memory, so it is lost whenever the device is powered down. In order to update the root file system in flash, it must be re-archived, checksummed, and written to the three redundant partitions, which is a timely process. However, since many files are not mission critical, we offer two separate partitions mounted at `/config` and `/data` for temporary storage that is necessary to persist through reboots, such as intermediate data products. Currently, these partitions contain EXT4 file systems with no built-in redundancy, so the only protections offered on these partitions are those offered by simple file system consistency checks. In the future, missions may opt into another file system such as ZFS that incorporates redundancy, although this would increase system complexity even further.

Limited RAM

The second potential issue is the limited amount of RAM that remains for computation. Since a constant amount of the available RAM must be reserved for the file system itself, this restricts the amount of memory space for use in data processing, which is especially suboptimal for AI-based missions. To remedy this, the root file system size is reduced as much as possible, with only necessary utilities for core functionality kept in storage, as has been done through Yocto and Busybox. In MOCI's case, we expect a total of six of the original eight gigabytes of RAM available for normal operations, which requires the team to carefully design and monitor the memory usage of our computer vision algorithms.

Boot Time

The largest concern regarding the induced complexity of a redundant boot scheme is the increased boot time. This process currently takes over seven minutes without any optimization. The majority of this time is spent from reading the file system into RAM and performing checksums. Since the kernel is available at this point, optimization such as multi-threading can be employed to speed up the process. With such adjustments, we intend to decrease the boot time to under two minutes. Nevertheless, this is a relatively long boot time that must be taken into account during mission operations and planning. If necessary, additional speed-up may be granted by splitting the tarball into multiple BLOBS of smaller size, each with its own hash, and parellelizing even further, or by moving all non-essential validity checks to a chron job post-boot.

TESTING

Due to problems in the supply chain related to the ongoing COVID-19 pandemic, we currently lack the number of TX2i devices required for additional radiation tests. Hence, our current tests simply simulate bit flips in flash to ensure proper control flows and expected fault tolerance. As these devices become available within the next few months, we intend to replicate APL's SEE tests on SOL-equipped TX2i devices in order to determine how much the provided decrease on flash memory reliance improves the expected operational lifetime in orbit.

CONCLUSION

Although no major conclusions can be drawn until radiation tests are complete, there are a few ma-

jor takeaways from the design process. The primary tradeoff our team has experienced in terms of implementing boot-level redundancies and moving to a RAM based file system has been one of complexity versus reliability. That is, reliability of the machine can come at a cost that, if left unchecked, undermines the functionality of the mission itself. Such examples range from limiting memory for AI computation to slowing down the boot time—and hence initial response time—of the payload. For MOCI in particular, our team finds a comfortable middle ground in such scenarios, minimizing boot time and static RAM usage as much as possible without comprising the integrity of essential software in our root file system. However, we leave these decisions as programmable options within the open-source SOL repository for future developers to integrate with payloads depending on their individual mission requirements.

References

[1] JN Maki, D Gruel, C McKinney, MA Ravine, M Morales, D Lee, R Willson, D Copley-Woods, M Valvo, T Goodsall, et al. The mars 2020 engineering cameras and microphone on the perseverance rover: A next-generation imaging system for mars exploration. *Space science reviews*, 216(8):1–48, 2020.

[2] Neil Abcouwer, Shreyansh Daftry, Tyler del Sesto, Olivier Toupet, Masahiro Ono, Siddarth Venkatraman, Ravi Lanka, Jialin Song, and Yisong Yue. Machine learning based path planning for improved rover navigation. In *2021 IEEE Aerospace Conference (50100)*, pages 1–9, 2021.

[3] Daniela Girimonte and Dario Izzo. Artificial intelligence for space applications. In *Intelligent Computing Everywhere*, pages 235–253. Springer, 2007.

[4] Armen Poghosyan and Alessandro Golkar. Cubesat evolution: Analyzing cubesat capabilities for conducting science missions. *Progress in Aerospace Sciences*, 88:59–83, 2017.

[5] Richard Vuduc and Jee Choi. *A Brief History and Introduction to GPGPU*, pages 9–23. Springer US, Boston, MA, 2013.

[6] Windy Slater, Nayana Tiwari, Tyler Lovelly, and Jesse Mee. Total ionizing dose radiation testing of nvidia jetson nano gpus. pages 1–3, 09 2020.

[7] Gautam D. Badhwar. The radiation environment in low-earth orbit. *Radiation Research*, 148(5):S3–S10, 1997.

[8] Jetson tx2 series module data sheet. `https://developer.nvidia.com/embedded/downloads`, 04 2021. [Online].

[9] Jackson O Parker. Design and implementation of a 6u cubesat for low earth orbit computer vision. Master's thesis, University of Georgia, 2020.

[10] Christopher Heistand, Sarah Katz, and Amanda Voegtlin. Nvidia jetson tx2i radiation report. Single Event Effects (SEE) Symposium / Military and Aerospace Programmable Logic Devices (MAPLD) Workshop, 2020.

[11] Yocto project reference manual. `https://docs.yoctoproject.org/current/ref-manual/index.html`. [Online].

[12] Caleb Adams, Allen Spain, Jackson Parker, Matthew Hevert, James Roach, and David Cotten. Towards an integrated gpu accelerated soc as a flight computer for small satellites. In *2019 IEEE Aerospace Conference*, pages 1–7. IEEE, 2019.

[13] Das u-boot – the universal boot loader. `https://www.denx.de/wiki/U-Boot`, 11 2019. [Online].