EasyChair Preprint

№ 10267

Performance Evaluation of the Radiation-Tolerant NVIDIA Tegra K1 System-on-Chip

Derrek Landauer and Tyler Lovelly

May 25, 2023

Performance Evaluation of the Radiation-Tolerant NVIDIA Tegra K1 System-on-Chip

Derrek C. Landauer [1,2] and Tyler M. Lovelly [1,3]
[1] The University of New Mexico, Albuquerque, New Mexico, U.S.A.
[2] BlueHalo, Kirtland Air Force Base, New Mexico, U.S.A.
[3] Air Force Research Laboratory, Kirtland Air Force Base, New Mexico, U.S.A.
derrek.landauer@bluehalo.com, tyler.lovelly.1@spaceforce.mil

Abstract*—Radiation-hardened (rad-hard) processors are designed to be reliable in extreme radiation environments, but they typically have lower performance than commercial-off-the-shelf (COTS) processors. For space missions that require more computational performance than rad-hard processors can provide, alternative solutions such as COTS-based systems-on-chips (SoCs) may be considered. One such SoC, the NVIDIA Tegra K1 (TK1), has achieved adequate radiation tolerance for some classes of space missions. Several vendors have developed radiation-tolerant single-board computer solutions targeted primarily for low Earth orbit (LEO) space missions that can utilize COTS-based hardware due to shorter planned lifetimes with lower radiation requirements. With an increased interest in space-based computing using advanced SoCs such as the TK1, a need exists for an improved understanding of its computational capabilities. This research study characterizes the performance of each computational element of the TK1, including the ARM Cortex-A15 MPCore CPU, the NVIDIA Kepler GK20A GPU, and their constituent computational units. Hardware measurements are generated using the SpaceBench benchmarking library on a TK1 development board. Software optimizations are studied for improved parallel performance using OpenMP for CPU multithreading, ARM NEON for single-instruction multiple-data (SIMD) operations, Compute Unified Device Architecture (CUDA) for GPU parallelization, and optimized Basic Linear Algebra Subprograms (BLAS) software libraries. By characterizing the computational performance of the TK1 and demonstrating how to optimize software effectively for each computational unit within the architecture, future designers can better understand how to successfully port their applications to COTS-based SoCs to enable improved capabilities in space systems. Experimental outcomes show that both the CPU and GPU achieved high levels of parallel efficiency with the optimizations employed and that the GPU outperformed the CPU for nearly every benchmark, with single-precision floating-point (SPFP) operations achieving the highest performance.

I. Introduction

On-orbit data processing is crucial for space missions that require high computational performance, but achieving this capability is challenging due to the harsh radiation environments in space. Commercial off-the-shelf (COTS) processors with embedded graphical processing units (GPUs) typically offer better performance than radiation-hardened (rad-hard) processors, but the latter is more reliable in extreme environments. Previous research has identified the computational limitations of rad-hard processors [1–3], and other studies have recommended alternative architectures for better performance and energy efficiency [4, 5].

In the past, spaceflight processors were unsuitable for on-orbit data processing, leading to compromises in terms of computational capabilities, such as using low-resolution instruments for data collection and relying on ground stations for further processing [6]. However, the increasing need for improved onboard processor performance for tasks such as autonomous satellite collision avoidance [7] and the use of higher resolution sensors for various applications [5] have made this a crucial area of focus for evolving space missions.

There are concerns about the ability of high-performance processors to operate reliably and predictably in space while achieving high performance per watt and delivering high computational performance. Although space commercialization is being pursued to drive investment and technological progress, enhanced rad-hard solutions typically require government support and many years of research and development to achieve their objectives [8]. However, the feasibility of on-orbit data processing may not be as questionable as previously thought [9], and the cost-effectiveness of deploying equipment with a shorter expected lifespan for certain missions, compared to previous missions, can be improved by reducing launch expenses [10]. For these reasons, using more capable radiation-tolerant COTS equipment for certain tasks may be more beneficial than a purely rad-hard system.

Rad-hard processors are designed to withstand radiation exposure in space and other harsh environments. These processors often have re-engineered architectures to achieve this capability, but the radiation-hardening techniques can degrade their performance compared to commercial processors. Additionally, the base architecture of rad-hard processors may be outdated by the time they are released. Despite these limitations, rad-hard processors have proven reliable for long-term use in space missions. For example, the BAE RAD750 is a rad-hard processor with flight heritage; it was released in 2001 and first launched on NASA's Deep Impact mission in 2005. The RAD750 is present in more recent NASA missions, such as the Perseverance rover launched in 2020 and the James Webb Space Telescope launched in 2021 [11]. Although these missions use legacy processor cores capable of only approximately 0.27 giga-operations per second (GOPS) [2], the newly publicized BAE RAD510 SoC runs at a maximum rated 1.3 GOPS [12]. For perspective, the RAD510 performs at two orders of magnitude below COTS processors from almost a

decade ago and five orders below the recently released NVIDIA Jetson AGX Orin, which NVIDIA purports to deliver up to 275 tera-operations per second (TOPS) [13].

Due to the performance gap between specialized and COTS processors, COTS processors may become essential to supporting short-duration Low Earth Orbit (LEO) missions that require robust onboard computation [14]. Achieving this will likely require advancements in both software and hardware. For example, there is ongoing research on developing image processing algorithms that are resilient to adverse radiation effects such as single event upsets (SEUs) [15, 16], as well as modern load balancing strategies to improve the system's performance per watt [17, 18]. Private industry has also addressed some challenges due to the overlapping concerns of the power-conscious embedded systems market. Modern embedded GPUs can achieve better energy efficiency compared to Central Processing Units (CPUs) for calculations that involve repetitive tasks with few instructions, low memory access requirements, and can be computed concurrently [19, 20].

The NVIDIA Tegra K1 (TK1) System-on-Chip (SoC) has been reported to be radiation-tolerant [21]. Despite the aging Kepler architecture, previous research has demonstrated the potential of the TK1 for modern space applications. For example, the TK1 has been used for tasks such as 3D scanning [22], aircraft detection [23], autonomous robotics [24], collision avoidance [25], encryption speedup [26], fully convolutional networks [27], object detection [28, 29], image processing [30], Synthetic Aperture Radar (SAR) imaging [31], and target tracking [32, 33].

Radiation-tolerant single-board computers (SBCs) based on the TK1 have recently been developed to support low Earth orbit (LEO) missions, such as the Innoflight CFC-500 and Ibeos Edge [34, 35]. These boards represent a shift towards commercial off-the-shelf (COTS) processors in a domain previously limited to specialized rad-hard devices. The development of radiation-tolerant SBCs reflects an increasing interest in using COTS processors for on-orbit space applications, which motivates this study to further evaluate the TK1 for such applications. In this study, the architecture of the TK1 is evaluated in terms of computation performance.

II. Background and Related Research

This study aims to characterize the theoretical performance of the TK1 for space applications, measure realizable performance through hardware benchmarking, and conduct a final analysis of the results. This section describes the TK1 SoC and the notions of ideal and realizable computational performance. First, a general description of the TK1 architecture is given, followed by a brief examination of the methods for calculating and measuring computational performance. The following information lays the foundation for subsequent sections that compare the realizable performance of the TK1.

The TK1 is an intricate, high-performance SoC that comprises an ARM Cortex-A15 MPCore CPU and an NVIDIA Kepler GK20A GPU, shown in Fig. 1. The GPU can parallelize

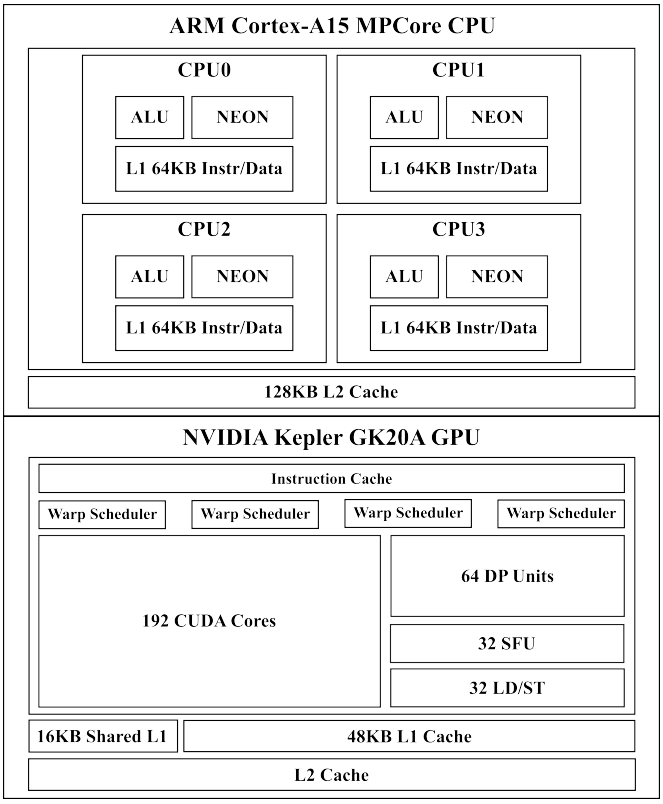


Fig. 1: NVIDIA Tegra K1 SoC model [36–39].

massive computations containing many repetitive and independent subtasks of which many instructions execute within a single cycle, reducing CPU load [40, p. 4-2]. The TK1 also uses a unified memory model in which the CPU and GPU share the main system memory. This design significantly reduces memory transfer overhead and further enables the GPU to act as a performance accelerator.

A. ARM Cortex-A15 MPCore CPU

The TK1 CPU comprises four cores optimized for high performance and operate at a maximum frequency of 2.07 GHz, with an additional power-saving core [41]. However, the performance analysis does not include the additional core as it is prohibited from being used concurrently with the central cores, as per the design of the ARM Cortex [42]. Each CPU core has two major computational units relevant to performance metrics: the Integer Execute Unit and the Advanced SIMD (NEON) and Vector Floating Point (VFP) units. The Integer Execute Unit includes two symmetric Arithmetic Logic Unit (ALU) pipelines, an integer multiply-accumulate (MAC) pipeline, integer divide hardware, branch and condition codes resolution logic, and result forwarding and comparator logic. The NEON and VFP units are distinct but share registers to support Single Instruction Multiple Data (SIMD) and floating-point operations. Each NEON register can store up to 128 bits of vectorized data of the same data type; for example, a NEON register can hold four 32-bit SPFP numbers or eight 16-bit signed integers [42, Fig. A2-3]. This feature allows for multiple operations per

clock cycle by issuing a single instruction on multiple data. The purpose of the VFP unit is to perform scalar floating-point operations. The Advanced SIMD unit does not support double-precision floating-point (DPFP) types, but the VFP unit does. However, VFP vector support is deprecated and not implemented in hardware on the ARMv7 [43, p. 1-2], so the VFP unit can only support single instruction single data (SISD) operations with DPFP data types. The SIMD unit can execute two instructions simultaneously, known as dual issuing, when executing a NEON load/store instruction and a NEON computational instruction [44, p. 5-5]. However, this study focuses on computational performance and ignores SIMD dual issuing. Additionally, according to [40, p. 7-5], the number of cycles required for NEON instructions is not specified, and some instructions may require several cycles and could vary between implementations. Due to a lack of information from NVIDIA, this study assumes one cycle per instruction, potentially inflating the theoretical performance results.

B. NVIDIA Kepler GK20A GPU

The GPU incorporates a single Kepler Streaming Multiprocessor (SMX), which performs the computations. The SMX launches a CUDA kernel function, implemented as an array of concurrent threads known as the Single Instruction Multiple Thread (SIMT) execution model (an extension to Flynn's Taxonomy) [45]. The underlying SMX components assessed in this study are the Streaming Processors (also referred to as CUDA cores), Double Precision (DP) units, and Special Function Units (SFU). The SMX, based on the Kepler microarchitecture, contains 192 32-bit CUDA cores capable of a maximum frequency of 852 MHz. Each core executes a kernel instance and participates in a bundle of 32 threads called a warp, where a group of eight CUDA cores spends four clock cycles executing a SIMT. The Kepler architecture can issue 1024 threads per block, where each thread block is issued to a CUDA core. The TK1 includes 64 DP units to support DPFP computations, as the CUDA cores are restricted to SPFP data. The TK1 has 32 SFUs that are hardware accelerators for SPFP intrinsic functions [46]. The SFU implements basic arithmetic and transcendental functions with varying rounding criteria (e.g., round up, round down, and round nearest even).

C. Quantifying System Performance

The TK1 is a complex SoC that integrates both a CPU and GPU, each of which has a distinct architecture that presents unique challenges when evaluating performance. The device's computational performance varies depending on the data type, operation, and memory usage. To adequately address performance variation between computational units and to determine the theoretical maximum performance, this study adopts the methodology described by [47, 48]. This study's approach to measuring device performance, known as computational density (CD), is shown in Eq. 1.

$$CD = f \times \sum_{i=1}^{n} \frac{N_i}{CPI_i} \tag{1}$$

where f is the device frequency, N is the number of execution units, and CPI is the number of cycles per instruction. The summation operation over i accounts for SIMD architectures, where n is the number of discrete values for which a given data type can fit into a SIMD register. The ARM CPU has four high-performance cores, each with a dual pipeline integer unit, a MAC unit, and NEON/VFP units. Therefore, CPU CD is represented by Eq. 2.

$$CD_{CPU} = CD_{ALU} + CD_{MAC} + CD_{NEON/VFP} \tag{2}$$

The GPU has a maximum core frequency of 852 MHz, 192 CUDA cores, also called Streaming Processors (SP), 64 DPs, and 32 SFUs; therefore, the GPU CD is expressed by Eq. 3.

$$CD_{GPU} = CD_{SP} + CD_{DP} + CD_{SFU} \tag{3}$$

The total SoC computational performance is composed of the total CD of the GPU and CPU, as shown in Eq. 4.

$$CD_{SoC} = CD_{CPU} + CD_{GPU} \tag{4}$$

D. Measuring System Performance

Assessing processor performance poses a challenge due to memory transfers, pipeline hazards, and additional overhead. Furthermore, comparing the performance claimed by manufacturers can prove difficult due to the need for more transparency in their metrics. A viable approach for evaluating processor performance in space-based computing involves benchmarking software, such as NSF SHREC SpaceBench. This software assesses the execution time of a diverse array of computational kernels, employing varying parameters on processors specifically designed for space applications [49].

The benchmark tasks are meticulously constructed to appraise the performance of various combinations of parameters, encompassing the compute unit, kernel function, problem size, and data type. These tasks focus on specific devices and units, including the CPU's integer unit and SIMD/VFP unit, as well as the GPU's CUDA cores, DP units, and SFU units. SpaceBench executes various computational kernels, such as matrix addition, multiplication, transposition, and convolution.

Distinct implementations of these kernels cater to different computing architectures, including the general-purpose CPU pipeline, OpenMP for multithreading, NEON for SIMD, GPU for SIMT, and GPU SFU for low-precision SPFP acceleration. These kernels comprise $m \times m$ matrix operations where m can be 128, 256, 512, 1024, or 2048, and they support a range of data types, such as SPFP and DPFP, 8-bit, 16-bit, 32-bit, and 64-bit integers. Nonetheless, it is important to acknowledge that each compute unit may only support a subset of these data types.

III. Methodology

This section outlines the hardware and software configurations used to execute SpaceBench, and the analysis tools used to verify processor utilization and report runtimes in GOPS.

A. Experimental Setup

The NVIDIA Jetson TK1 development kit was used to examine the SoC. The board was installed with the latest TK1-supported release of Linux for Tegra (L4T) 21.8, which provides Ubuntu 14.04 with the GNU/Linux kernel version 3.10.40, and CUDA toolkit 6.5 [50, 51]. The graphical user interface (GUI) was disabled for all testing (i.e., headless mode). The CPU governor was set from the default "interactive" mode to "performance" mode before benchmarking to reduce frequency ramp-up time. After doing so, the operating system reported a maximum frequency of 2.32 GHz, rather than the previously reported upper limit of 2.07 GHz. SpaceBench was executed via a terminal BASH script, and the output was redirected to a comma-separated value (CSV) file.

B. Analysis Tools

The NVIDIA Profiler helped identify potential SpaceBench bottlenecks. The most time-consuming tasks are usually associated with memory allocation, which is not included in the recorded runtime. The next longest task is when the system synchronizes threads, impacting runtime calculations. The CUDA Occupancy Calculator helps ensure proper GPU settings to utilize all available cores when appropriate [52]. Several parameters contribute to GPU occupancy, such as the NVIDIA microarchitecture, L2 partition size for shared memory, the number of threads per block, and registers per thread. The Table of Processes (`top`) command monitored the CPU utilization to confirm processor saturation.

C. Calculating Theoretical Performance

The CPU integer ALU is a dual pipeline and, therefore, capable of performing two instructions per cycle [36, p. 2-3]. The ALU performance, CD_{ALU}, can be found using Eq. 1, where $N = 4$ (i.e., one integer unit per core), and $CPI = 0.5$ (since CPI is the inverse of instructions per cycle). The MAC unit is a single pipeline capable of one instruction per cycle. CD_{MAC} is found using Eq. 1, where $N = 4$ (i.e., one MAC per core), and $CPI = 1$. The SIMD unit executes a single instruction on 128 bits of vectorized data; accordingly, the effective operations per cycle differ per the used data type size. $CD_{NEON/VFP}$ is found using Eq. 1, where $N = 4$ (i.e., one SIMD unit per core), $CPI = 1$, and n is the amount of numbers for which a given data type can fit into a 128-bit SIMD register. For example, if the SIMD performs 8-bit integer operations (e.g., add, shift, multiply), the unit can support up to 16 operations per instruction. Also, recall that the SIMD unit only supports up to 32-bit vectorized data types, so only one DPFP can be used per register. The theoretical CPU performance given the frequency value provided by the TK1 datasheet is shown in Table I. The performance of the CUDA cores, CD_{SP}, is found using Eq. 1, where $N = 192$, and $CPI = 1$. The total performance of the DP units, CD_{DP}, is also found using Eq. 1, where $N = 64$, and $CPI = 1$. Finally, the performance of the SFU units, CD_{SFU}, is found using Eq. 1, where $N = 32$, and $CPI = 1$. Refer to Table I for the GPU performance calculations. The total performance of the TK1 is found by simply applying Eq. 4. See Table I for a comprehensive theoretical performance summary reported in GOPS.

D. Software Optimizations and Enhancements

The differing architectures of the underlying TK1 processors require different procedures to implement comparable algorithms, and application programming interfaces (APIs) are needed to utilize specialized accelerators. Accordingly, SpaceBench uses three crucial software extensions to target underlying devices: OpenMP for CPU multithreading, NEON for SIMD, and CUDA C for the GPU. OpenMP is an API to enable CPU multithreading. SpaceBench uses OpenMP to execute 1-12 threaded kernel function runs. NEON is an advanced SIMD computer architecture extension for ARM Cortex A-series processors used for vectorized computation. SpaceBench uses a C API to utilize NEON intrinsic functions, which interface the ARM SIMD unit for computation speedup. CUDA is a programming model created for NVIDIA GPUs, and SpaceBench uses CUDA C (an extension to C/C++) to utilize the GPU. Due to its complicated memory hierarchy and thread issuing strategy, optimizing CUDA code requires more fine-tuning than the previously implemented enhancements. The hardware capabilities are fairly represented by considering additional factors, such as pipeline stalling caused by cache misses. For example, multiplication, transposition, and convolution-blocking algorithms are implemented to compare against naive algorithm implementations.

E. Measuring Realizable Performance

Characterizing the TK1's computational performance provides a sensible assertion of its capabilities [49]. This study focused on the TK1's processor computational performance instead of pure instruction performance. A single operation in this context is a mathematical computation consisting of addition and multiplication, whether it is an add instruction and a multiply instruction or only one MAC instruction. This method focuses the metrics on the core calculation performance rather than any programmatic overhead for assigning values to counters and other analogous program instructions.

SpaceBench was used to measure the realizable performance of the TK1 by recording the execution time for functions crafted to utilize the CPU and GPU independently. The average runtime over several iterations for each computational kernel was then converted to GOPS, the normalized number of operations per runtime. The transpose function takes exception to this criteria since the whole process consists only of memory transfers; therefore, an operation in this context is simply the finalized relocation of a value from the input matrix to the output matrix.

IV. Results and Analysis

To measure the computational performance of the TK1, the processor benchmarking aimed to measure the number of operations it could perform in a given unit of time. However, accurately comparing the performance of different processor architectures can be difficult due to the varying overhead of

TABLE I: Theoretical TK1 Performance

Processor	Unit	add, sub, shift					multiply					multiply & accumulate				
		Int8	Int16	Int32	SPFP	DPFP	Int8	Int16	Int32	SPFP	DPFP	Int8	Int16	Int32	SPFP	DPFP
ARM Cortex-A15 MPCore CPU	ALU/Int MAC	16.5	16.56	16.56	–	–	8.28	8.28	8.28	–	–	8.28	8.28	8.28	–	–
	NEON/VFP	132.48	66.24	33.12	33.12	8.28	132.48	66.24	33.12	33.12	8.28	132.48	66.24	33.12	33.12	8.28
NVIDIA Kepler GK20A GPU	SP	136.32	136.32	136.32	163.58	–	27.26	27.26	27.26	163.58	–	27.26	27.26	27.26	163.58	–
	DP	–	–	–	–	6.82	–	–	–	–	6.82	–	–	–	–	6.82
	SFU	–	–	–	27.26	–	–	–	–	27.26	–	–	–	–	–	–
NVIDIA Tegra K1 SoC	CPU+GPU	285.36	219.12	186.00	223.97	15.10	168.02	101.78	68.66	223.97	15.10	168.02	101.78	68.66	196.70	15.10

– unit, type, and instruction combination not supported.

implementing equivalent algorithms. In order to address this challenge, the runtime of various algorithms implemented on different compute units of the TK1 was measured, and the calculations only incorporated core mathematical operations required to solve each problem. This approach aids in preventing inflated performance metrics and ensures that implementation details do not distort the results.

A. Theoretical Performance

The following analysis relies on the estimated theoretical CPU performance shown in Table I and Fig. 2. The CPU's maximum performance decreases as the data type size increases because the SIMD registers are limited to 128-bits. Therefore, the CPU can achieve the highest performance by performing SIMD operations using the smallest data type, Int8. It is anticipated that Int16 will exhibit approximately half the performance of Int8 due to its increased memory requirements, while Int32 is expected to exhibit roughly half the performance of Int16. On ARMv7, NEON does not support DPFP operations, and VFP vectorization is deprecated. As a result, the CPU's DPFP performance using the VFP unit is limited to one instruction per DPFP per cycle.

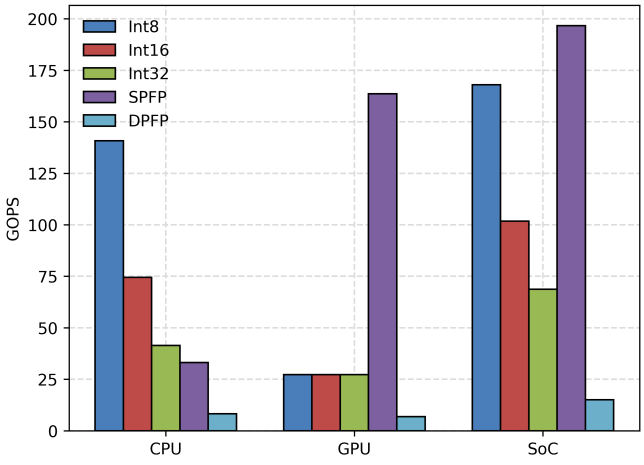


Fig. 2: TK1 theoretical performance for MAC operations

This study posits that Int8 and Int16 exhibit performance on par with Int32, which is estimated at 27 GOPS for SP units, as there is insufficient information available on the GPU's performance for Int8 and Int16 arithmetic instructions. Moreover, the GPU's SFUs for transcendental functions contribute an additional estimated performance of 27 GOPS, excluding rounding operations. While the GPU's 64 DP units have a more modest capability of performing approximately eight operations per cycle, they still achieve a respectable theoretical performance of around 6.8 GOPS for addition and multiplication tasks.

B. CPU Parallel Performance

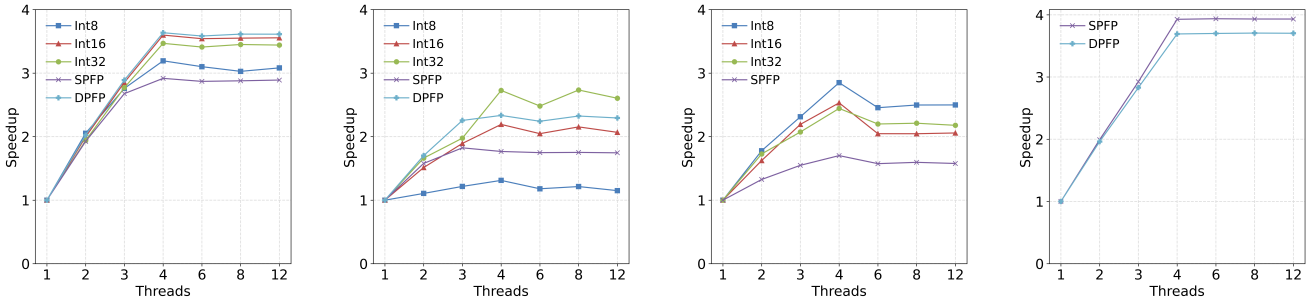
The parallel efficiency of TK1 CPU was evaluated for matrix multiplication using OpenMP and four different implementations: naive, blocking, NEON, and OpenBLAS. A single-threaded run served as a benchmark for comparison with additional multi-threaded tests for each implementation to evaluate their performance under varying thread numbers.

The results for the naive algorithm are shown in Fig. 3a. Near-linear speedup was observed until a thread occupied each core, after which performance plateaued. Int32 data type performed best for the smallest problem sizes; however, as the problem size increased, a decline in performance was observed due to increased cache misses.

The performance of the blocking algorithm is shown in Fig. 3b, where it was observed that the algorithm scaled up as more threads were assigned to it until reaching a maximum performance when each core was assigned a single thread. The Int32 data type performed the best, achieving a maximum of approximately 2.75 GOPS for the 2048×2048 problem size. The SPFP performance reached its limit at the 1024×1024 problem size, while the DPFP performance reached its limit at the 256×256 problem size. The performance of the integer benchmark decreased as the problem size grew, most likely due to increased cache misses.

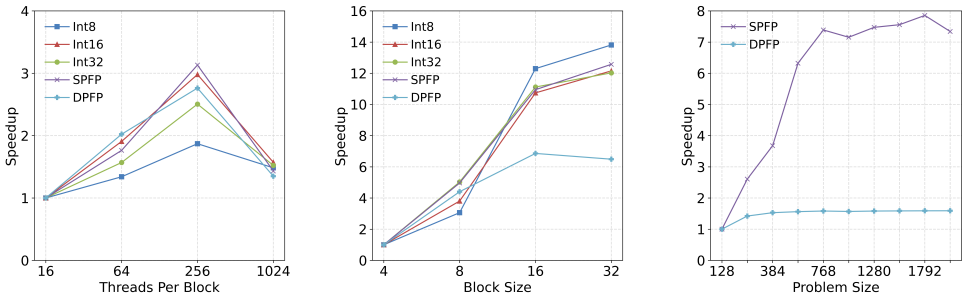
The OpenBLAS library's matrix multiplication implementation was the highest-performing algorithm tested on the CPU, supporting only SPFP and DPFP data types. The outcomes of this implementation are displayed in Fig. 3d, where it achieved a maximum performance of nearly 8 GOPS for SPFP and about 6.5 GOPS for DPFP, with a near-linear increase in performance as more threads were added until all available cores were utilized.

The parallel efficiency of the TK1 CPU was evaluated for matrix multiplication using four different implementations: naive, blocking, NEON, and OpenBLAS. The OpenBLAS library performed best but did not improve further when more than four threads were added. The results show that the although naive algorithm had a greater percent increase in performance



(a) naive algorithm (b) blocking algorithm (c) NEON acceleration (d) OpenBLAS library

Fig. 3: ARM Cortex-A15 MPCore CPU parallel speedup



(a) naive algorithm (b) blocking algorithm (c) CuBLAS library

Fig. 4: NVIDIA Kepler GK20A GPU parallel speedup

per added thread, the blocking algorithm performed better overall. However, it was observed that the blocking algorithm did not achieve the desired level of scalable performance. In future studies, similar blocking algorithms will be implemented to improve temporal locality and reduce cache misses, thereby better demonstrating the performance of the CPU pipeline.

C. GPU Parallel Performance

In the current version of SpaceBench, the CUDA programming model is utilized to take advantage of the parallel processing capabilities of the GPU. However, an unresolved bug impacted the execution of the naive and blocking algorithms on problem sizes of 2048×2048 and larger. Consequently, the results presented in the subsequent sections are limited to matrix multiplication implementations with a maximum problem size of 1024×1024. The results shown in Fig. 4a indicate that the GPU performance improved as the number of threads per block increased. However, utilizing the maximum value of 1024 threads per block resulted in a decline in performance, suggesting that fully occupying the GPU does not guarantee improved performance. The blocking algorithm performed well with larger problem sizes, but it did not have a linear increase in performance. The size of the blocks used in the algorithm also had a notable effect on performance, with larger block sizes resulting in better performance, as shown in Fig. 4b. However, it is challenging to quantify how much of the improvement can be attributed solely to the algorithmic block size in the current implementation, as it is coupled with the number of threads per block. The implementation of CuBLAS, a library of BLAS optimized for GPU acceleration, resulted in a significant improvement in speed for SPFP, as evidenced by the results shown in Fig. 4c. However, performance improvement stalled after reaching a problem size of 768×768. Meanwhile, the DPFP implementation saturated at a problem size of 256×256. Due to data transfer overhead, smaller problem sizes usually do not benefit from using the GPU. However, the GPU performed better than the CPU in all tests except DPFP, likely due to its unified memory model, which eliminates the need to transfer memory between the CPU and GPU. The GPU generally exhibits improved performance with increasing problem size, except in the case of DPFP, where the highest level of performance was achieved at a problem size of 256×256 and did not improve with further size increases. When creating algorithms for the GPU, it is crucial to consider various parameters carefully to attain optimal performance. For example, utilizing many threads may be straightforward, as demonstrated in the naive implementation, but other strategies may be more effective. The CUDA programming model provides a convenient way to implement a blocking algorithm using internal thread indexing variables, as used in the blocking algorithm. However, this can also affect device performance unexpectedly, making it challenging to isolate and identify the specific factors that

impact performance. Therefore, optimizing GPU resources and fine-tuning algorithm parameters are essential for obtaining the best results. The crucial point to remember is to utilize optimized libraries, such as CuBLAS, whenever possible, as they are designed to be high performing.

D. Realizable Performance

In Fig. 5, we present the results of all matrix multiplication benchmarks performed on the TK1 device. The best-case CPU performance for integer data types varied between 2.26 and 6.86 GOPS, with Int32 showing the lowest performance across all benchmarks. The CPU performance for SPFP and DPFP data types was significantly higher, reaching 7.76 and 6.83 GOPS, respectively. On the other hand, the GPU integer performance remained relatively consistent, with a range of 8.47 and 9.15 GOPS, and again Int32 demonstrated slightly lower performance than the other integer data types. The best-performing benchmarks on the GPU were SPFP and DPFP, with a maximum performance of 97.46 and 6.40 GOPS, respectively.

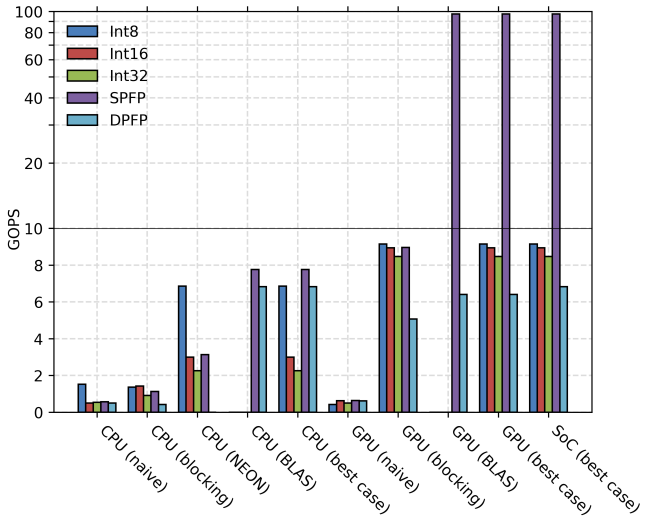


Fig. 5: TK1 realizable performance

TABLE II: Realizable TK1 Performance

Processor	Implementation	Performance (GOPS)				
		Int8	Int16	Int32	SPFP	DPFP
ARM Cortex-A15 MPCore CPU	naive	1.52	0.50	0.54	0.56	0.50
	blocking	1.36	1.42	0.92	1.13	0.42
	NEON	6.86	3.00	2.26	3.14	–
	BLAS	–	–	–	7.76	6.83
	best case	6.86	3.00	2.26	7.76	6.83
NVIDIA Kepler GK20A GPU	naive	0.42	0.63	0.50	0.64	0.62
	blocking	9.15	8.94	8.47	8.96	5.07
	BLAS	–	–	–	97.46	6.40
	best case	9.15	8.94	8.47	97.46	6.40
NVIDIA Tegra K1 SoC	best case	9.15	8.94	8.47	97.46	6.83

– unit, type, and instruction combination not implemented.

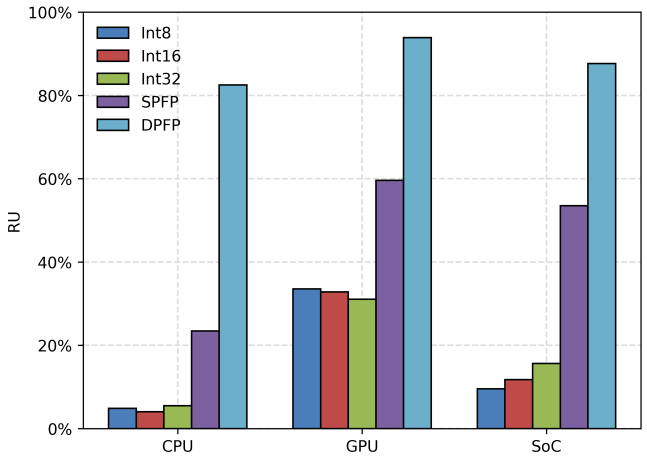


Fig. 6: TK1 realizable utilization

E. Realizable Utilization (RU)

The CPU and GPU performance results on different data types are shown in Fig. 6. The CPU performed better in DPFP operations but had a lower RU for all other data types. The OpenBLAS library had an RU of 23% for SPFP and 83% for DPFP, whereas the GPU's floating-point units showed strong performance, with the CuBLAS library achieving a RU of 60% for SPFP and 94% for DPFP. In contrast, the TK1 integer units had the lowest RU, with the CPU averaging 5% and the GPU at around 33%. This is due to the TK1's optimization for high SPFP performance, making it ideal for applications such as video processing, computer graphics, and scientific computations.

V. Conclusions and Future Research

In space-based computing, it is vital to consider the impact of extreme radiation environments on processor performance. Rad-hard processors are designed to be reliable in these conditions but may not offer the computational performance needed for next-generation space missions. As a COTS-based alternative, we explored the TK1 SoC, which has demonstrated the ability to withstand the radiation levels present in LEO. This study characterized the theoretical and realizable CPU and GPU performance to better understand the TK1's capabilities and determine its suitability for space-based computing applications. The TK1's GPU outperformed its CPU in nearly every benchmark, with SPFP operations achieving the highest performance. Based on these findings, it is generally preferred to use the GPU rather than the CPU for all data types, especially for SPFP operations. Optimizing software for each computational unit within the TK1's architecture, and using optimized libraries such as BLAS, can significantly improve performance and make it possible to effectively port applications to the TK1 and improve space systems' capabilities. The results of this study suggest that the TK1 may be a suitable alternative to traditional radiation-hardened processors for short-duration LEO space missions requiring enhanced computational performance. Future research directions will

include testing both the CPU and GPU simultaneously, adding new optimizations and computations to the analysis, and comparisons with other radiation-tolerant SoCs, such as the AMD Ryzen Embedded V1605B.

Acknowledgments

The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the U.S. government. Distribution Statement A: Approved for Public Release. Distribution is Unlimited. Public Affairs Release Approval #AFRL-2023-0405.

References

[1] R. Ginosar, "Survey of processors for space," *European Space Agency, (Special Publication) ESA SP*, vol. 701, Jan. 2012.

[2] T. M. Lovelly, "Comparative analysis of space-grade processors," Ph.D. dissertation, University of Florida, Gainesville, FL 32611, 2017. [Online]. Available: https://ufdcimages.uflib.ufl.edu/UF/E0/05/18/36/00001/LOVELLY_T.pdf.

[3] T. M. Lovelly and A. D. George, "Comparative analysis of present and future space-grade processors with device metrics," *Journal of Aerospace Information Systems*, vol. 14, no. 3, pp. 184–197, 2017. DOI: 10.2514/1.I010472.

[4] M. J. Cannizzaro, E. W. Gretok, and A. D. George, "RISC-V benchmarking for onboard sensor processing," in *2021 IEEE Space Computing Conference (SCC)*, 2021, pp. 46–59. DOI: 10.1109/SCC49971.2021.00013.

[5] L. Kosmidis, J. Lachaize, J. Abella, O. Notebaert, F. J. Cazorla, and D. Steenari, "GPU4S: Embedded GPUs in space," in *2019 22nd Euromicro Conference on Digital System Design (DSD)*, 2019, pp. 399–405. DOI: 10.1109/DSD.2019.00064.

[6] C. A. Jones *et al.*, "Recommendations to advance space trusted autonomy," in *ASCEND 2021*. DOI: 10.2514/6.2021-4219.

[7] European Space Agency, "AIKO: Autonomous satellite operations thanks to artificial intelligence," Jan. 28, 2019. [Online]. Available: https://www.esa.int/Applications/Technology_Transfer/AIKO_Autonomous_satellite_operations_thanks_to_Artificial_Intelligence (visited on Sep. 8, 2022).

[8] S. Chin, "DoD to invest up to $170 million in rad-hard foundry company," *Fierce Electronics*, Oct. 2019. [Online]. Available: https://www.fierceelectronics.com/electronics/dod-to-invest-up-to-170-million-rad-hard-foundry-company (visited on Aug. 9, 2022).

[9] D. Bhattacherjee, S. Kassing, M. Licciardello, and A. Singla, "In-orbit computing: An outlandish thought experiment?" In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*, ser. HotNets '20, New York, NY, USA: Association for Computing Machinery, 2020, 197–204, ISBN: 9781450381451. DOI: 10.1145/3422604.3425937.

[10] T. M. Harris and A. E. Landis, "Space sustainability engineering: Quantitative tools and methods for space applications," in *2019 IEEE Aerospace Conference*, 2019, pp. 1–6. DOI: 10.1109/AERO.2019.8741939.

[11] D. McComas, "Lessons from 30 years of flight software," NASA Goddard Space Flight Center, Tech. Rep., Sep. 2015. [Online]. Available: https://ntrs.nasa.gov/api/citations/20150019915/downloads/20150019915.pdf (visited on Sep. 8, 2022).

[12] BAE Systems, "RAD510 3U CompactPCI single-board computer," 9300 Wellington Road, Manassas, Virginia 20110-4122, Data Sheet. CS-21-D01 ES-C4ISR-071621-0157, Aug. 2021. [Online]. Available: https://www.baesystems.com/en-media/uploadFile/20211206201500/1434554723601.pdf.

[13] NVIDIA Corporation. "Jetson Orin modules and developer kit," NVIDIA Corporation. (2022), [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin (visited on Sep. 12, 2022).

[14] P. Besha and A. MacDonald, "Economic development of low earth orbit," National Aeronautics and Space Administration, NASA Headquarters 300 E Street SW, Washington, DC 20546, Tech. Rep. NP-2016-03-2140-HQ, Mar. 2016. [Online]. Available: https://www.nasa.gov/sites/default/files/atoms/files/economic-development-of-low-earth-orbit_tagged_v2.pdf.

[15] R. L. Davidson and C. P. Bridges, "Error resilient GPU accelerated image processing for space applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 9, pp. 1990–2003, 2018. DOI: 10.1109/TPDS.2018.2812853.

[16] E. T. Kain, T. M. Lovelly, and A. D. George, "Evaluating SEU resilience of CNNs with fault injection," in *2020 IEEE High Performance Extreme Computing Conference (HPEC)*, 2020, pp. 1–5. DOI: 10.1109/HPEC43674.2020.9286168.

[17] B. Delporte, R. Rigamonti, and A. Dassatti, "HPA: An opportunistic approach to embedded energy efficiency," in *2016 International Conference on High Performance Computing & Simulation (HPCS)*, 2016, pp. 792–799. DOI: 10.1109/HPCSim.2016.7568415.

[18] F. Busato and N. Bombieri, "A performance, power, and energy efficiency analysis of load balancing techniques for gpus," in *2017 12th IEEE International Symposium on Industrial Embedded Systems (SIES)*, 2017, pp. 1–8. DOI: 10.1109/SIES.2017.7993387.

[19] S. Lal, J. Lucas, M. Andersch, M. Alvarez-Mesa, A. Elhossini, and B. Juurlink, "GPGPU workload character-

istics and performance analysis," in *2014 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV)*, 2014, pp. 115–124. DOI: 10.1109/SAMOS.2014.6893202.

[20] E. Wyrwas, "Body of knowledge for Graphics Processing Units (GPUs)," NASA Goddard Space Flight Center, Greenbelt, Maryland, BOK. NEPP-BOK-2018, 2018. [Online]. Available: https://ntrs.nasa.gov/api/citations/20180006915/downloads/20180006915.pdf.

[21] J. M. Badia *et al.*, "Reliability evaluation of LU decomposition on GPU-accelerated system-on-chip under proton irradiation," *IEEE Transactions on Nuclear Science*, vol. 69, no. 7, pp. 1467–1474, 2022. DOI: 10.1109/TNS.2022.3155820.

[22] C.-Y. Chang, W.-C. Kou, and Y.-C. Chang, "A color 3D scanner based on Jetson TK1 embedded system," in *2022 IEEE International Conference on Consumer Electronics - Taiwan*, 2022, pp. 155–156. DOI: 10.1109/ICCE-Taiwan55306.2022.9869127.

[23] A. Zarandy, T. Zsedrovits, B. Pencz, M. Nameth, and B. Vanek, "A novel algorithm for distant aircraft detection," in *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2015, pp. 774–783. DOI: 10.1109/ICUAS.2015.7152361.

[24] M. N. Rud and A. R. Pantiykchin, "Development of GPU-accelerated localization system for autonomous mobile robot," in *2014 International Conference on Mechanical Engineering, Automation and Control Systems (MEACS)*, 2014, pp. 1–4. DOI: 10.1109/MEACS.2014.6986850.

[25] T. Zsedrovits, A. Zarandy, B. Pencz, A. Hiba, M. Nameth, and B. Vanek, "Distant aircraft detection in sense-and-avoid on kilo-processor architectures," in *2015 European Conference on Circuit Theory and Design (ECCTD)*, 2015, pp. 1–4. DOI: 10.1109/ECCTD.2015.7300065.

[26] K. Earanky, H. Elmiligi, and M. Rahman, "GPU-acceleration of blowfish cryptographic algorithm," in *2015 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, 2015, pp. 507–512. DOI: 10.1109/PACRIM.2015.7334889.

[27] C. A. De S. P. Rodrigues, C. Vinhal, and G. da Cruz, "Fully convolutional networks for segmenting images from an embedded camera," in *2017 IEEE Latin American Conference on Computational Intelligence (LA-CCI)*, 2017, pp. 1–6. DOI: 10.1109/LA-CCI.2017.8285709.

[28] J. Liu, Y. Huang, J. Peng, J. Yao, and L. Wang, "Fast object detection at constrained energy," *IEEE Transactions on Emerging Topics in Computing*, vol. 6, no. 3, pp. 409–416, 2018. DOI: 10.1109/TETC.2016.2577538.

[29] H. Mao, S. Yao, T. Tang, B. Li, J. Yao, and Y. Wang, "Towards real-time object detection on embedded systems," *IEEE Transactions on Emerging Topics in Computing*, vol. 6, no. 3, pp. 417–431, 2018. DOI: 10.1109/TETC.2016.2593643.

[30] S. K. Rethinagiri, O. Palomar, J. A. Moreno, O. Unsal, and A. Cristal, "Heterogeneous platform to accelerate compute intensive applications," in *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*, 2015, pp. 31–31. DOI: 10.1109/FCCM.2015.62.

[31] M. Fatica and E. Phillips, "Synthetic aperture radar imaging on a CUDA-enabled mobile platform," in *2014 IEEE High Performance Extreme Computing Conference (HPEC)*, 2014, pp. 1–5. DOI: 10.1109/HPEC.2014.7040960.

[32] H. Cheng, L. Lin, Z. Zheng, Y. Guan, and Z. Liu, "An autonomous vision-based target tracking system for rotorcraft unmanned aerial vehicles," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 1732–1738. DOI: 10.1109/IROS.2017.8205986.

[33] S. Pundlik, V. Patil, M. Navalgund, A. Sahasrabudhe, and S. Shinde, "Object tracking bot using on-board Jetson TK1: An approach to reduce communication overhead and time delay," in *2017 International Conference on Big Data, IoT and Data Science (BID)*, 2017, pp. 65–72. DOI: 10.1109/BID.2017.8336575.

[34] Innoflight Incorporated, "CFC-500 TFLOP flight computer/payload processor," Tech. Rep., Jan. 2019. [Online]. Available: https://www.innoflight.com/product-overview/cfcs/cfc-500/ (visited on Feb. 1, 2022).

[35] Ibeos, "Ibeos EDGE payload processor," Tech. Rep., Jun. 2021. [Online]. Available: https://www.ibeos.com/edge-datasheet (visited on Aug. 20, 2022).

[36] Arm Ltd., "Cortex-A15 MPCore technical reference manual," Tech. Rep. ARM DDI 0464F, 2012. [Online]. Available: https://documentation-service.arm.com/static/5f042229dbdee951c1cd83d3 (visited on Sep. 31, 2022).

[37] NVIDIA Corporation, "Tegra K1 embedded platform design guide," NVIDIA Corporation, 2701 San Tomas Expressway, Santa Clara, CA 95050, Tech. Rep. DG-07508-001v03, Feb. 2015. (visited on Sep. 27, 2022).

[38] J. Wang, "NVIDIA's next generation CUDA compute architecture: Kepler GK110/210," White Paper. v1.1, Jan. 2014. [Online]. Available: https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-product-literature/NVIDIA-Kepler-GK110-GK210-Architecture-Whitepaper.pdf (visited on Aug. 27, 2022).

[39] NVIDIA Corporation, "CUDA C++ programming guide," NVIDIA Corporation, Tech. Rep. PG-02829-001_v6.5, 2014. [Online]. Available: https://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf (visited on Aug. 27, 2022).

[40] Arm Ltd., "Arm cortex-a series programmer's guide," Tech. Rep. ARM DEN0013D, 2013. [Online]. Available: https://developer.arm.com/documentation/den0013/latest (visited on Oct. 12, 2022).

[41] NVIDIA Corporation, "NVIDIA Tegra K1 series processors with Kepler mobile GPU for embedded applications," Data Sheet. DS-06742-007, Dec. 2019. [Online]. Available: http://developer.nvidia.com/embedded/dlc/tegra-k1-datasheet (visited on Aug. 30, 2022).

[42] Arm Ltd., "ARM architecture reference manual ARMv7-A and ARMv7-R edition," Tech. Rep. ARM DDI 0406C.d, 2018. [Online]. Available: https://developer.arm.com/documentation/ddi0406/latest (visited on Sep. 4, 2022).

[43] Arm Ltd., "Cortex-A7 Floating-Point Unit technical reference manual," Tech. Rep. ARM DDI 0463F, 2013. [Online]. Available: https://developer.arm.com/documentation/ddi0463/f/introduction/product-revisions (visited on Oct. 17, 2022).

[44] Arm Ltd., "NEON programmer's guide," Tech. Rep. ARM DEN0018A, 2013. [Online]. Available: https://developer.arm.com/documentation/den0018/latest (visited on Oct. 11, 2022).

[45] Y. Lin and V. Grover, "Using CUDA warp-level primitives," Jan. 2018. [Online]. Available: https://developer.nvidia.com/blog/using-cuda-warp-level-primitives/ (visited on Sep. 19, 2022).

[46] NVIDIA Corporation, "CUDA math API," NVIDIA Corporation, Reference Manual. v6.5, Aug. 2014.

[47] J. W. Richardson, A. D. George, and H. Lam, "Performance analysis of GPU accelerators with realizable utilization of computational density," in *2012 Symposium on Application Accelerators in High Performance Computing*, 2012, pp. 137–140. DOI: 10.1109/SAAHPC.2012.13.

[48] J. Williams, C. Massie, A. D. George, J. Richardson, K. Gosrani, and H. Lam, "Characterization of fixed and reconfigurable multi-core devices for application acceleration," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 3, no. 4, Sep. 2010, ISSN: 1936-7406. DOI: 10.1145/1862648.1862649.

[49] T. Lovelly, T. Wise, S. Holtzman, and A. George, "Benchmarking analysis of space-grade central processing units and field-programmable gate arrays," *Journal of Aerospace Information Systems*, vol. 15, pp. 1–12, Jun. 2018. DOI: 10.2514/1.I010621.

[50] NVIDIA Corporation. "Unleash your potential with the Jetson TK1 DevKit," NVIDIA Corporation. (2022), [Online]. Available: https://developer.nvidia.com/embedded/jetson-tk1-developer-kit (visited on Sep. 17, 2022).

[51] NVIDIA Corporation. "Linux for tegra r21.8," NVIDIA Corporation. (2019), [Online]. Available: https://developer.nvidia.com/linux-tegra-r218 (visited on Sep. 17, 2022).

[52] NVIDIA Corporation, "CUDA occupancy calculator," NVIDIA Corporation, Tech. Rep. DA-05679-101_v11.7, Aug. 2022. [Online]. Available: https://docs.nvidia.com/cuda/pdf/CUDA-Occupancy-Calculator.pdf (visited on Aug. 29, 2022).