INVESTIGATING, OPTIMIZING, AND EMULATING CANDIDATE ARCHITECTURES FOR ON-BOARD SPACE PROCESSING

by

Benjamin Joseph Schwaller

B.S. Electrical Engineering, University of Florida, 2016

Submitted to the Graduate Faculty of

the Department of Electrical and Computer Engineering

Swanson School of Engineering in partial fulfillment

of the requirements for the degree of

Master of Science in Electrical Engineering

University of Pittsburgh

2018

UNIVERSITY OF PITTSBURGH

SWANSON SCHOOL OF ENGINEERING

This thesis was presented

by

Benjamin Schwaller

It was defended on

April 6, 2018

and approved by

Alexander Jones, Ph.D., Professor
Department of Electrical and Computer Engineering

Jun Yang, Ph.D., Professor
Department of Electrical and Computer Engineering

Thesis Advisor: Alan D. George, Ph.D., Professor
Department of Electrical and Computer Engineering

INVESTIGATING, OPTIMIZING, AND EMULATING CANDIDATE ARCHITECTURES FOR ON-BOARD SPACE PROCESSING

Benjamin Schwaller, M.S.

University of Pittsburgh, 2018

With increasing computational demands in the defense and commercial industries, future space missions will require new, high-performance architectures. Extensive research, benchmarking, and analysis of candidate architectures is required before performing the expensive, time-consuming process of radiation-hardening on suitable devices. In this work, we first compare two such candidate architectures: the Texas Instruments KeyStone II octa-core DSP and the ARM Cortex-A53 quad-core CPU. We evaluate the performance of a key kernel used in space applications, the Fast Fourier Transform (FFT), and a key space application, the complex ambiguity function (CAF), on each architecture. We also develop and evaluate a direct-memory access scheme to take advantage of the KeyStone II architecture to perform FFTs. The KeyStone II's batched 1D-FFT performance-per-watt is 4.1 times greater than the ARM Cortex-A53 and the CAF performance-per-watt is 1.8 times greater. Next, we develop and employ an emulator to study the performance of the High-Performance Spaceflight Computing (HPSC) processor. The HPSC processor is a future architecture under development by Boeing and funded by NASA and AFRL for their future space missions. HPSC is comprised of "chiplets" which have two quad-core ARM Cortex-A53 CPUs connected by an AMBA bus. These chiplets can be connected by different serial interfaces depending on mission needs. By employing two ARM platforms, an octa-core ARM architecture and two quad-core ARM architectures connected by Ethernet, we project HPSC performance for FFTs and another key space application: synthetic-aperture radar (SAR). We project that SAR will scale well on a multi-chiplet platform with a performance gain of 2.94 over

a single US+ board when using two connected chiplets. Our research provides new insights on the tradeoffs encountered when parallelizing functions on these candidate architectures, including novel optimization techniques for each architectures.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF EQUATIONS

PREFACE

This work was supported by SHREC industry and agency members and by the IUCRC Program of the National Science Foundation under Grant No. CNS-1738783. I wish to acknowledge Dr. Asheesh Bhardwaj from Texas Instruments, Dr. Josh Bruckmeyer from Harris Corporation, and Dr. Andrew Pineda from Air Force Research Lab, for their contributions in conducting the research presented in this paper. Most importantly, I would like to thank my research advisor, Dr. George, for seeing my potential in 2015, bringing me into the lab and encouraging my work throughout the years.

1.0 INTRODUCTION

Space is truly the final frontier in the realm of computer engineering. While ground-based systems aim to break the exascale barrier (i.e. 10^{18} operations-per-second) in the coming decade, processors in space are just stepping into terascale (i.e. 10^{12} operations-per-second) [1]. Nevertheless, high-performance on-board processing is a major requirement for space missions. Sensor technology is developing rapidly, leading to sensors that generate more data than ever before [2]. The IKONOS satellite launched in 1998, had a .82 m panchromatic resolution, and generated 240,000 square km of imagery per day. By comparison, the WorldView-4 satellite, launched in 2016, has a .31 m panchromatic resolution and can generate 680,000 square km of imagery per day [3]. Additionally, scientists continue to use this data for a wide variety of applications, ranging from weather tracking [4] to national defense [5]. Satellite engineers have two options for processing this critical data: send it to Earth for computation at a ground station or have perform the complex signal-processing onboard the satellite. Due to the bandwidth limitations between Earth and space systems, which are on the order of hundreds of kbps downlink data rates for CubeSats [6], many developers are forced to choose the latter. Therefore, space computers must be capable of processing sensor data that was previously processed on ground-based systems. Creating a radiation-hardened (radhard) processor that can not only perform reliably in the harsh environment of space but also is also capable of performing these compute-intensive applications is a lengthy and costly process. Such a process requires extensive research

prior to choosing an existing architecture to harden. The research in this thesis provides insight into several candidate architectures for space and their viability at performing these intensive computations. The thesis is broken into two phases in accordance with the timeline for the research. The goal of the first phase is to compare two candidate architectures and their performance and performance-per-watt for a kernel and application used in space missions. The second phase emulates a radhard processor currently in development and projects its performance for a space kernel and application.

1.1 PHASE ONE: KS2 AND ARM COMPARISONS

In the first phase of our research, we examine and compare two candidate architectures for space processing: the Texas Instruments KeyStone II (KS2) and a quad-core ARM Cortex-A53. The KS2 architecture integrates reduced instruction set computers (RISC), specifically ARM cores, and digital signal-processing (DSP) cores. We chose to investigate the KS2 because of its multi-core DSP architecture. The European Space Agency has used DSPs on their missions for many years. In 2009, they established the Next-Generation DSP (NGDSP) project in search of an architecture to meet their projected needs and replace their current radhard DSP [7]. DSPs are designed to compute signal-processing applications, such as the functions considered in this paper, utilizing their fast multiply-accumulate units and specialized instruction sets [8]. DSPs are particularly interesting because they can achieve efficient solutions to many complex space applications, which usually involve some form of signal-processing.

In a similar effort, NASA and AFRL launched the High-Performance Spaceflight Computing (HPSC) initiative to investigate a new multi-core processor for their future space

missions which became the motivation for exploring a quad-core ARM CPU [9]. In March 2017, Boeing was awarded the contract for this initiative after proposing their “chiplet” approach.

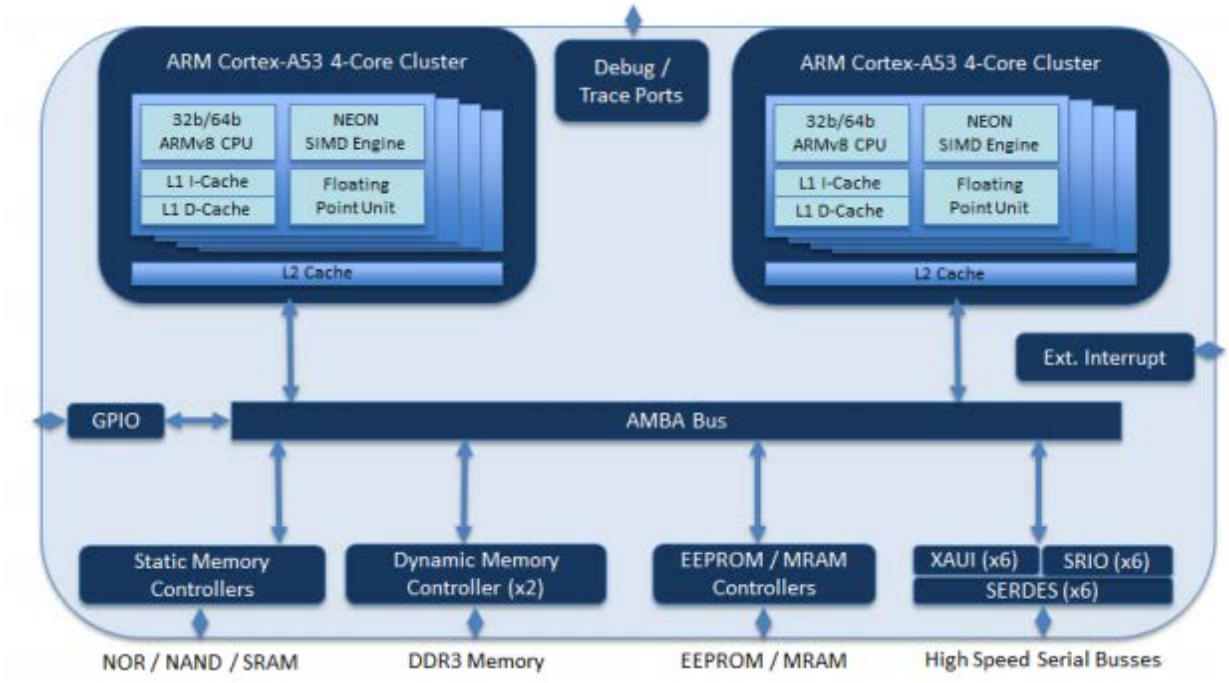


Figure 1. HPSC chiplet block diagram [9]

The block-diagram for their proposed chiplet is shown in Figure 1. The main chiplet is composed of two ARMv8 Cortex-A53 quad-core clusters connected by an Advanced Microcontroller Bus Architecture (AMBA). Each cluster on the chiplet features on-chip memory, a Single-Instruction, Multiple-Data (SIMD) engine called NEON, and a Floating-Point Unit (FPU). The chiplet provides three serial protocols that are used to connect to peripheral chiplets, such as an FPGA accelerator or another compute chiplet. The first protocol, Serial RapidIO (SRIO), is built upon the RapidIO protocol which is considered the leading-edge fault tolerant standard interconnect [11]. The second interface is XAUI (10-Gigabit-Attachment-Unit-Interface)

which is a standard for extending the XGMII (10-Gigabit-Media-Independent-Interface) between the MAC and PHY layer of 10-Gigabit-Ethernet to be able to provide chip-to-chip connectivity [12]. The final interface is SerDes (Serializer / Deserializer) which is a pair of functional blocks that convert between serial and parallel data [13]. HPSC is still under development and will be for several years, but it will be an important processor model moving forward. Studying HPSC with hardware emulation is, therefore, important for future developers and even for the HPSC architects themselves.

Our methodology for examining these two architectures focuses on studying a key space kernel and a key space application on each device. The kernel we chose is the Fast Fourier Transform (FFT), a widely-used function that transforms a signal from the time domain to the frequency domain. Our work focuses specifically on batched 1D and 2D-FFTs. A batched 1D-FFT performs the FFT function on each row in a 2D array of data, whereas the 2D-FFT performs the same function along both the rows and columns of that array. Accessing the columns of an array is often inefficient, as will be discussed further in Sections 3.0 and 5.2.1. FFTs are known to be memory-bound functions which stress the memory bandwidth capabilities of devices, a factor which we explore in our device comparisons. To conduct these comparisons, we leverage the FFTLIB library from TI for the KS2 and the open-source FFTW library for the ARM Cortex-A53.

The space application chosen is the complex ambiguity function (CAF). CAF is a radar signal-processing algorithm that can be used for geolocation. Many global positioning systems calculate the time difference of arrival (TDOA) and frequency difference of arrival (FDOA) of the same signal received at two different locations to pinpoint the origin of the signal. CAF can be employed to jointly calculate these values. CAF is not an embarrassingly parallel application and

contains several complex stages of computation. As a result, it provides a more robust assessment of the candidate architectures' capabilities.

The contributions from Phase-1 of this research are twofold. The first contribution is the newly created direct memory-access (DMA) transfer scheme that is tailored to the KS2 architecture. The scheme takes advantage of the L2 and L1 on-chip memories in the DSP cores, which can be partially configured as scratchpad memory. By pre-fetching data into these locations, we can accelerate execution time for batched processes. The second contribution is the benchmarking comparisons of FFTs and CAF on the KS2 architecture and on a quad-core ARM A53 architecture. We examined both the performance and performance-per-watt for each system.

1.2 PHASE TWO: HPSC EMULATION AND PROJECTION

Phase-2 of this research studies the viability of using multiple ARM platforms to emulate the HPSC processor. As described earlier, the chiplets for the HPSC have dual quad-core ARM Cortex-A53 processors connected by an AMBA bus. We chose to use the HiKey LeMaker Board to emulate the HPSC processor because it has a similar architecture. Another important part of HPSC emulation is the multi-chiplet performance. This emulation is achieved by connecting two Xilinx Zynq UltraScale + (US+) boards together via Ethernet. The 1 Gb/s Ethernet (GbE) interface on the board was used as a substitute to connect the boards since none of the proposed HPSC serial connections existed natively on the board. Since GbE is not the exact HPSC serial interface, we extrapolated performance for 10Gb/s Ethernet (10GbE) and 6-lane 10Gb/s Ethernet (6x10GbE) using a custom model. The benchmarking used for projecting HPSC performance is described in Section 5.2.3.

Similar to Phase-1, we study a key space kernel and space application on the HiKey and the US+ platforms. Batched 1D-FFTs and 2D-FFTs are benchmarked on both platforms using the FFTW library just as with the ARM architecture in Phase-1. However, instead of employing the CAF application to evaluate the platforms, we chose the synthetic-aperture radar (SAR) algorithm for the following reasons. NASA has identified SAR as a highly relevant application for testing space-bound processors. SAR requires several complex phases of computation and is a part of the HPSC benchmark suite [14]. We also chose SAR because previous students of the NSF SHREC Center, along with Dr. Pineda and other members of the Air Force Research Lab, have developed an optimized SAR application which was readily available for testing purposes.

SAR is a radar system used on both aircraft and spacecraft that uses a craft's trajectory to act as an extremely large antenna. By collecting radar reflections from different positions, the antenna array is artificially lengthened. This approach allows for a narrow array beam pattern and a finer azimuth resolution. As such, SAR systems deliver highly-detailed images even through weather effects and at night [15]. A detailed explanation of the SAR algorithm is beyond the scope of this thesis and is explained thoroughly by the AFRL researchers in [16,17].

Phase-2 provides three key contributions into the candidate architectures for future developers of these platforms and the HPSC processor. The first contribution details the performance tradeoffs of using the AMBA bus and serial connections for parallelizing applications, which will help developers determine a suitable granularity for parallelized tasks on these processors. The second contribution projects both single-chiplet performance and multi-chiplet performance for the functions studied, which inform the computing community about the performance that can be expected from these platforms. The final contribution is an optimization for the 2D-FFT FFTW function, which can be used by any developer using this library.

2.0 SPACE-COMPUTING CHALLENGES

This section discusses the constraints in space for high-performance computing to illustrate the difficulty in making processors for space applications and the need for choosing a suitable platform before investing time, money, and development resources. Size, weight, power, and cost, or SWaP-C, are all important considerations for designing space-grade processors. Weight and size considerations are derived from weight and mechanical housing restrictions on the spacecraft. For example, NASA's CubeSat Launch Initiative (CSLI) provides educational institutions and non-profit organizations access to space for small satellites. These CubeSats must conform to standard dimensions defined as U, which is 10cm x 10cm x 10cm. Each CubeSat can be 1U, 2U, 3U, or 6U in size and usually weigh less than 3lbs per U [18]. Power considerations arise because CubeSats can only supply processors with a few watts [19]. By comparison, the terrestrial supercomputers today require megawatts to achieve their high performance [20]. Of course, cost considerations are present in any system design. However, radhard components are more expensive than their commercial off-the-shelf (COTS) counterparts. Additionally, buying space on a rocket for deployment is expensive. For example, placing a payload on one of the most economical spacecraft available today, the SpaceX Falcon Heavy, costs $640 per pound for Low Earth Orbit (LEO) launches [21].

Space processors also need to tolerate the effects of radiation that ground stations are shielded from. Particles, such as protons, neutrons and heavy ions, can create temporary and permanent faults in an electronic system, as exemplified in Figure 2. These faults are referred to as Single-Event Effects (SEEs). There are several different types of SEEs. The first is Single-Event Upsets (SEUs), which are caused by particle hitting a transistor and is observed as a bit flip inside

a register or memory element. SEUs may or may not manifest as an error depending on which bit is flipped. The affected transistor could hold unused data, or a value could be written to the element after the SEU occurs and before the faulty value is read, effectively masking the fault. However, the bit flip could be on a critical part of the system, such as the program counter, which could cause a catastrophic system failure. Single-Event Transients (SETs) are transient voltage pulses that can later be captured by a storage element and propagate throughout the rest of the component. Single-Event Latch-ups (SELs) occur when a highly-charged particle hits the semiconductor and causes the current of the device to be driven out of spec. SELs can result in shorts occurring between transistors and power rails inside the ICs. They can disrupt program execution or cause transistors to fail permanently. There are several other types of SEEs including Single-Event Snapbacks (SESs), Single-Event Burnouts (SEBs), Single-Event Functional Interrupts (SEFIs), and Single-Event Gate Ruptures (SEGR) [22,23]. All of the above SEEs can normally be remedied by rebooting the device, apart from some SELs, SEBs, and SEGRs. However, rebooting is generally not an option for critical space missions. Instead, radhard processors reduce the impact these SEEs can have on their architecture and repair themselves when a SEE occurs. More advanced techniques, discussed later in this section, have been developed to reduce SEEs and to combat their effects.

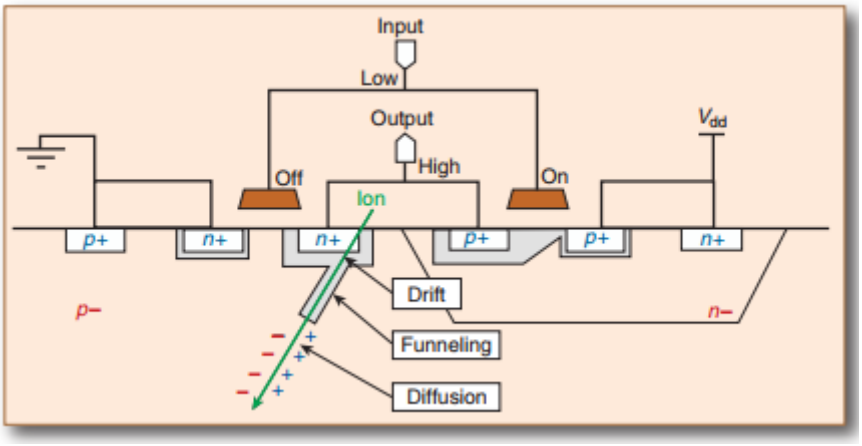


Figure 2. Heavy ion strike on the cross-section of a bulk CMOS memory cell [21]

In addition to these short-term effects, long-term exposure to radiation can cause faults to accumulate in a device. The amount of exposure a device can withstand before it ceases to operate within its specification or stops functioning all together is called the Total Ionizing Dosage (TID). TID is measured in rads or radiation absorbed dose. While SEEs are typically caused by high energy particles, TID effects are caused by low energy protons and electrons being captured within the silicon [23].

The number of SEEs and severity of TID a processor must cope with depends on the orbit a spacecraft inhabits. In LEO for instance, the spacecraft is still within the earth's magnetosphere and is partially shielded from radiation. The TID in LEO is about 100 krad/year without shielding [24]. Using less reliable, non-space-grade parts in these orbits is a prudent decision to save on cost. One such example is the Hewlett Packard Enterprise supercomputer, launched in August 2017, which is currently inside the well-shielded International Space Station in LEO [1]. This high-performance computer is composed entirely of commercial parts and, at the Supercomputing Conference in November 2017, HPE reported that the device has not experienced any faults. Orbits farther from Earth are subject to higher amounts of radiation and require additional redundancy or

more reliable parts. For example, a processor in Geostationary Transfer Orbit (GTO) experiences 10 Mrad/year without shielding, three orders of magnitude higher than LEO [24]. The Van Allen Belts, layers of charged particles held by Earth's magnetic field, can also cause fluctuations in radiation levels. The South Atlantic Anomaly (SAA) occurs over South America and the southern Atlantic Ocean where the inner Van Allen Belt comes closest to the Earth's surface. The SAA causes spacecraft to observe increased levels of radiation, even in LEO [25]. The HARFT framework [26] on the CHREC Space Processor (CSP) demonstrates one solution to this phenomenon by reconfiguring the system to provide additional redundancy, at the cost of higher performance, when higher levels of radiation are detected. Space processors need to be able to withstand their worst-case scenario radiation levels or be able to reconfigure themselves, similar to the CSP.

To combat these challenges, radhard processors undergo an extensive development process. Radhard device often start their development lifecycle as COTS components. Designers then integrate electronic components that are built specifically to resist radiation. The number of electronic components that need to be added to or replaced on the COTS device depend on the mission needs and expected radiation levels. These components can take advantage of different materials, like silicon on sapphire [27], have more complex circuit-level designs, such as using TMR on flip-flops [28], use SRAM over DRAM, incorporate shielding to reduce radiation exposure, and integrate many other techniques to mitigate the effects of space radiation [29]. Designers can also add redundant parts to a chip to act as fail-safes which requires a non-negligible amount of room on the die. Adding this redundancy can lead to the removal of parts or features, ranging from floating-point execution units to entire cores, to meet SWaP-C constraints. In addition, the radiation-hardening process usually requires space-grade processors to use a reduced

clock speed leading to lower performance. These factors contribute to an extensive design process lasting several years which has created the technological gap between space and commercial processors and illustrates the need for a thorough vetting process when selecting a new processor for radiation-hardening.

3.0 RELATED RESEARCH

Previous research has studied at many aspects of the candidate architectures presented in this research and the functions used to test them. Researchers have also benchmarked the CAF algorithm on other radhard space processors such as the Boeing Maestro many-core CPU [30] and the BAE RADSPEED™ DSP [31]. Direct comparisons between the results of these studies and the results of this research are difficult to make due to many factors. For example, literature on the Maestro only reveals iteration time for CAF rather than operations per second as considered in Phase-1 of this research. Marshall et al. [31] did not include preprocessing techniques required to turn the signal into analytic format presented in this paper in software and instead performed this task on an analog signal generator. Additionally, input parameters for these papers were not well defined and could have been much larger than the inputs used in this research. Singh et al. [30] did present FFTW performance for 1K point FFTs which had comparable performance to the architectures described in this research reaching roughly 2.6 giga floating-point operations per second (GFLOPS) on double-precision 1K FFTs compared to our results of 2 GFLOPS on the ARM architectures and 18 GFLOPS on the KS2. However, the Maestro results were obtained using a simulator rather than physical hardware. It is important to note that radhard processors by their very nature have lower clock rates and other hardware limitations to overcome the harsh environment of space, so comparing the KS2 and ARM CPUs to the Maestro and RADSPEED processors requires data about the latter after hardening.

Previous research has also been conducted on the optimization of other functions on the KS2 architecture using the L1 and L2 on-chip memory. Gao et al. detail the complex tradeoffs in using part of the SRAM as scratchpad [32]. Scratchpad memory is a high-speed internal memory

for the temporary storage of data. However, they did not attempt to perform computations and transfers asynchronously, as this research does. Bahtat et al. applied radar-pulse compression on this architecture using part of the L2 cache as scratchpad [33]. However, they left the L1 space configured entirely as cache, hence their transfer scheme differs from the one described in Section 5.1.1.

As previously noted, we chose the FFTW library for performing the FFTs on the ARM devices. FFTW, an open-source library from MIT, has been shown to have remarkable performance and tunes well for many different architectures. Frigo and Johnson [34] describe the first version of the library and its performance compared to other available FFT software. They concluded that FFTW had better performance than all other open-source FFT libraries and was comparable to machine-tuned libraries at the time. Many improvements have been made since its inception in 1998. The current version is labelled FFTW3 and continues to provide portable, high-performance C codebase for FFTs. Frigo and Johnson compares the library to others and details the new changes [35]. Notably, FFTW3 has a plan feature which explores the algorithm space of FFTs and chooses the one that executes fastest on the architecture at runtime. The plan function can tweak how long the plan will consider different steps. FFTW3 is also one of the few libraries that takes advantage of SIMD instructions, which allow it to use the ARM NEON engine.

Other research has optimized FFTs for processors in a similar fashion to that of this paper. Mermer et al. describe the process for efficiently implementing 2D-FFTs on Hitachi/Equator's MAP mediaprocessor. They also used a DMA scheme to double buffer data from off chip memory to on-chip cache [36]. They acknowledge that accessing along the columns for the 2D-FFT is inefficient and report that transposing twice, as described later in our research, is a valid method to obtain an increase in performance. This method does necessitate extra cycles purely for data

management. Therefore, they are able to process multiple image columns simultaneously to avoid the memory-access inefficiency. This optimization led to a runtime improvement of 1.3x over the double transpose method and a 1.85x improvement over the naïve method [36].

4.0 ARCHITECTURE OVERVIEW

As previously stated, our research studies the feasibility of multiple architectures for space applications. This section details the features and specifications of four different development boards which contain these architectures.

4.1 K2EVM-HK

The TI KeyStone II architecture examined in this paper is part of the K2EVM-HK board, shown in Figure 3. This system has four ARM Cortex-A15 MPCore processors with eight TMS320C66x DSPs. The DSPs are housed within the C66x CorePacs which also include 32 KB of L1 and 1 MB of L2 on-chip memory. These memories can be configured to be entirely cache or part cache, part scratchpad. The external direct-memory access (EDMA) and internal direct-memory access (IDMA) peripherals enable the device to perform DMA-transfers between DDR3 memory and the L1 and L2 memory space on the C66x cores. These DMA-transfers do not involve the CPU, so computation can run in parallel with DMA-transfers without consuming extra cycles. This architectural feature gives developers flexibility in tuning the application to their architecture and is the main feature exploited in this research [37].

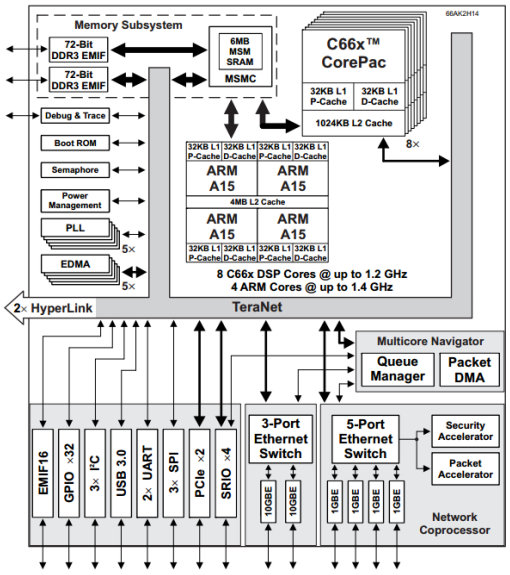


Figure 3. K2EVM-HK block diagram [37]

4.2 ODROID-C2

The next architecture examined in this paper is a quad-core ARM Cortex-A53. Several boards featured this architecture, the first being the ODROID-C2. The ODROID-C2 is a 64-bit quad-core single board computer (SBC) featuring an Amlogic S905 quad-core ARM Cortex-A53 (ARMv8) processor running at 1.5GHz whose functional block diagram is shown in Figure 4**.** The ODROID-C2 is less expensive than the K2EVM-HK and consumes less power [38].

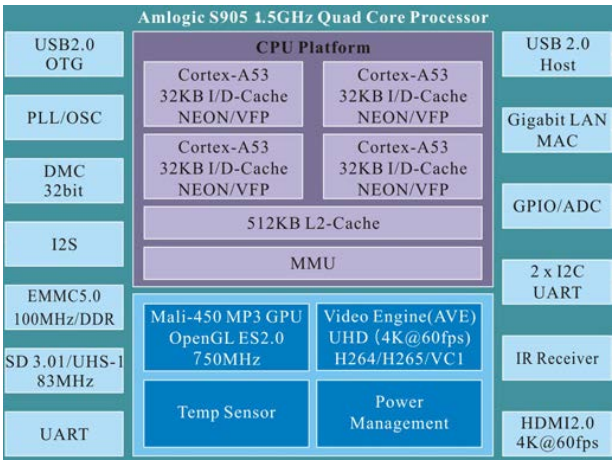


Figure 4. ODROID-C2 block diagram [38]

4.3 HIKEY LEMAKER 2GB

The next device used in this study is the HiKey LeMaker 2GB whose block diagram is shown in Figure 5. The HiKey is another relatively low-cost ARMv8 architecture development board, similar to the ODROID-C2, running at a maximum of 1.2GHz. The HiKey is an octa-core CPU as opposed to the quad-core ODROID-C2. The octa-core is constructed from two quad-core clusters connected by a CoreLink CCI-400 interconnect, which is built on the AMBA AXI4 specification. Therefore, architecture closely resembles the current layout of a single chiplet within the HPSC, which contains two quad-cores connected by an AMBA bus [39].

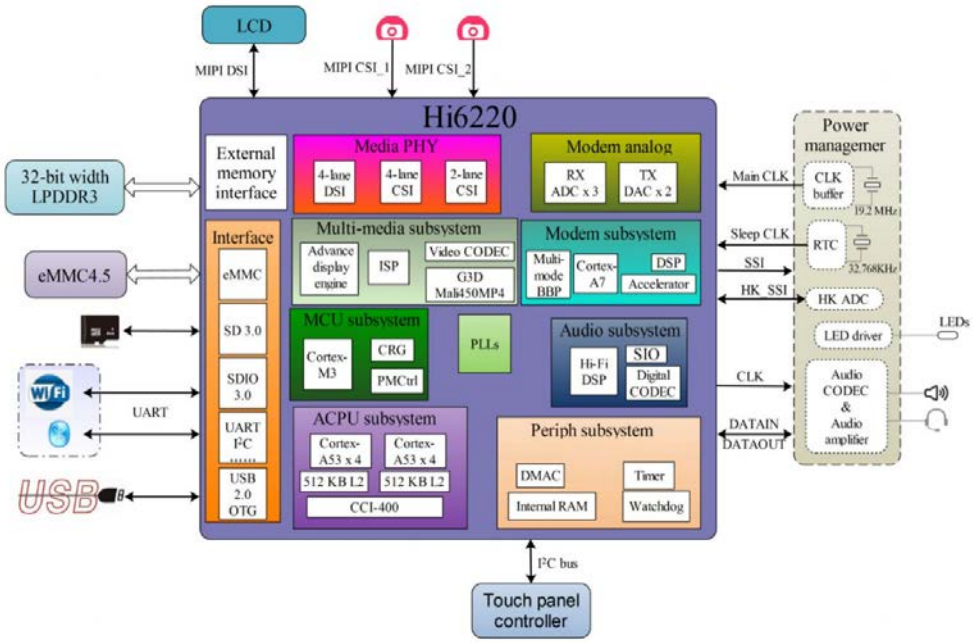


Figure 5. HiKey LeMaker 2GB block diagram [39]

4.4 ZYNQ ULTRASCALE +

The final device used in this research is the Zynq Ultrascale + (US+) featured on the ZCU102 evaluation board whose block diagram is shown in Figure 6. This development board includes both a quad-core ARM Cortex-A53 CPU, which runs at a maximum frequency of 1.2 GHz, and a FPGA fabric. We have connected two of these boards via GbE to emulate the serial connection that will connect chiplets in HPSC. We created a model, described in Section 5.2, to extrapolate the multi-chiplet performance from this platform [40].

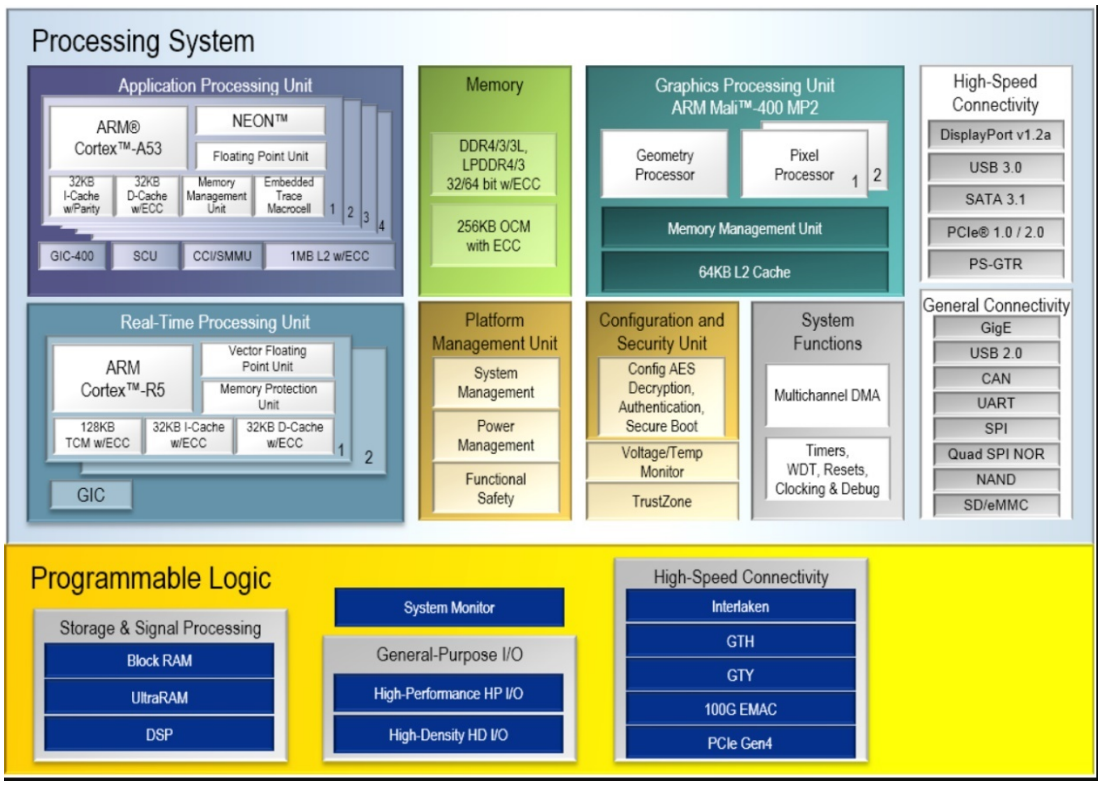


Figure 6. Zynq UltraScale+ block diagram [41]

5.0 METHODOLOGY

Multiple novel techniques were developed in this research to optimize and investigate the candidate architectures. This section describes these techniques, how the functions were benchmarked, and the data collection ideology for both phases of our research.

5.1 PHASE-1 METHODOLOGY

This section describes the method in which FFTs and the CAF algorithm are executed on the KS2 and the quad-core ARM A53 architectures. The DMA scheme developed for batched FFTs on the KS2 is also described in this section. Additionally, the method of evaluating CAF on both architectures is outlined.

5.1.1 FFT Methodology on K2EVM-HK

TI provides several libraries that perform FFT functions. We studied the DSPLIB and FFTLIB libraries to give a baseline performance for the KS2. The FFT functions under investigation are complex-to-complex, single-precision batched 1D and 2D-FFTs. All functions are verified by using MATLAB generated inputs and outputs. The FFT functions are called from a single ARM core and sent to a single DSP core using OpenCL (Open Computing Language). OpenCL “provides a common language, programming interfaces, and hardware abstractions enabling developers to accelerate applications with task-parallel or data-parallel computations in a

heterogeneous computing environment" [42]. The input and output buffers and function to be executed are declared using OpenCL function calls and then dispatched to a DSP core. Next, OpenMP is invoked on a single DSP core to parallelize computation across the other DSP cores. OpenMP is an API that "facilitate[s] shared-memory parallel programming" [43]. Rather than parallelizing an individual FFT function, different portions of the batch of FFTs are sent to different cores for execution, leveraging the inherent task-parallelism of the batched FFT.

Through benchmarking we found that FFTLIB performed better than DSPLIB, shown in Section 6.0, and thus was chosen as a foundation for this work. The performance difference stems from the FFTLIB taking advantage of the L2-SRAM space for DMA-transfers whereas DSPLIB does not. A custom DMA-transfer scheme, shown in Figure 7, was developed to accelerate batched FFT computations using FFTLIB. This scheme is called "ping-pong" and is built upon FFTLIB's DMA scheme. While the FFTLIB batched functions can take advantage of the L2-SRAM space, using L1-SRAM in addition is a novel approach.

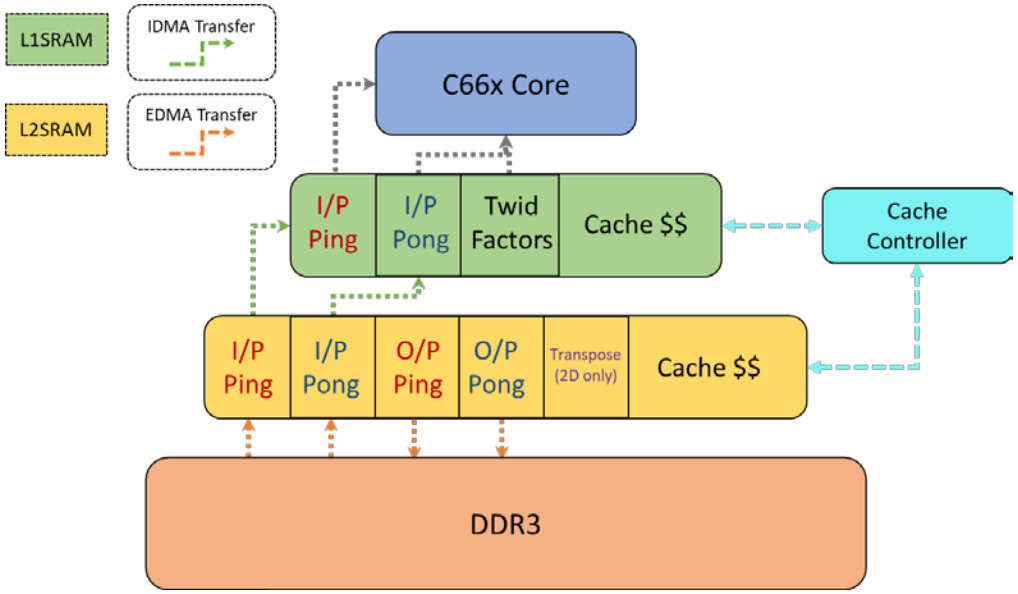
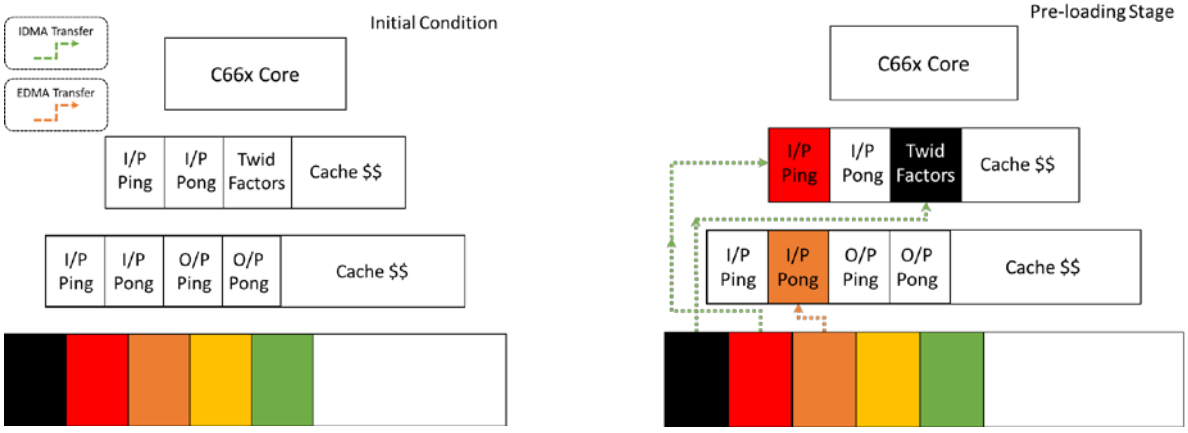


Figure 7. Ping-pong DMA scheme using L1-SRAM

The motivation behind this new scheme is to accelerate computation by manually pre-fetching data into the L1-SRAM scratchpad to reduce future memory-access latency. The L1 memory is the closest memory space to the C66x core. Placing data here enables the core to access the input data faster than data stored in DDR3 memory and reduces overall execution time by limiting the amount of cache misses. Four buffers are set in the L2-SRAM space: two for input and two for output. Two buffers are also set in the L1-SRAM space, both input buffers, along with additional room for the twiddle factors in the L1-SRAM space. Twiddle factors are trigonometric constant coefficients that are used throughout the course of the FFT algorithm, so it is important to keep them in L1-SRAM. Since data can be transferred back to the L2-SRAM scratchpad after computations, output buffers for the L1-SRAM scratchpad are not required. The maximum FFT size studied is 1K due to the size limitation of L1-SRAM. To fit two input buffers and the twiddle factors, a 1K FFT using single-precision floating-point data requires 24KB of space. The L1 space is only 32KB, so larger FFTs are not be feasible with this scheme.



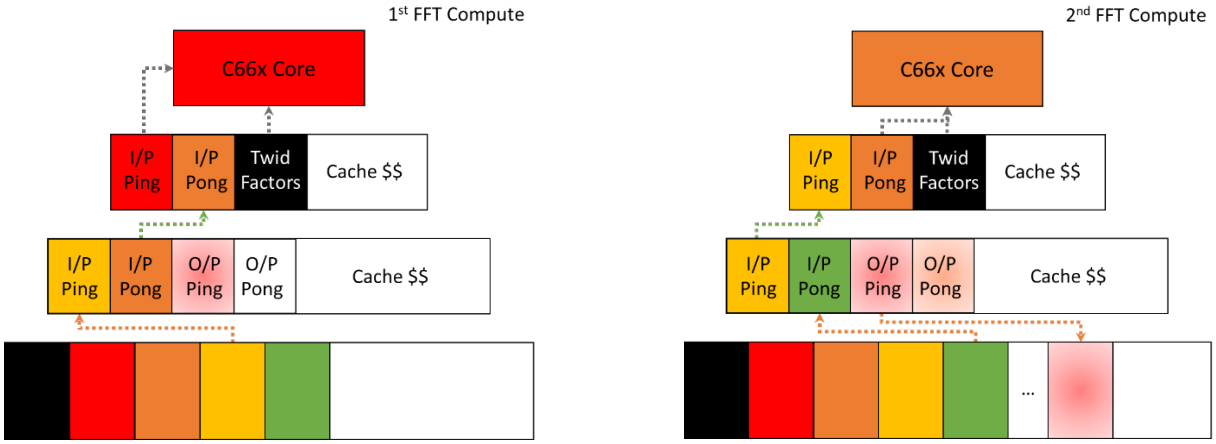


Figure 8. Ping-pong scheme in action for batched 1D-FFT

Next, we describe the process for performing batched 1D-FFTs as illustrated in Figure 8. Solid colored squares are input FFT blocks while gradient colored squares are completed FFT blocks. The first FFT input and twiddle factors are loaded into the input ping buffer in L1 and twiddle buffer, respectively, from DDR3 memory, while the second FFT input is loaded into the L2 input pong buffer. This preloading stage only occurs at the beginning of the function. Then, the first FFT is computed from L1 ping buffer, while at the same time the second input set is loaded into the L1 pong buffer from the L2 pong buffer. The third FFT set is also loaded into L2 ping at this time. The FFT result is stored into the L2 output ping buffer which is transferred back to DDR3 memory on the next cycle. This process repeats until all FFTs have been computed as the DMA managers switch between loading and storing to ping and pong buffers, hence the name “ping-pong”. 2D-FFTs are performed in a similar same way, however, instead of only using four buffers in L2-SRAM, scratchpad there are five, the fifth being the transpose buffer. All data from computation is sent to the transpose buffer instead of the output buffers. Finally, an optimized transpose using an FFTLIB function is performed from the transpose buffer to one of the output

buffers to complete a batched 1D-FFT with transpose. This process is performed again to compute the 2D-FFT.

Since the L2 space is much larger than the L1 space, the program could load, for example, 10 sets of FFTs into the L2-SRAM followed by 10 L1 DMA-transfers and then FFT computations would occur before transferring the data back from L2 to DDR3 memory. Ideally, the time required to transfer data to the L1-SRAM and compute all FFTs loaded into the L1 space would be the same as the time to transfer everything into the L2 space since both operations are occurring simultaneously. Since L2 transfer time usually takes longer than L1 transfer time, using different transfer sizes is typically an advantageous strategy. In contrast, the base FFTLIB simply transfers the maximum possible amount of data into the L2 and L1-SRAM scratchpads. Balancing the transfer time and computation time of the data is critical to achieving maximum performance. For the results, these transfer sizes are tuned for each problem-size and number of cores used to obtain maximum performance.

5.1.2 FFT Methodology on ODROID-C2

The FFTW3 library is used to calculate the FFTs on the ODROID-C2 platform. The rationale for using this library was presented in Section 3.0. FFTW3 automatically detects the number of cores available for parallelization through OpenMP. Overall, FFT programming for the ODROID-C2 is a much faster process because the optimizations are performed by the FFTW3 library. The only functionality we had to develop for the ODROID-C2 was batching the FFTW3 library's 1D-FFT function.

5.1.3 Complex Ambiguity Function Methodology

The mathematical definition of CAF is shown in Equation 1, where S1 and S2 are continuous-time signals in analytic signal format, T is the integration period, τ is the time delay between the two signals, and f is the frequency offset between the signals. The * symbol denotes the complex conjugate of the variable. In discrete time, the function becomes similar to the definition of a Discrete Fourier Transform (DFT) which is efficiently computed using an FFT. Therefore, Equation 1 can be rewritten as Equation 2.

$$CAF(\tau, \mathrm{f}) = \int_0^T S1(t)S2*(t+\tau)e^{-j2\pi ft} \quad \textbf{5-1}$$

$$CAF(\tau, \mathrm{k}) = \mathrm{FFT}[\mathrm{S1(n)S2}*(\mathrm{n}+\tau)] \quad \textbf{5-2}$$

The magnitude of the CAF surface produced will peak when τ and k/N (the fractional frequency difference) are equal to the TDOA and FDOA, the target values of the function. As previously noted, CAF requires that the S1 and S2 signals are in analytic signal format. Typically, signals include both positive and negative frequencies. However, an analytic signal has only positive frequency components. The Hilbert Transform can be used to transform a real-valued signal into its analytic representation.

CAF is computed in a similar fashion on both the K2EVM-HK and ODROID-C2 boards. Figure 9 shows the flow diagram for CAF. The input data is based on a boat emitting a pure sine wave as it moves away from a receiver in a ship port tower. *X1* and *X2* are the input signals to the

algorithm. *X1* is a pure sine wave at 9GHz and X2 is the same sine with a frequency offset of 4kHz and a time offset of 100us. Both input signals have 16K samples.

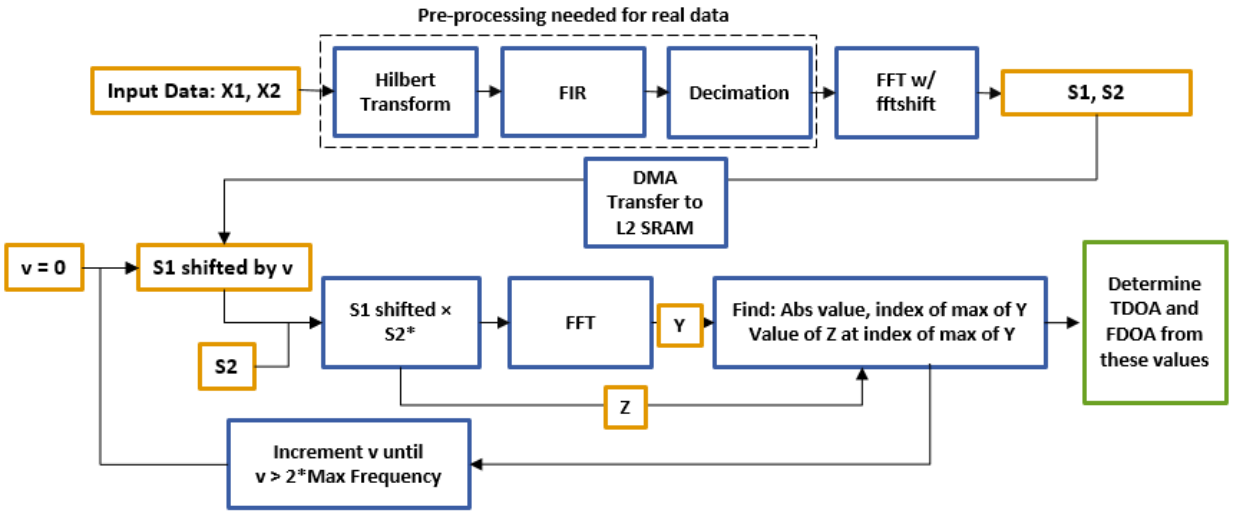


Figure 9. Complex ambiguity function flow diagram

This input was chosen in conjunction with our partners at Harris Corporation. The format of CAF is based off MATLAB code from [44]. The fine computation described [44] is not used in our function because the computational requirements needed for a nearly-negligible improvement in output precision were deemed unnecessary. The input data does not contain complex numbers and is transformed into analytic format for correct computation in the pre-processing stage in Figure 9. This process is composed of a Hilbert Transform, a lowpass finite response filter (FIR), and decimation by two. This process also mimics the quadrature demodulation that would occur in a real system. The Hilbert Transform and FIR are optimized using OpenMP and the decimation is optimized with SIMD instructions native to the ODROID-C2 architecture. The application ran on the KS2 did not include this SIMD optimization. Next, the FFTs are performed along with an optimized version of MATLAB's fftshift function to create the *S1* and *S2* signals for the next block

as denoted in Figure 8. For the K2EVM-HK, these signals are transferred using DMA to the L2 space for the remaining computation. However, the DMA transfer does not occur on the ODROID-C2 because it lacks the required functionality.

In the next stage, *S1* is shifted by different amounts ranging from the maximum negative frequency to the maximum positive frequency, which is application-specific. The shifted *S1* is then multiplied by the complex conjugate of *S2*. The FFT of the product of these signals is performed and the maximum value of the FFT, index of this value, and value at that index of the *S1* / *S2* product, determine the TDOA and FDOA. The batches of FFTs are parallelized with OpenMP by having different cores work on different shifted *S1* signals. The final results are compared to MATLAB results for verification.

The optimized batched FFT method described in Section 5.1.1 is not optimal for CAF on K2EVM-HK. Attempts were made to ping-pong the data to L1 and back, but this resulted in slower performance. The performance loss occurs because the same data set is computed in the for loop for CAF rather than loading new data every iteration, as in our FFT functions. In the beginning of CAF, when the processor tries to retrieve the first input array, there will be a cache miss. For all subsequent operations, there will be cache hits since that data remains in cache. The advantage of the ping-pong scheme for FFTs is the avoidance of cache misses, but the data movement in CAF does not lead to cache misses due to its inherent spatial locality. However, the transfer of *S1* and *S2* to L2-SRAM did result in a speedup of 2.3 times. Additionally, the pre-processing did not occur in the L2 space because the gain in computation speed is too small to offset the additional transfer time for these functions due to the small problem-size studied in this paper.

5.1.4 Performance Measurements

Timing starts on the KS2 after the function has been dispatched to the DSP cores using OpenCL. Any initializations, including allocating memory space, declaring external DMA managers, setting the number of OpenMP cores to use, and setting the L1 and L2-SRAM scratchpad sizes, occur prior to starting the timer. This timing ideology was chosen because real-world applications would likely run continuously on the DSP cores, so these initializations would only affect performance at startup. All computation, DMA-transfer time, and OpenMP management, is included in our timing. The KeyStone II instruction set includes a function to return the current cycle count from the internal cycle counter. By noting this number at the beginning and end of the program, we can calculate how many cycles were consumed in the function. The performance, measured in GFLOPS, is calculated by Equation 3. The 10^9 cycles/second factor is calculated using the 1 GHz clock speed of the device.

$$GFLOPS = \frac{Number\ of\ FLOPs}{total\ cycles} * \frac{10^9\ cycles}{second} * \frac{1\ GFLOP}{10^9 FLOP} \quad \textbf{5-3}$$

For the ODROID-C2, we followed the same ideology for timing the function. There is not a cycle counter for the ODROID-C2, so a wall-clock time function is used to measure the performance of the kernel. In this case, we can use Equation 4 to calculate GFLOPS.

$$GFLOPS = \frac{Number\ of\ FLOPs}{total\ time} * \frac{1\ GFLOP}{10^9 FLOP} \quad \textbf{5-4}$$

5.1.5 Power Measurements

Power consumption for the kernels and applications is measured using a Watts Up? Pro power meter from Vernier. The power meter measures the power drawn by the entire board by placing it between the board and wall power. A baseline power reading is taken before beginning any computation. For the K2EVM-HK, the baseline power is 23W and for the ODROID-C2 it is 5.5W. Then, the function under testing is run 10000 times in succession so that the power comes to a steady-state value. The power reported in the results follows Equation 5.

$$Power_{Computation} = Power_{SteadyState} - Power_{Baseline} \quad \textbf{5-5}$$

In this way, we record the power used solely for the computation of the function and eliminate any peripherals on the boards that are drawing power.

5.2 PHASE-2 METHODOLOGY

This section describes the method in which FFTs and SAR are executed on the HiKey and the US+ boards. An optimization of the FFTW functions to compute 2D-FFTs is also discussed. Finally, the model for projecting HPSC performance is detailed at the end of this section. The timing methodology is identical to that used for the ODROID-C2 outlined in Section 5.1.

5.2.1 FFT Methodology on HiKey and US+

Since the HiKey and US+ are ARM CPUs, the FFTW3 library is also used to compute batched 1D-FFT and 2D-FFT on these devices. In this phase of the research, both single- and double-precision FFTs were investigated after industry feedback on the first phase of research. The research for Phase-1 was published in [45] and resulted in a peer suggestion to include double-precision data, which is required for some applications Again, OpenMP is used to parallelize the functions across the Cortex-A53 cores.

An additional optimization over FFTW was discovered for 2D-FFTs in the second phase of this research. In previous research on the KS2, there were many methods investigated for performing the 2D-FFT before deciding to use a transpose buffer. We attempted to perform the second dimension of the FFTs along the columns without transposing as well as transposing the data while performing the DMA transaction from the L2 cache to DDR3. Both techniques led to too much overhead and it was concluded that performing a separate transpose was ultimately better than inefficient memory accesses along columns as in the previous two methods. We believe this extra transpose is performed in the shared-memory FFTW library, instead the second batched 1D-FFT is simply performed along the columns. From the FFTW documentation: “For the Cilk and threads versions of FFTW, we simply divide these one-dimensional transforms equally between the processors…Unfortunately, this means that the arrays being transformed by different processors are interleaved in memory, resulting in more memory contention than is desirable. [46]” The consequences of using an extra transpose for 2D-FFTs are discussed later in Section 6.0. Unfortunately, the HiKey boards became nonfunctional when trying to execute SAR across eight cores before this optimization was discovered. Therefore, HiKey FFT performance only reports the base FFTW performance.

For the US+, GbE is used to connect two boards and the FFTs are divided between them. For batched 1D-FFTs, this process is trivial since half of the batch of data is sent to the client board for computation while the server board does computation on the other half of data. Then, the client board sends its finished half of the task back to the server. For the 2D-FFT, the FFTW function had to be broken up to parallelize across boards (which led to the discovery in the previously mentioned discovery). The batched 1D-FFT along the rows of the matrix is performed identically to the regular batched 1D-FFT. After all the data has been sent back to the server, the server performs a transpose on the array of data, again using OpenMP for parallelization. This process is repeated to complete the 2D-FFT.

5.2.2 SAR Methodology on HiKey and US+

Previous SHREC students worked with AFRL to develop code to perform the SAR function on a sample of real data. AFRL members went on to refine this code, which was sent back to us for testing in this project. The code accepts a raw file as an input along with a parameters document which describes the settings used for collecting the data. The data is split into “patches” and each patch then has four functions performed on it. These functions are range compression, patch processing, range migration, and azimuth compression. These steps are further detailed in [17]. The code, in its current state, can divide the image into 9 or 35 patches. These different levels of granularity of the code change the performance of the application as further detailed in Section 6.2.7. For parallelization across US+ boards, multiple patches are sent over Ethernet for processing on the client board and the final data is sent back to the server. Ethernet benchmarking revealed that sending the entire patch at once was inefficient. Each patch was broken up into 4096 integer segments, a more efficient packet size, and then transferred to reduce communication penalty.

5.2.3 HPSC Projection Model

The HPSC modeling is conducted in several steps. First, we benchmark the performance of FFT and SAR on the HiKey to understand how performance is impacted when splitting the application across the quad-cores using the AMBA bus. For HPSC chiplet estimation, we wanted to study the most advanced ARM architecture available. We could simply use the HiKey data to project single chiplet performance since it's architecture more closely resembles a single chiplet. However, as demonstrated in the results, the US+ outperforms the HiKey because the US+ has both a larger memory bus and the memory used is faster than that of the HiKey. We used Equation 6 to estimate the percent performance increase for upgrading from a single US+ quad-core ARM Cortex-A53 to two such cores connected by an AMBA bus.

$$\%\ change\ singleboard\ to\ chiplet = \%\ change\ HiKey\ 4\ to\ 8\ cores * \frac{\%\ change\ US+\ 2\ to\ 4\ cores}{\%\ change\ HiKey\ 2\ to\ 4\ cores} \quad \textbf{5-6}$$

This equation encapsulates the performance hit contributed by the AMBA bus, which is the first term on the right side of the equation, and the difference in scalability of the processors, which is denoted by the quotient of the second and third term on the right side of the equation. The projected single-chiplet performance is then found by multiplying the percent change value found above by the four-core performance of the US+. As a simple, sample case we will assume the HiKey percent change from four to eight cores is 120%, the US+ change from two to four cores is 175%, and the HiKey change from two to four core is 150% according to Equation 6. This would result in projecting single chiplet performance to be 140% of US+ four core performance. The

US+ scales better than the HiKey in this example, therefore it is natural to project two US+ quad-cores connected by an AMBA bus to scale better (140%) than the HiKey does (120%).

It is also necessary to understand how performance is affected by parallelizing applications across chiplets. Therefore, we next benchmark Ethernet and application performance for the dual-board configuration of the US+ boards. However, we also had to account for the difference in Ethernet speeds between the US+ (GbE) and HPSC (10GbE). To estimate the performance of HPSC we needed to create a model we could validate. We estimated dual-board GbE performance with only single-board results and Ethernet benchmark results using Equation 7. The total runtime should be the combination of the compute runtime when using more resources and the added overhead from using Ethernet to communicate between boards. The first two terms in the equation estimate the time to complete the computation of the problem. The last two terms estimate the overhead added from using Ethernet.

$$Dual\text{-}board\ GbE\ Runtime\ Estimate = Single\text{-}board\ Runtime * \%\ server\ work\ after\ split + Ethernet\ Receive\ Time * Number\ of\ Transfers \quad \textbf{5-7}$$

The "% server work after split" term is application-specific. For FFTs the number is 1/2 since the amount of work has been evenly split between two boards. For SAR the term depends on how many patches are used. For 9 patches the value is 2/3 and for 35 patches it is 5/9. These values are derived from the number of sections the problem is divided into when using different numbers of cores. For example, with 9 patches executing on four cores the problem is divided into three sections with each executing four, four, and one patch, respectively. When there are 9 patches executing on eight cores, the problem is divided into two sections with each executing eight and

one patch, respectively. Therefore, the work the server board must do 2/3 of the work it previously had to do after splitting the application.

The Ethernet communication time is also included as the second term on the right side of the Equation7. Ethernet receive time was always larger than the send time which led to its use in Equation 7, as opposed to Ethernet send time. Ethernet benchmarking results are also included in Section 6.2.4. The number of Ethernet transfers varied from application to application. For batched 1D-FFT, there were two transactions: sending half of the batch to the client board at the beginning of the application and receiving half of the output at the end of the application. For 2D-FFTs, there were four transactions. Half the data is sent to the client board and then sent back after computation to the server board. The server board computes the matrix transpose and then the process is repeated. For SAR, there were 8 transactions when using 9 patches: sending 4 patches to the client board and receiving the output of those four patches from the client board. For 35 patches, 16 patches were sent to the client board for computation. The detailed report for this GbE estimation compared to the actual results, in Section 6.2.6, but the average accuracy of the estimate across all problem-sizes for FFTs is 82%, for SAR with 9 patches is 83%, and for SAR with 35 patches is 97%. Accuracy was measured using Equation 8**.**

$$Accuracy = \frac{\left\| Performance_{Actual} - Performance_{Projected} \right\|}{Average(Performance_{Actual} - Performance_{Projected})} \quad \textbf{5-8}$$

The high accuracy we achieved allows us to estimate 10GbE performance with relative confidence. Equations 9 and 10 show our projections for 10GbE and 6x10GbE, respectively, which

are slight variations of Equation 7. One caveat to Equation 9 is that 6x10GbE is likely not six times faster than 10GbE in real applications, so this value represents an upper bound estimation.

$$Dual\text{-}board\ 10GbE\ Runtime\ Estimate = Single\text{-}board\ Runtime * \%\ server\ work\ after\ split + \frac{Ethernet\ Receive\ Time}{10} * Number\ of\ Transfers \quad \textbf{5-9}$$

$$Dual\text{-}board\ 10GbE\ Runtime\ Estimate = Single\text{-}board\ Runtime * \%\ server\ work\ after\ split + \frac{Ethernet\ Receive\ Time}{60} * Number\ of\ Transfers \quad \textbf{5-10}$$

These Ethernet projection equations can also be used to estimate dual-chiplet performance when connected by a 10GbE connection and with 6x10GbE. Single-board runtime is replaced by single-chiplet performance for this computation.

5.2.4 SAR Projection for HiKey

As previously mentioned, the HiKey boards became nonfunctional during this research which occurred while gathering the results for four- and eight-core SAR performance. Estimating HPSC SAR performance requires these values, as shown in Equation 6. Therefore, these values were estimated using HiKey 2DFFT performance and US+ SAR performance. Four-core performance can be estimated using existing HiKey and US+ performance data, as shown in Equation 11. This performance was projected similarly to single-chiplet projections from Equation 6. The first term represents the cost for moving from two to four cores and the second and third terms comprise the difference in scalability between the processors. HiKey four-core performance is found by multiplying this percentage change by the two-core performance measurement.

$$\%\ change\ HiKey\ 2\ to\ 4\ cores = \%\ change\ US+\ 2\ to\ 4\ cores * \frac{\%\ change\ HiKey\ 1\ to\ 2\ cores}{\%\ change\ US+\ 1\ to\ 2\ cores} \quad \textbf{5-11}$$

To project eight-core performance, we need to use HiKey 2D-FFT performance data since it is the only data that encapsulates the impact the AMBA bus has on performance. To determine if this is a valid approach for estimating the scalability of SAR we reviewed two studies, [47] and [48], which profiled runtime for the kernels within the SAR application. From these papers we derived the percentage of SAR runtime dedicated to FFTs and transposes, the two kernels within 2D-FFTs. For the TI scheme we assume FFTs comprise roughly two thirds of range and azimuth compression, since range and azimuth compression's main kernels are FFT, filter multiplication, and inverse Fast Fourier Transform (IFFT). With this assumption, and the profile data from Wang et al., we conclude FFTs and transposes account for roughly 60% of the application's runtime with the remainder composed of multiplications and data management [47]. Przytula et al. conclude that roughly 72% of the application runtime is comprised of FFTs and transposes, with multiplications and data management accounting for the remainder [48]. Since FFTs and transposes account for a significant portion of SAR, using 2D-FFT performance data for scaling is fair for a naïve estimation. To perform the estimation, we calculated the average percentage change of runtime from four to eight cores on the higher 2D-FFT sizes (SAR FFT size is larger than 1K) which was 83.5%. Therefore, estimated eight-core runtime is 83.5% of the estimated four-core runtime.

6.0 RESULTS

No research is complete without discussing the results. This section details the benchmarking results and performance estimations for the candidate architectures and discusses the trends seen in the results.

6.1 PHASE-1 RESULTS

This section describes the results and accomplishments of Phase-1 starting with the FFT results. Performance here, and throughout the rest of this section, is described with units of giga floating-point operations (GFLOPS) and mega floating-point operations (MFLOPS). Then, CAF performance and power vs performance results will be discussed. Performance is reported in GFLOPS in this section along with performance-per-watt in GFLOPS-per-watt (GFLOPS/W).

6.1.1 FFT on K2EVM-HK Results

Before creating the direct memory access scheme, we benchmarked the TI FFT libraries to see determine which would provide the best foundation for our work. Figure 10 shows the performance of DSPLIB, FFTLIB, and FFTLIB with the ping-pong scheme using L1-SRAM for batched 1D-FFTs on a single core. For these problem-sizes, the FFT and the batch size are the same. For example, a 512 FFT / batch size in the graph legend indicates that 512 1D-FFTs of size 512 were performed. We chose this size convention so 2D-FFTs were calculated with a comparable

problem-size. Across the problem-sizes investigated, FFTLIB outperformed DSPLIB, which is why we chose it for a baseline library. Further, the ping-pong scheme also outperformed FFTLIB across all problem-sizes investigated for single-core, 1D-batched FFTs. The average performance improvement of the ping-pong scheme over baseline FFTLIB is 32% for a single core.

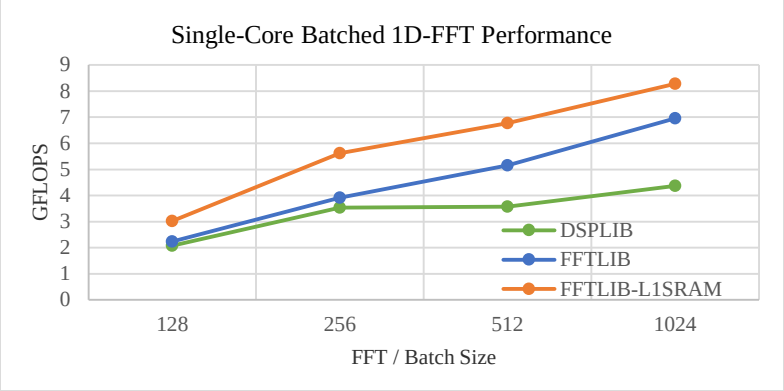


Figure 10. K2EVM-HK single-core, single-precision, batched 1D-FFT performance with different libraries

Before continuing FFT result discussion, we present the line sizes used for our ping-pong scheme. As discussed earlier, to completely optimize our scheme we used different transfer sizes for each of the problem-sizes investigated, as shown in Table 1. L2 lines refers to the number of sets of FFTs (each being a line) placed into the L2-SRAM in one transfer. L1 lines follow similarly for L1-SRAM.

Table 1. Optimal FFT Line Size for Ping-Pong Scheme

128 FFT Size			256 FFT Size		
# of Cores	L2Lines	L1Lines	# of Cores	L2Lines	L1Lines
1	16	4	1	16	2
2	8	4	2	16	2
4	8	4	4	4	1
8	8	4	8	4	2
512 FFT Size			1024 FFT Size		
# of Cores	L2Lines	L1Lines	# of Cores	L2Lines	L1Lines
1	16	1	1	16	1
2	16	1	2	16	1
4	16	2	4	8	1
8	2	1	8	4	1

Unfortunately, there was not a specific rule that appeared to determine what combination of line sizes led to the best performance. There were some general trends, however. Using 16 L2 lines was always best for lower numbers of cores, suggesting that we could use a higher percentage of the memory bandwidth available when only one core is in use. As more cores are used, and the bandwidth is exhausted, using less L2 lines is preferred.

For the remainder of this section, we will only be comparing the difference between the FFTLIB with and without our ping-pong scheme to judge the effectiveness of the scheme. Interesting trends are revealed as FFTs are parallelized across more cores. Figure 10 shows the performance of batched 1D-FFTs for different problem-sizes with and without L1-SRAM usage. The performance benefit for both schemes decreases as the number of cores increases.

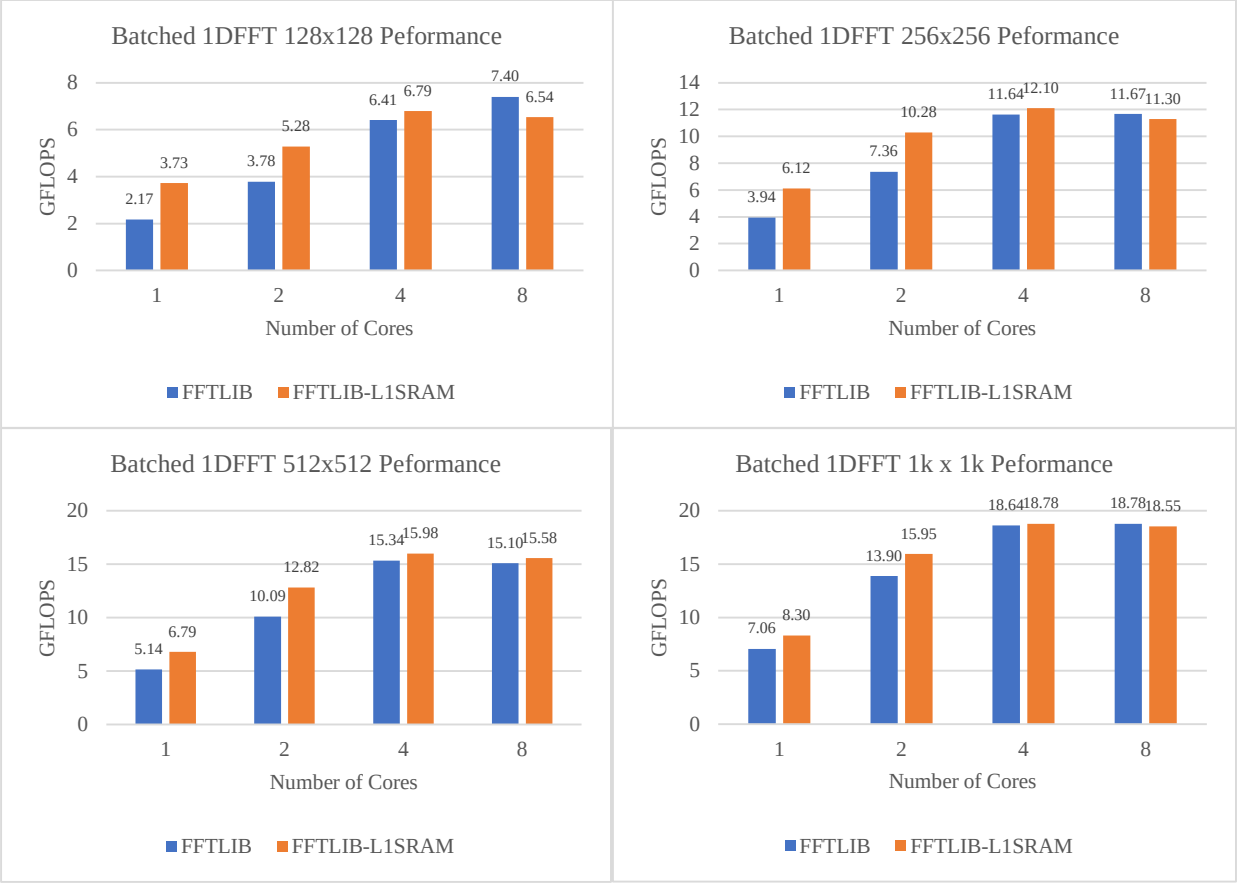


Figure 11. K2EVM-HK batched 1D-FFT performance with and without L1-SRAM in the DMA scheme

The dip in performance from four to eight cores for each of the problem-sizes in Figure 11 occurs because the FFT is a memory-bound computation. The bandwidth becomes increasingly limited as more cores are used. Transfers take longer because of the limited bandwidth, causing the program to stall while waiting for the transfers to complete. This effect is exacerbated when performing DMA-transfers to L1-SRAM, resulting in a dip in performance moving from four to eight cores. Using L1 partially as SRAM also limits how much L1 cache is available, which gives the cache controller less flexibility with line replacement. Another trend shown in this graph is that the ping-pong scheme performs better than the base FFTLIB scheme with fewer cores. When

enough bandwidth is available, the ping-pong scheme provides an advantage over the FFTLIB scheme for batched 1D-FFTs.

Figure 12 shows the same problem-sizes and core numbers as Figure 11, but for 2D-FFTs. Here, a problem-size of 256 indicates a 256 x 256 2D-FFT is being performed. The extra transposes at larger sizes may hide some of the transfer time, leading to a small gain rather than a dip. Small problem-sizes especially suffer from the new scheme for 2D-FFTs. The smaller FFT's transposes make up a larger portion of the total runtime, so our scheme is not beneficial for these instances. For larger problem-sizes, however, our scheme appears to be effective with performance gain up to 41%. Larger problem-sizes also have less of a performance dip from four to eight cores, likely because the transfer overhead was less of a factor due to the increased computation time.

Figure 12. K2EVM-HK 2D-FFT performance with and without L1-SRAM in the DMA scheme

Figure 13 shows the average speedup using our scheme for different numbers of cores and Figure 14 shows the average speedup using our scheme for different problem-sizes for batched 1D-FFT and 2D-FFTs. These two graphs highlight many of the trends previously discussed. Figures 13 and 14 show that the ping-pong scheme provides significant performance benefit on average when using less cores or for smaller problem-sizes for batched 1D-FFTs. Meanwhile, the scheme provides near constant performance gain across different core numbers and becomes better for higher problem-sizes for 2D-FFTs.

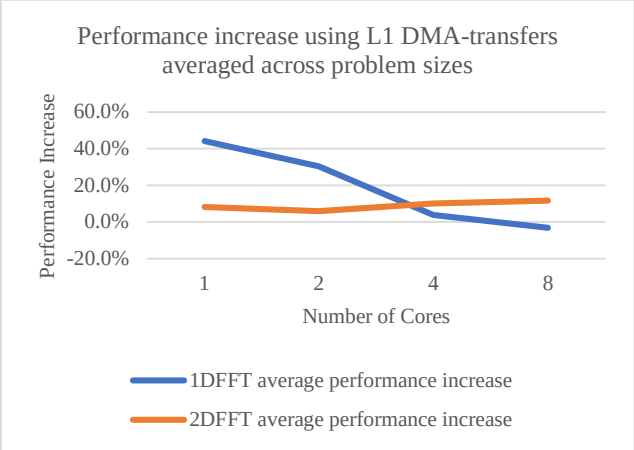


Figure 13. K2EVM-HK **p**erformance increase using L1 DMA-transfers averaged across problem-size

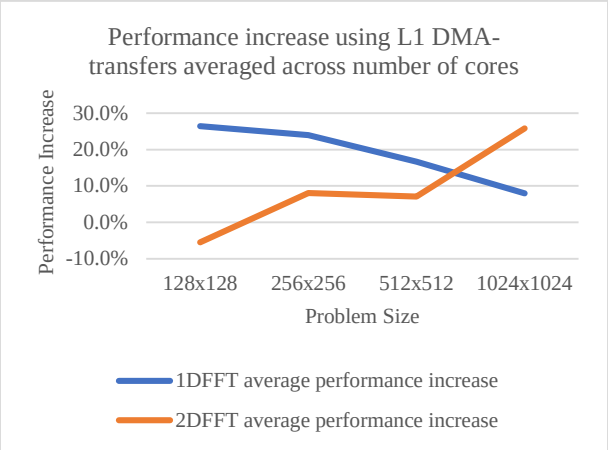


Figure 14. K2EVM-HK **p**erformance increase using L1 DMA-transfers averaged across number of cores used

6.1.2 FFT on ODROID-C2 Results

Figure 15 shows the performance for FFTs on the ODROID-C2. Unlike the K2EVM-HK, there were few unexpected trends on this processor. For batched 1D-FFTs, the performance scales almost linearly as more cores are used. There are not any memory bandwidth issues that affect how well the algorithm scales on this processor. For the 2D-FFT, 128x128 seems to perform much better than the rest of the problem-sizes which could be caused by the processor being able to handle those data sizes more efficiently in cache. However, accessing the columns of the array for the 2D-FFT could be causing increasing overhead as the problem-size grows, which is further discussed in Section 6.2.2.

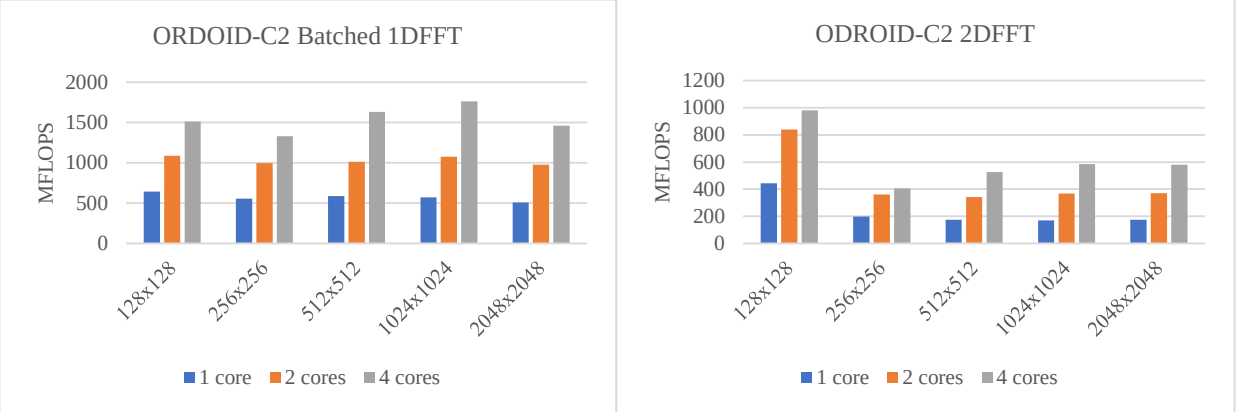


Figure 15. ODROID-C2 FFT performance

6.1.3 CAF Results

Figure 16 shows CAF performance on both the K2EVM-HK and ODROID-C2. K2EVM-HK outperforms the ODROID-C2 here for a variety of reasons. The first is that the K2EVM-HK has a

larger memory bandwidth than the ODROID-C2 due to the use of faster memory and a larger memory bus-width. Since the FFT is a memory-bound function, having a larger memory bandwidth will lead to greater performance. Also, the KS2 architecture allows for more flexibility in its memory-transfer scheme and has DSP units that can perform FFTs more efficiently. Note that there is no ODROID-C2 data point for 8 cores since there are only 4 cores available.

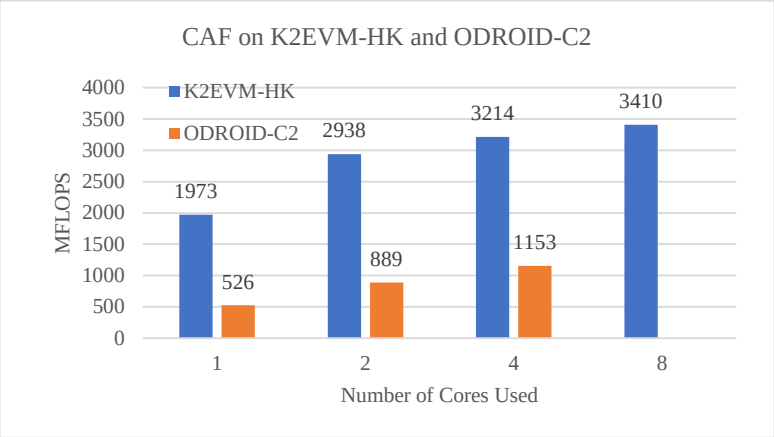


Figure 16. CAF performance on K2EVM-HK and ODROID-C2

6.1.4 Power Results

While performance is an important metric, for space applications power can be just as important. Figure 17 shows the power consumption, in Watts, along with performance for the largest problem-size for FFTs and for CAF. The power and performance follow the same trends on both processors, including the dip at eight cores on the K2EVM-HK. In general, as more cores are used more power is consumed. The dip in power for four to eight cores on the K2EVM-HK happens for

the same reason as the dip in performance. When stalls occur, there are fewer computations performed, and therefore less power consumed.

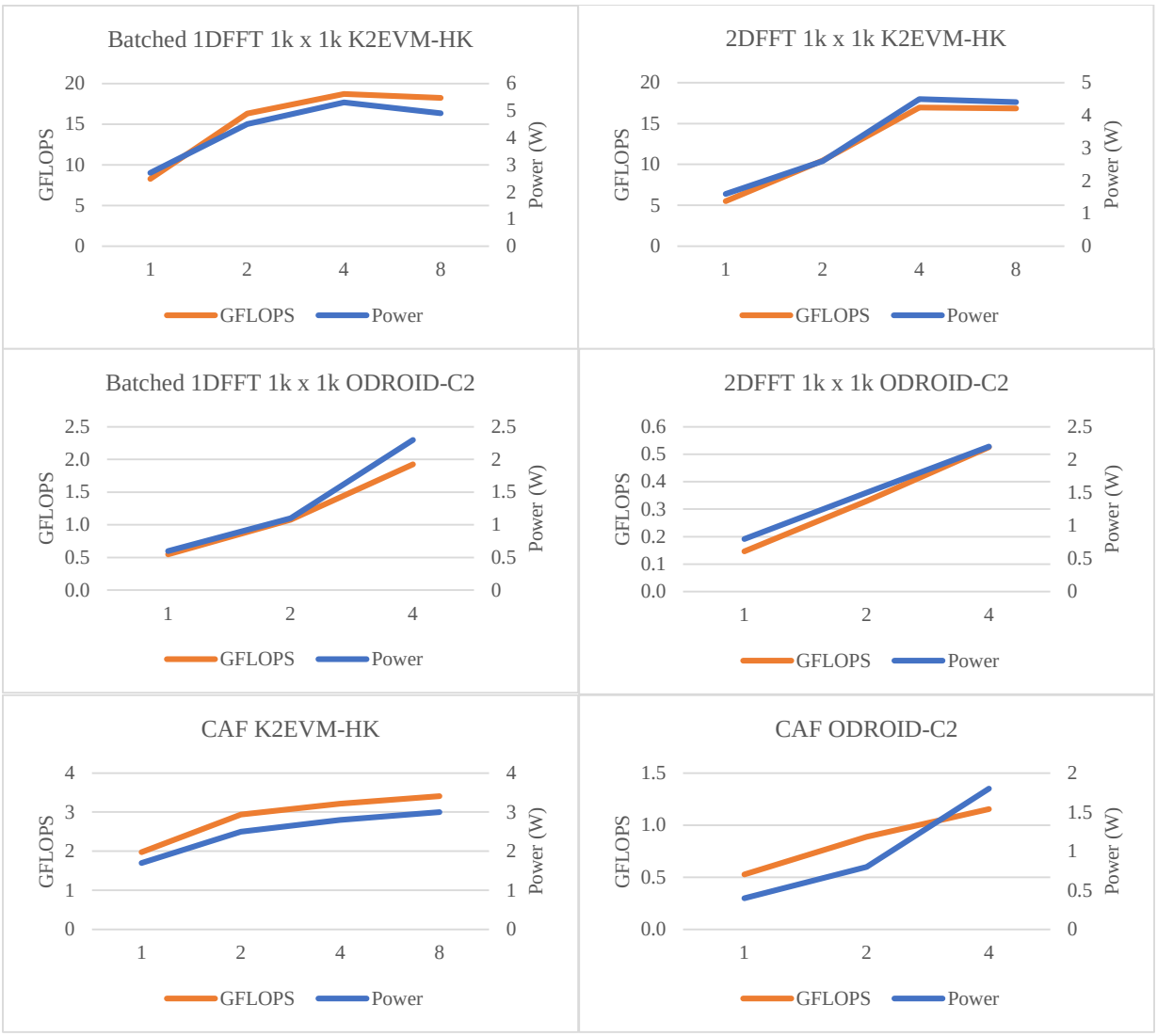


Figure 17. Performance and power results for FFTs and CAF on the K2EVM-HK and ODROID-C2 boards

6.1.5 Performance and Power Comparisons

Figure 18 compares performance and performance-per-watt for the largest problem-size FFT and the CAF algorithm. For the reasons stated above, the K2EVM-HK exhibits superior performance. The performance of FFTs scale much better on the K2EVM-HK as the problem-size increases because of the differences in memory architecture. Even though the ODROID-C2 consumed less power, the lack of performance led to poorer performance-per-watt. It is notable that, overall, the ODROID-C2 does use less board power than the K2EVM-HK and that it was significantly easier to program the device for these functions. The difference in performances for CAF on each board is comparable to the difference in performance for 128 batched 1D-FFT, likely because the input size of the CAF is similar to this FFT size. If the CAF input size was increased, the performance discrepancy between the two devices would likely increase.

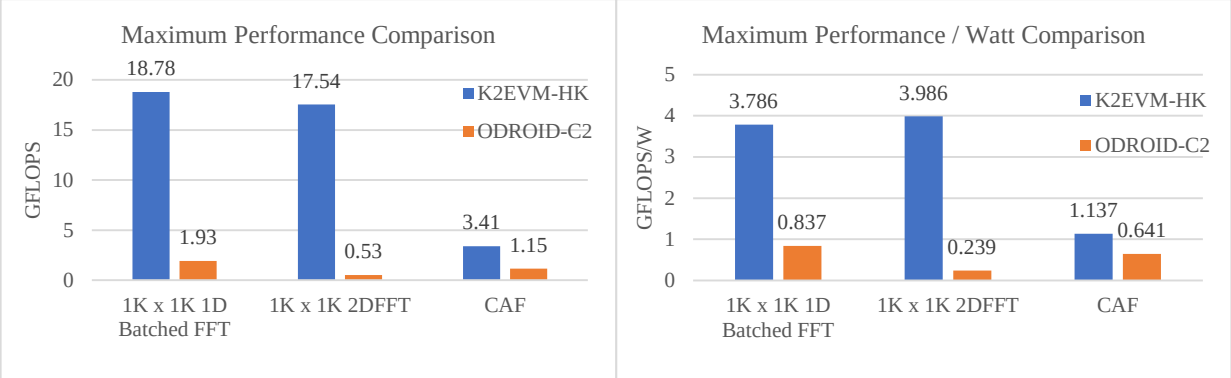


Figure 18. Maximum performance and performance-per-watt application comparison

6.2 PHASE-2 RESULTS

In this section we will discuss the results from Phase-2 of the research, including FFT and SAR single board performance for the HiKey and US+ boards, Ethernet benchmarking on the US+, FFT and SAR results when split between two US+ boards, an overall comparison of performance between US+ and HiKey, and HPSC projection modeling. Performance for FFTs is presented in MFLOPS. Performance for SAR is reported by runtime as opposed to FLOPS because the application is too large to accurately determine the number of operations performed.

6.2.1 HiKey FFT Results

Figure 19 depicts the single- and double-precision performance, respectively, for batched FFTW 1D-FFT on the HiKey. The problem-size indicated in the graph denotes both the size of the FFTs and the number of batches of those FFTs, as in Phase-1. The main difference between the two precisions is performance. Double-precision requires about double the time of single-precision. The overall performance improves as the problem grows due to Gustafson's law. The overhead due to the parallelization of the problem becomes amortized as the problem-size increases, increasing performance.

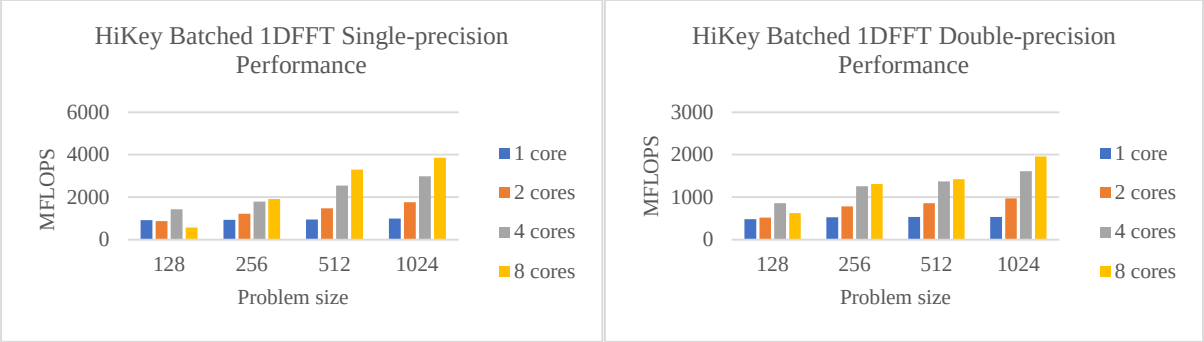


Figure 19. HiKey single- and double precision batched 1D-FFT performance

The most interesting insight from Figure 18 stems from the performance gains for small problem-sizes when moving from four to eight cores. When using eight cores, the HiKey must distribute some of the work using the AMBA bus connecting the quad-cores CPUs which incurs a non-negligible overhead. This overhead becomes more apparent when comparing the results between four and eight cores. For a problem-size of 256 the gain is minimal and for 128 the gain is negative. This performance trend occurs because the overhead of using the AMBA bus does not become a small percentage of the runtime of the kernel until larger problem-sizes are used. Similar logic follows for the one-to-two core performance trend for small problem-sizes where there is an added overhead to dispatching across cores.

The single- and double-precision performance of FFTW 2D-FFT are also performed on the HiKey, shown in Figure 20. To reiterate, the multiple-transpose optimization discovered for 2DFFTs could not be used on the HiKey board as described in Section 5.2.1. The four-to-eight-core performance trend is followed in a similar fashion as the batched 1D-FFT. However, unlike 1D-FFTs, 2D-FFT performance decreases as problem-size increases. As mentioned in Section 6.1.2, performance loss is likely due to the overhead of accessing the array along the columns, which increases as the problem-size increases. When the FFTW optimization results are presented,

where the access patterns are not along the columns, the performance remains relatively constant across problem-size.

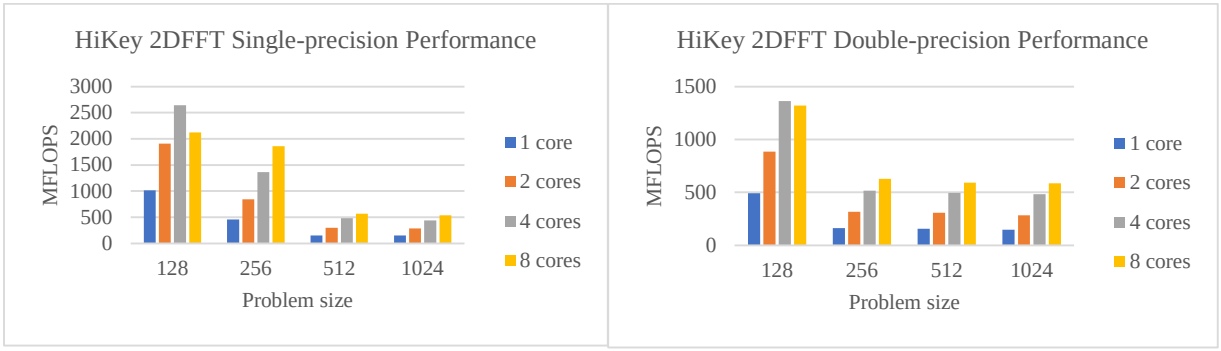


Figure 20. HiKey single- and double precision 2D-FFT performance

6.2.2 US+ FFT Single-Board Results

Figure 21 depicts the FFTW results for single- and double-precision batched 1D and 2D-FFTs. These results follow similar trends to those obtained using from the HiKey device in the previous subsection However, the performance increase with problem-size increases for the US+ is not as drastic as for the HiKey but overall the US+ has better parallel efficiency. The US+'s larger memory bus width might play a factor here along with other architectural details not observable with the tools available for this research.

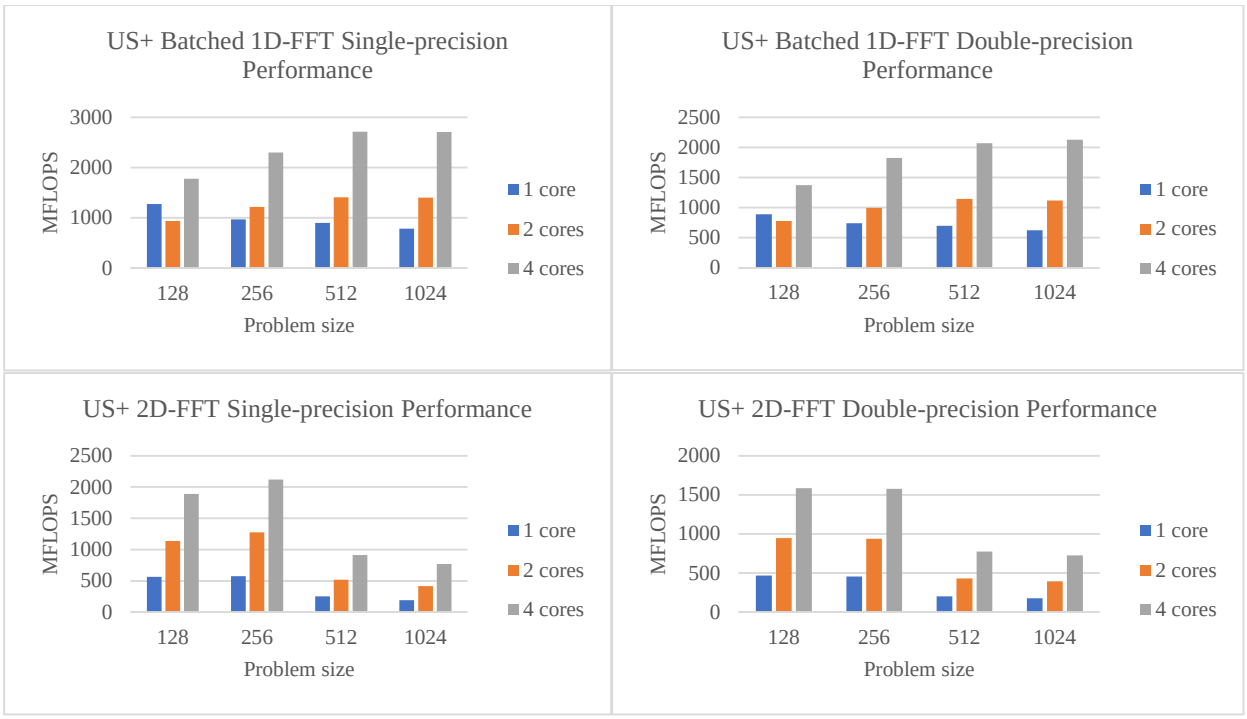


Figure 21. US+ batched 1D-FFT performance

Figure 22 is the result of using the custom transpose scheme for 2D-FFTs. Note that the performance of the application is now relatively constant across all problem-sizes rather than decreasing as problem-size increased. The overhead imposed by the transpose grows proportionally to the problem-size performance increase in this case. By contrast, the overhead of accessing along the columns grows faster than the problem-size performance increase when using base FFTW.

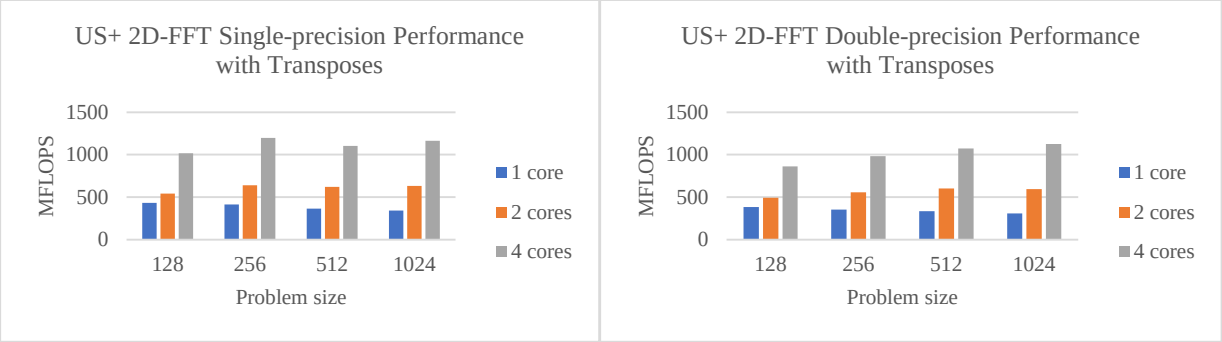


Figure 22. US+ 2D-FFT performance

Figure 23 shows the percent improvement of this new method with transposes over the base FFTW method. For smaller problem-sizes, base FFTW is better because the penalty of accessing the columns of the matrix is less than the overhead of an additional transpose. Once the problem-size becomes sufficiently large, this trend is reversed, and the transpose incurs less overhead than accessing along the columns.

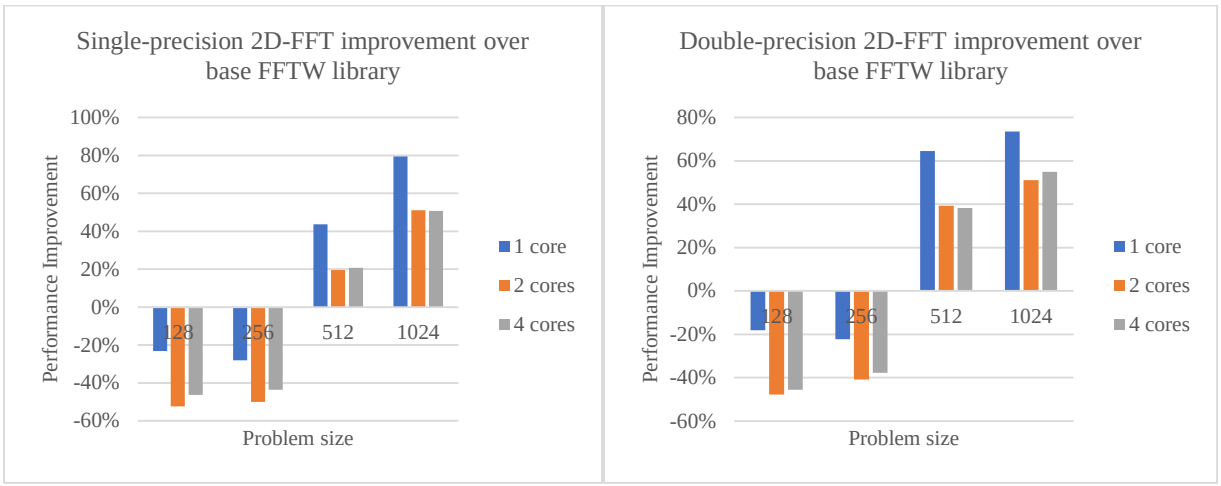


Figure 23. 2DFFT improvement over base FFTW scheme

6.2.3 FFT Board Comparison

Figure 24 shows the FFT performance comparison using only the base FFTW scheme. One of the main differences between the boards is the difference in memory. The US+ has a 64-bit bus width connecting the ARM cores to DDR4 memory whereas the HiKey has a 32-bit bus width connecting the ARM cores to DDR3 memory. As a result, the US+ performs better or near the same for this memory-bound application. For higher problem-sizes of batched 1D-FFT, the HiKey might perform better due to a difference in the Linux builds. HiKey's build environment allows the CPU to use a frequency governor to a performance mode, which is not available on the US+. Something similar might be happening for the 128-problem size 2D-FFT. Unfortunately, we do not possess the tools or documentation necessary to fully these differences on a fundamental level. The US+ is used to project the HPSC chiplet performance, since it has overall better performance.

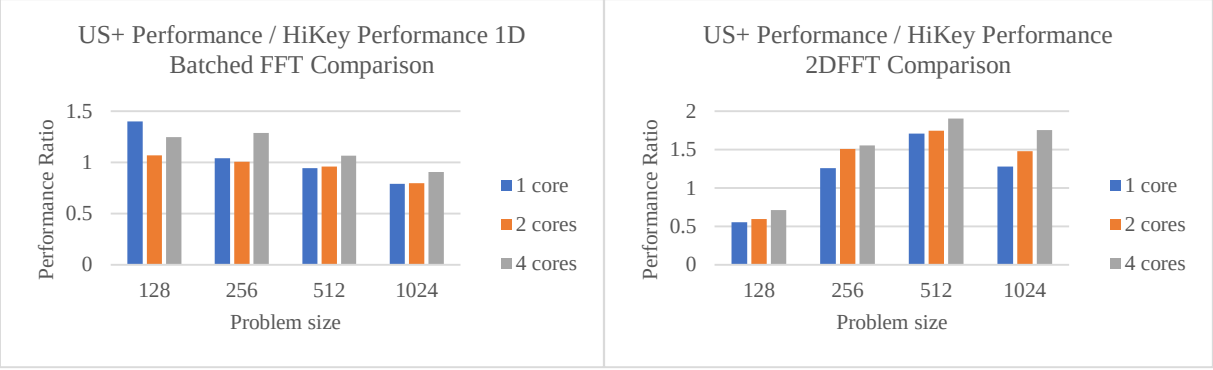


Figure 24. US+ Performance / HiKey Performance FFT Comparison

6.2.4 US+ Ethernet Results

To estimate dual-board performance, we benchmarked the GbE interface on the US+. We used the socket functions send and receive for this benchmark. Figure 25 shows the latency of each function for different packet sizes. The overhead for sending anything over Ethernet can be approximated by the time that it takes to send one integer. This number converges to 4.5ns.

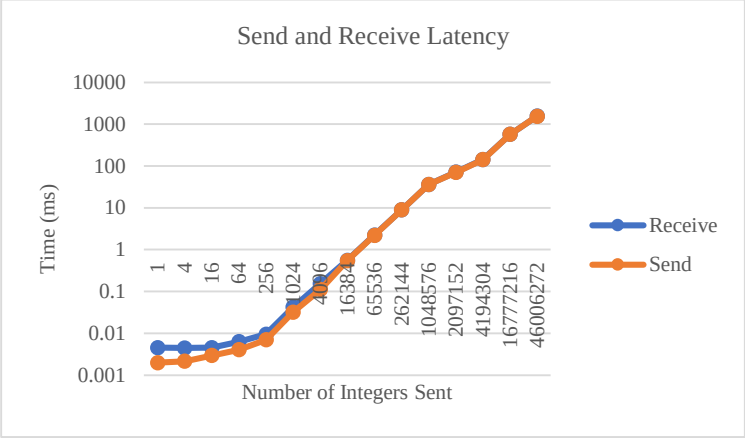


Figure 25. Send and receive Ethernet latency between US+ boards

Using this data, we can translate the throughput for each of the packet sizes, as shown in Figure 26. As the number of integers sent reaches an appropriate size the bandwidth approaches 1Gb/s as expected for a GbE interface. There is a range of packet sizes from 256 to 1048576 which allows for a bandwidth near 1 Gb/s. After the 1048578 packet size, there is a severe degradation in performance. The last data point, 460006272, is the size of each patch within the SAR application when using 9 patches. The poor performance for this packet size is the reason the SAR

patch transfer was broken up into 4096 integer chunks. This change increased the performance of SAR by 14%.

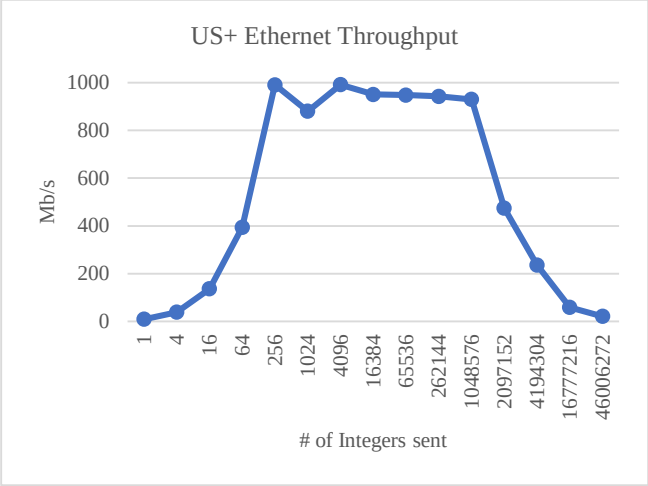


Figure 26. US+ Ethernet throughput between boards

6.2.5 Dual-Board US+ FFT Results

Figure 27 shows the dual-board performance for the US+. By necessity, the function uses our custom FFT scheme since the function had to be broken up across boards. The performance is significantly worse than single-board performance because of the immense overhead incurred from Ethernet. The same general trends continue here as in single-board performance. For this kernel, parallelizing across two boards connected by GbE does not lead to performance benefits.

Figure 27. US+ Dual-board FFT single-precision performance

Figure 28 shows the accuracy of the performance estimation for dual-board performance using Equations 9 and 10. The model is poorer for smaller problem-sizes, which can be explained by the observed variance for Ethernet performance which is greater at smaller problem-sizes. Overall, the accuracy of the model is 82% and for the larger problem-sizes of 512 and 1024 it is 87% accurate. Batched 1D-FFT proved to perform closer to the model, especially at the higher problem-sizes which are 96% accurate. The model appears to be more accurate at depicting performance for applicationns with simple Ethernet send and receive patterns.

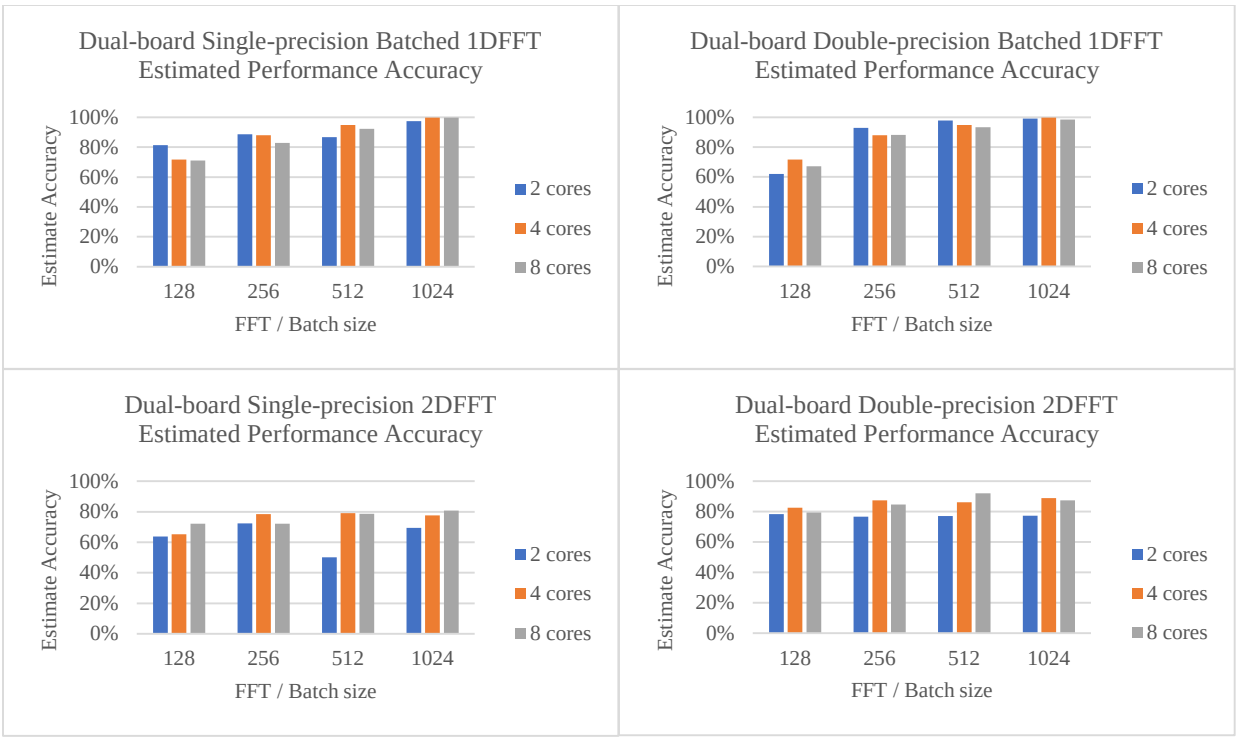


Figure 28. Dual-board FFT estimated performance accuracy

6.2.6 Dual-board US+ FFT with 10GbE and 6x10GbE Projections

Since our dual-board estimates were sufficiently accurate, we projected dual-board US+ performance with both 10GbE and 6x10GbE as the interconnect between the US+ boards. Figure 29 shows these projections along with the single-board performance and dual-board performance for comparison. These projections are an intermediate step to projecting HPSC performance. Ethernet performance directly impacts system performance for these small kernels. Using GbE is substantially worse than using a single US+ board. Using 10GbE leads to better performance over single board with the exception of smaller problem-sizes on 2D-FFTs. Even eight cores with 10GbE can be worse than single-board performance in those cases.

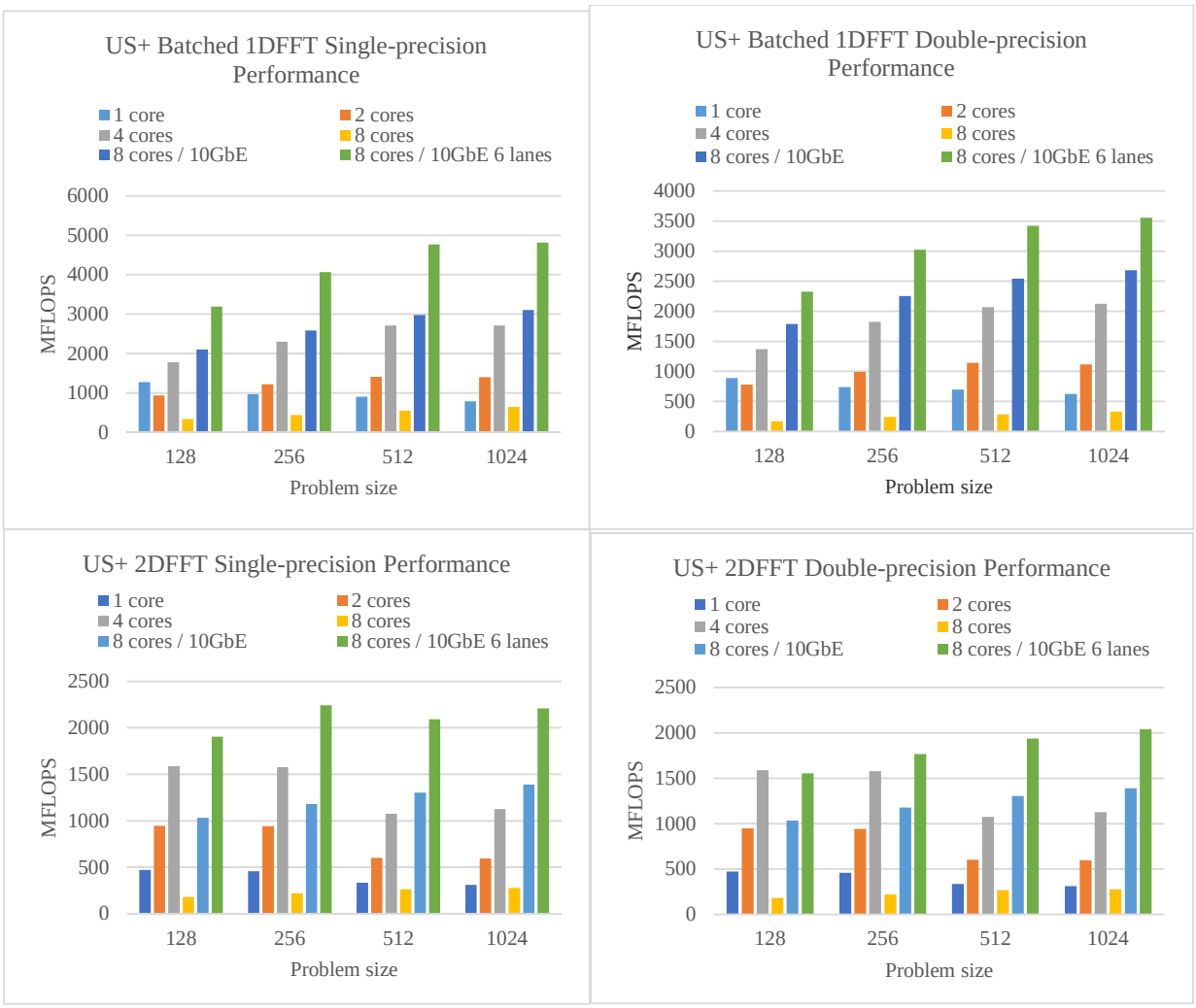


Figure 29. US+ FFT performance with 10GbE and 6x10GbE estimations

6.2.7 US+ and HiKey SAR Results

Figure 30 shows the results of running SAR on the HiKey and US+ boards with both levels of granularity. Resources with an asterisk next to them indicate they were estimated as described in Section 5.2.3. 9 patch granularity consistently performed better than 35 patch granularity because there is less overhead when using 9 patches. For example, the input file and output files are accessed less frequently when using less patches. Again, the greater memory bandwidth of the

US+ leads to greater performance compared to the HiKey. Additionally, unlike in FFT testing, using GbE between the US+ boards led to a performance increase. SAR is complex enough that the time to transfer the patches to the other board, estimated to be 13.8 and 27.6 seconds for 9 and 35 patches respecitvely, is less than the runtime reduction from using more cores. SAR US+ dual-board performance is 97% and 83% accurate to the dual-board projection model for 9 and 35 patches, respectively. Like the FFTs, the simpler Ethernet transaction pattern with 9 patches was better represented by the model.

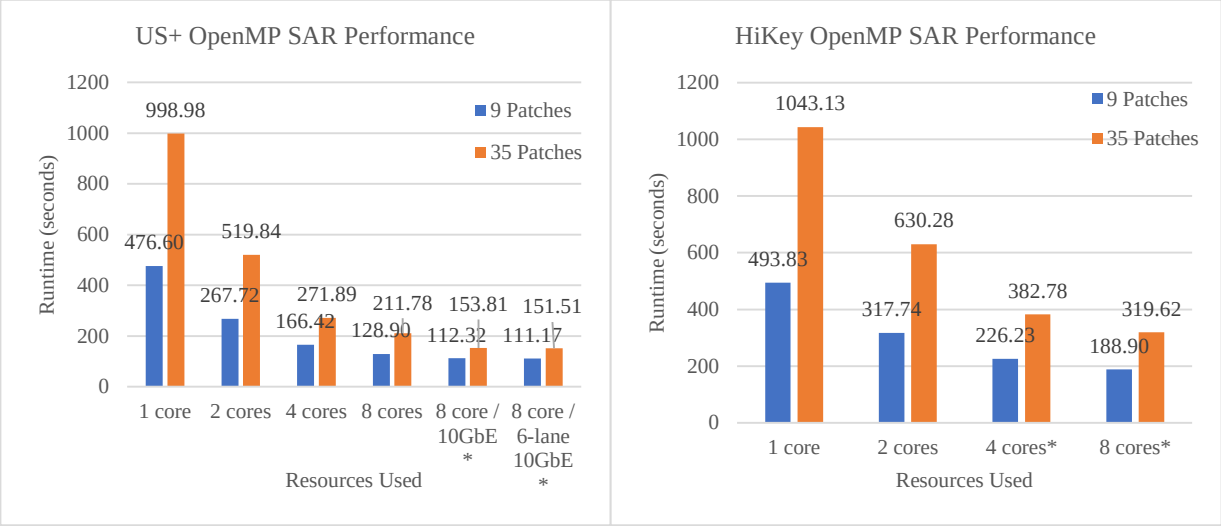


Figure 30. SAR performance on the HiKey and US+ in single and dual-board configurations

6.2.8 HPSC Projections

HPSC performance projections are the culmination of the Ethernet, FFT, and SAR benchmarking above. Figure 31 shows the FFT and CAF HPSC performance projections. Each graph has the maximum US+ single-board performance as a reference point. 1 Chiplet designates an estimate of

a processor composed of two US+ quad-core ARM Cortex-A53s connected by an AMBA bus, 2 Chiplet designates an estimate of two chiplets connected by 10GbE, and 2 Chiplet 6-lane designates an estimate of two chiplets connected by 6x10GbE.

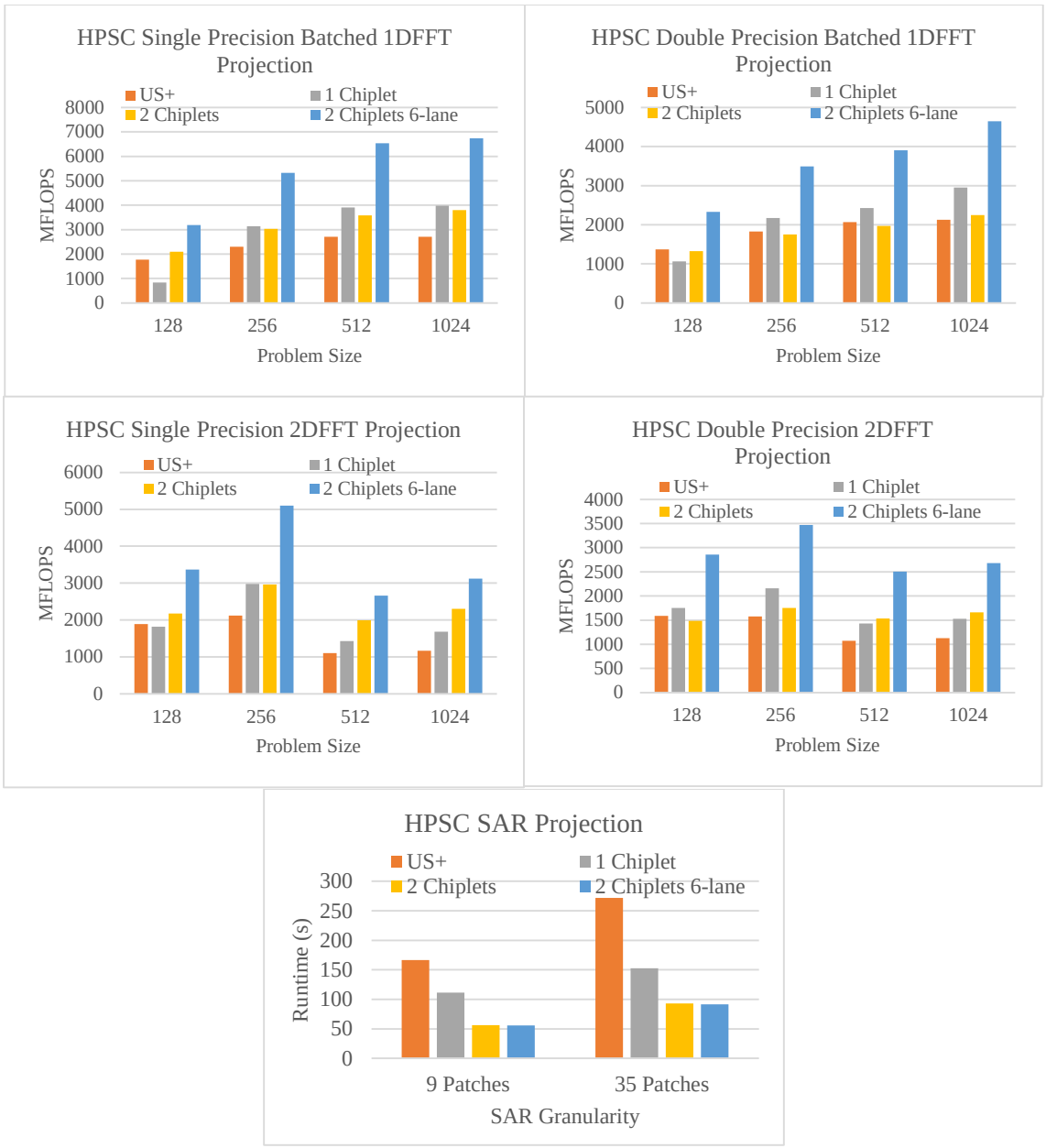


Figure 31. HPSC projections for FFTs and SAR

For small problem-sizes in FFTs, using a chiplet over a single board provides little, and sometimes negative, performance gain because of the overhead caused by the AMBA bus. Similarly, the FFT problem-size needs to be sufficiently large to gain performance with two chiplets because of the Ethernet overhead. 6x10GbE provides the best performance for all FFT problem-sizes but this is an upper bound estimate because 6x10GbE is likely not six times faster than 10GbE. On average, one chiplet provides a performance gain of 1.22 over a single US+ board, two chiplets provide a performance gain of 1.28, and two chiplets with 6x10GbE provide an upper bound performance gain of 2.17.

SAR projects better to HPSC than the FFT kernels. On average, one chiplet provides a performance gain of 1.63 over a single US+ board, two chiplets provide a performance gain of 2.94, and two chiplets with 6x10GbE provide an upper bound performance gain of 2.97. There is little benefit to using 6x10GbE over 10GbE because of the proportionally low Ethernet overhead. Additionally, the naïve estimation for HiKey four- and eight-core performance results in a chiplet estimation that closely matches the US+ eight-core 10GbE estimation. These two estimations should be similar because the AMBA and Ethernet overheads are a minuscule portion of the total runtime.

7.0 CONCLUSIONS

In Phase-1 we studied the TI K2EVM-HK, which features eight DSP cores, and the ODROID-C2, a quad-core ARM Cortex-A53 CPU, to determine if these architectures are suitable for future space missions. We developed a custom DMA-transfer scheme for the KS2 that accelerated batched FFTs. Our scheme performs particularly well when using a single core, providing a 44% speedup over the baseline method for batched 1D-FFTs. However, when eight cores are used the performance gain decreases to an average of 6%. Our scheme is not desirable for 2D-FFTs with small problem sizes, but once the problem size is large enough it can provide a performance gain, as a 32% speedup was observed for 1K × 1K 2D-FFTs over the baseline FFTLIB. We observed that memory bandwidth plays a large part in the efficacy of this DMA-transfer scheme with performance occasionally dipping from four to eight cores. If a radhard version of this device was created, designers could use less cores to achieve similar performance for FFTs and similar kernels. Additionally, the ping-pong scheme can be applied to other batched computations with the caveat that some manual tuning of transfer sizes is needed to obtain the best performance. The FFT performance on the K2EVM-HK is always better than on the ODROID-C2, which provided 9.7 times the performance for 1K × 1K batched 1D-FFTs. It also surpassed the ODROID-C2 in performance-per-watt by 4.1 times.

Additionally, CAF was benchmarked on each device. The K2EVM-HK provided 3 times the performance and 1.7 times the performance-per-watt than the ODROID-C2. The performance margin between processors observed in FFT studies was smaller with CAF. The CAF input size was smaller than the 1K × 1K FFT and might not have saturated the memory as the FFT did. Nevertheless, the K2EVM-HK proved to be a more powerful device, but the ODROID-C2 had

lower total power usage, is lower cost, and is easier to program. Additionally, the ODROID-C2 had better parallel efficiency. These factors, along with NASA and DOD goals to use multiple ARM processors together, suggest ARM processor performance can scale well in the future.

In Phase-2 we benchmarked batched 1D-FFTs, 2D-FFTs, and SAR on the octa-core ARM Cortex-A53 LeMaker HiKey and the quad-core ARM Cortex-A53 Zynq UltraScale + in single-board and dual-board configurations. US+ performs better than HiKey for nearly every problem-size because it uses a faster memory and a larger memory bus width. The results revealed the overhead incurred by adding an AMBA bus or Ethernet connection to the system. In some instances, parallelizing across more resources using these interfaces resulted in slower system performance. FFTs, in particular, are affected by this overhead because their computation time is comparable to the overhead. SAR computation time is significant enough to outweigh the overhead time and parallelizing with these interconnects always leads to speedup. We created a model to estimate dual-board US+ performance for FFTs and SAR. This model had an average accuracy of 82% for FFTs and 90% for SAR. This high accuracy let us expand the model to predict dual-board performance with different interconnect topologies, namely 10GbE and 6x10GbE. The model could also be used for other serial connections, such as SRIO and SerDes, and future 10GbE and 6x10GbE benchmarking would increase the accuracy of the model. An optimization for the 2D-FFT function in the FFTW library was also found when splitting the function across the US+ boards which led to an average speedup of 1.44 for larger FFT sizes.

We used the benchmarking results to project the performance of the future HPSC chiplet and multiple HPSC chiplets connected by 10GbE and 6x10GbE with our model. Parallelizing FFTs across multiple chiplets is an inefficient use of resources with an upper bound performance increase of 2.17 over single-board US+ when using two chiplets connected by 6x10GbE. By

comparison, SAR has a better parallel efficiency with 1.63 times speedup over single-board US+ by using just one chiplet and 2.94 times speedup when using two chiplets. The scalability difference in FFTs and SAR suggest that coarser granularity for parallelization is preferable on this system. SAR is a computationally relevant application to NASA and AFRL and such scalability is auspicious for the future of the HPSC chiplet approach.

To summarize, this research showcased two distinct architectures and introduced new optimizations for signal-processing functions. The KS2 architecture provides high performance with the ability to tune the memory access scheme to the application. Our ping-pong scheme offers a new way to accelerate batched functions on the architecture. We also found that adding transposes to multi-dimensional FFTs can greatly increase performance for larger problem-sizes by eliminating column-wise memory accesses. The ARM Cortex-A53 architecture provides a simple-to-use, low power platform that scales signal-processing applications well, especially when paired with a large memory bus width and fast memory. Boeing's HPSC chiplet approach for future space missions will be a powerful platform based off the emulation studies conducted in this research, especially when using coarse-grained task parallelization. Our model for projecting HPSC performance provides notable accuracy. Supplementary benchmarking into other serial interfaces in the future could augment the model's accuracy.

BIBLIOGRAPHY

[1] Boyle, Alan. “One small teraflop: HPE’s supercomputer takes a giant leap on the space station.” *GeekWire*. 20 Sep. 2017. Web. 17 Mar. 2018. https://www.geekwire.com/2017/one-small-teraflop-hpes-supercomputer-takes-one-giant-leap-space-station/

[2] Melesse, Assefa M., et al. "Remote sensing sensors and applications in environmental resources mapping and modelling." Sensors 7.12 (2007): 3209-3241.

[3] “Satellites.” *European Space Imaging.* Web. 3 Mar. 2018. http://www.euspaceimaging.com/satellites/

[4] Yang, Jun, et al. "Introducing the new generation of Chinese geostationary weather satellites, Fengyun-4." *Bulletin of the American Meteorological Society* 98.8 (2017): 1637-1658.

[5] Andreas, Nancy S. "Space-based infrared system (SBIRS) system of systems." *Aerospace Conference, 1997. Proceedings., IEEE*. Vol. 4. IEEE, 1997.

[6] Schaire, Scott, et al. "NASA Wallops Flight Facility-Morehead State Ground Network for Small Satellite Mission Operations." *SpaceOps 2014 Conference*. 2014.

[7] “Next generation processor for on-board payload data processing application.” *European Space Agency.* 26 Oct. 2007. Web. 14 Apr. 2017. http://spacewire.esa.int/edp-page/documents/NGDSP%20Round%20Table%20Synthesis%20V1.0.pdf

[8] Kuo, Sen M., Bob H. Lee, and Wenshun Tian. *Real-time digital signal processing: fundamentals, implementations and applications*. John Wiley & Sons, 2013.

[9] G. Mounce, J. Lyke, S. Horan, W. Powell, R. Doyke, and R. Some, “Chiplet based approach for heterogenous processing and packaging architectures.” *IEEE Aerospace Conference*, 5-12 March 2016.

[10] “High Performance Spaceflight Computing (HPSC) Processor Chiplet,” *FedBizOpps*, 20-Jun-2016. Web. 5 Oct. 2017. https://www.fbo.gov/index?s=opportunity&mode=form&tab=core&id=eefe806f639ae00527a13da6b73b3001

[11] Smith, Grant L., and Lou de la Torre. "Techniques to enable FPGA based reconfigurable fault tolerant space computing." *Aerospace Conference, 2006 IEEE*. IEEE, 2006.

[12] D'Ambrosia, John, Sham Rogers, and Jim Quilici. "‘XAUI—An Overview." *white paper* (2002).

[13] Lewis, Dave. "SerDes Architectures and Application." *Proc. of the DesignCon*. 2004.

[14] Doyle, Richard, et al. "High performance spaceflight computing (HPSC) next-generation space processor (NGSP): a joint investment of NASA and AFRL." *Proceedings of the Workshop on Spacecraft Flight Software*. 2013.

[15] “What is Synthetic Aperture Radar (SAR)?” *Sandia National Laboratories.* Web. 3 Feb. 2018. http://www.sandia.gov/radar/what_is_sar/index.html

[16] Miller, Craig, et al. "Parallel processing of spaceborne imaging radar data." *Proceedings of the 1995 ACM/IEEE conference on Supercomputing*. ACM, 1995.

[17] David, T. S. "SAR Image Formation: ERS SAR processor coded in MATLAB." (2008).

[18] “CubeSat Launch Initiative.” *NASA.* 5 Dec. 2017. Web. 17 Mar. 2018. https://www.nasa.gov/content/about-cubesat-launch-initiative

[19] Frost, Chad, and E. Agasid. "Small spacecraft technology state of the art." *NASA Technical Report TP-2014-216648/REV1*, NASA Ames Research Center (2014).

[20] “November 2017 Top 500 List.” *Top500*. 18 Nov. 2017. Web. 17 Mar. 2018. https://www.top500.org/lists/2017/11/

[21] “Capabilities & Services.” *SpaceX*. Web. 17 Mar. 2018. http://www.spacex.com/about/capabilities

[22] Petersen, Edward. Single-Event effects in aerospace. John Wiley & Sons, 2011.

[23] Maurer, Richard H., et al. "Harsh Environments: Space Radiation." Johns Hopkins APL technical digest 28.1 (2008): 17.

[24] Samwel, S. W., et al. "Studying the Total Ionizing Dose and Displacement Damage Dose effects for various orbital trajectories." *First Middle East-Africa Regional IAU Meeting*. 2008.

[25] Heirtzler, James R., Joe H. Allen, and Daniel C. Wilkinson. "Ever-present South Atlantic Anomaly damages spacecraft." *EOS, Transactions American Geophysical Union* 83.15 (2002): 165-169.

[26] Wilson, Christopher, et al. "Hybrid, adaptive, and reconfigurable fault tolerance." *Aerospace Conference, 2017 IEEE*. IEEE, 2017.

[27] Manasevit, H. M., and W. I. Simpson. "Single-Crystal Silicon on a Sapphire Substrate." *Journal of Applied Physics* 35.4 (1964): 1349-1351.

[28] Oliveira, Roystein, Aditya Jagirdar, and Tapan J. Chakraborty. "A tmr scheme for seu mitigation in scan flip-flops." *Quality Electronic Design*, 2007. ISQED'07. *8th International Symposium* on. IEEE, 2007.

[29] Wilson, J.R. “Radhard moves into the submicron age” *Military & Aerospace Electronic*. 10 Jun. 2014. Web. 15 Mar. 2018. http://www.militaryaerospace.com/articles/print/volume-25/issue-6/technology-focus/radhard-moves-into-the-submicron-age.html

[30] K. Singh, J.P. Walters, J. Hestness, J. Suh, C. Rogers, and S. Crago, “FFTW and complex ambiguity function performance on the maestro processor.” *IEEE Aerospace Conference*, 5-12 March 2011.

[31] J. Marshall, R. Berger, M. Bear, L. Hollinden, J. Robertson, and D. Rickard, “Applying a high performance tiled radhard digital signal processor to spaceborne applications.” *IEEE Aerospace Conference*, 3-10 March 2012.

[32] Y. Gao, F. Zhang, and J. Bakos, “Sparse matrix-vector multiply on the keystone II digital signal processor.” *IEEE high Performance Extreme Computing Conference (HPEC)*. 9-11 September 2014.

[33] M. Bahtat, S. Belkouch, P. Ellaume, and P. Le Gall, “Efficient implementation of a complete multi-beam radar coherent-processing on a telecom soc.” *IEEE International Radar Conference*. 13-17 October 2014.

[34] Frigo, Matteo, and Steven G. Johnson. "FFTW: An adaptive software architecture for the FFT." *Acoustics, Speech and Signal Processing, 1998*. Proceedings of the 1998 IEEE International Conference on. Vol. 3. IEEE, 1998.

[35] Frigo, Matteo, and Steven G. Johnson. "The design and implementation of FFTW3." *Proceedings of the IEEE* 93.2 (2005): 216-231.

[36] Mermer, Coskun, Donglok Kim, and Yongmin Kim. "Efficient 2D-FFT implementation on mediaprocessors." *Parallel Computing* 29.6 (2003): 691-709.

[37] “Multicore dsp+arm keystone ii system-on-chip (soc).” *Texas Instruments*. Web. 15 Apr. 2017. http://www.ti.com/lit/ds/symlink/66ak2h12.pdf

[38] “ODROID-C2.” *HardKernel.* Web. 16 Apr. 2017. http://www.hardkernel.com/main/products/prdt_info.php?g_code=G145457216438&tab_idx=2

[39] “HiKey User Manual” *96Boards.* Web 15 Mar. 2018. https://github.com/96boards/documentation/blob/master/consumer/hikey/hardware-docs/hardware-user-manual.md

[40] “ZCU102 Evaluation Board User Guide.” *Xilinx*. 2. Aug. 2017. Web. 3 Feb. 2018. https://www.xilinx.com/support/documentation/boards_and_kits/zcu102/ug1182-zcu102-eval-bd.pdf

[41] “Zynq UltraScale+ MPSoC.” *Xilinx.* Web. 16 Mar. 2018. https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html

[42] Stone, John E., David Gohara, and Guochun Shi. "OpenCL: A parallel programming standard for heterogeneous computing systems." *Computing in science & engineering* 12.3 (2010): 66-73.

[43] Chapman, Barbara, Gabriele Jost, and Ruud Van Der Pas. *Using OpenMP: portable shared memory parallel programming*. Vol. 10. MIT press, 2008.

[44] J. J. Johnson, "Implementing the cross ambiguity function and generating geometry-specific signals." M.S thesis, Naval Postgraduate School, Monterey, California, 2001.

[45] Schwaller, Benjamin, Barath Ramesh, and Alan D. George. "Investigating TI KeyStone II and quad-core ARM Cortex-A53 architectures for on-board space processing." High Performance Extreme Computing Conference (HPEC), 2017 IEEE. IEEE, 2017.

[46] "Parallel Versions of FFTW." *FFTW*. Web. 16 Feb. 2018. http://www.fftw.org/parallel/parallel-fftw.html

[47] Wang, Dan, Murtaza Ali, and Ellen Blinka. "Synthetic Aperture Radar (SAR) Implementation on a TMS320C6678 Multicore DSP." *Texas Instruments* (2015).

[48] Przytula, K. W., and J. G. Nash. "Implementation of synthetic aperture radar algorithms on a systolic/cellular architecture." *Systolic Arrays*, 1988., *Proceedings of the International Conference on*. IEEE, 1988.