Mälardalen University Press Dissertations
No. 347

SPACE COMPUTING USING COTS HETEROGENEOUS PLATFORMS

INTELLIGENT ON-BOARD DATA PROCESSING IN SPACE SYSTEMS

Nandinbaatar Tsog

2021

School of Innovation, Design and Engineering

ISBN 978-91-7485-528-9
ISSN 1651-4238
Printed by E-Print AB, Stockholm, Sweden

Abstract

Space computing is growing due to the technological advances of high performance commercial off-the-shelf (COTS) computing platforms. Space offers a complex and challenging environment, with size, weight, power, and timing constraints, communication limitations, and radiation effects.

The research presented in this thesis aims at investigating and supporting intelligent on-board data processing using COTS heterogeneous computing platforms in space systems. We investigate platforms with at least one Central Processing Unit (CPU) and one Graphics Processing Unit (GPU) on the same chip. The main goal of the research presented in this thesis is twofold. First, investigate the heterogeneous computing platforms to propose a solution to tackle the above-mentioned challenges in space systems. Second, to complement the proposed solution with novel scheduling techniques for real-time applications that run on COTS heterogeneous platforms in harsh environments like space.

The proposed solutions are based on the system model that considers the use of alternative executions of parallel segments of tasks. Although offloading a parallel segment to a parallel computation unit (such as GPU) improves the best-case execution times of most applications, it can increase the response times of tasks in some applications due to the overuse of GPU. Hence, using the proposed task model can be a key to decreasing the response times of tasks and improving schedulability of the system. The server-based scheduling techniques support the proposed task model by guaranteeing the execution slot for parallel segments on CPU(s). Our experimental evaluation shows that the proposed allocation can increase the number of schedulable task sets of the real-time systems up to 90% compared to the static allocation of applications.

We also present a dynamic allocation method using server-based scheduling with the proposed task model that can improve the schedulability up to 16%. Finally, the thesis presents a simulation tool that supports designers in choosing heterogeneous processing units using the proposed task model while considering the different levels of radiation tolerance to the processing units.

Sammanfattning

Rymddatorn växer på grund av de tekniska framstegen inom högpresterande kommersiella plattformar (COTS). Rymden erbjuder en komplex och utmanande miljö med storlek, vikt, effekt och tidsbegränsningar, kommunikationsbegränsningar och strålningseffekter.

Forskningen som presenteras i denna avhandling syftar till att undersöka och stödja intelligent omborddatabehandling med hjälp av COTS heterogena datorplattformar i rymdsystem. Vi undersöker plattformar med minst en Central Processing Unit (CPU) och en Graphics Processing Unit (GPU) på samma chip. Huvudmålet med forskningen som presenteras i denna avhandling är tvåfaldigt. Undersök först de heterogena dataplattformarna för att föreslå en lösning för att hantera ovan nämnda utmaningar i rymdsystem. För det andra, för att komplettera den föreslagna lösningen med nya schemaläggningstekniker för realtidsapplikationer som körs på COTS heterogena plattformar i tuffa miljöer som rymden.

De föreslagna lösningarna baseras på systemmodellen som överväger användningen av alternativa utföranden av parallella segment av uppgifter. Även om avlastning av ett parallellt segment till en parallell beräkningsenhet (t.ex. GPU) förbättrar de bästa tillämpningstiderna för de flesta applikationer, kan det öka svarstiderna för uppgifter i vissa applikationer på grund av överanvändning av GPU. Därför kan användning av den föreslagna uppgiftsmodellen vara en nyckel för att minska responstiderna för uppgifter och förbättra systemets schemaläggning. De serverbaserade schemaläggningsteknikerna stöder den föreslagna uppgiftsmodellen genom att garantera exekveringsplatsen för parallella segment på CPU(er). Vår experimentella utvärdering visar att den föreslagna fördelningen kan öka antalet schemalagda uppgiftsuppsättningar för realtidssystemen upp till 90% jämfört med den statiska fördelningen av applikationer.

Vi presenterar också en dynamisk allokeringsmetod med hjälp av serverbaserad schemaläggning med den föreslagna uppgiftsmodellen som kan förbättra schemaläggningen upp till 16%. Slutligen presenterar avhandlingen ett simuleringsverktyg som stöder konstruktörer i att välja heterogena bearbetningsenheter med hjälp av den föreslagna uppgiftsmodellen samtidigt som man beaktar de olika strålningstoleransnivåerna för behandlingsenheterna.

To my family.

Where there's a will there's a way

Acknowledgments

First of all, I would like to express my sincere gratitude to my supervisors, my principal supervisor Professor Mikael Sjödin, industrial co-supervisor, Adjunct Professor Fredrik Bruhn, academic co-supervisors Professor Moris Behnam and Associate Professor Saad Mubeen. I am glad that I could travel this journey with you under your expert guidance, persistent support, and enormous encouragement. I am grateful for your valuable ideas, guidance, suggestions, comments, and feedback.

I appreciate the contributors, Dr. Matthias Becker and Dr. Harris Gasparakis, to this thesis. Many thanks to all of my co-authors for collaborating with me. Thank you Jakob Danielsson, Marcus Larsson, Ashalatha Kunnappilly, Dr. Mobyen Uddin Ahmed, Dr. Shahina Begum, Alexandros Binios, Dr. Jaan Praks, and Dr. René Laufer.

Special thanks to Jakob, who is my friend, colleague, bro, contributor, psychologist, room&team-mate, our team-leader, my driver, etc. It is not only a matter of time. "Admire, respect, support, sharpening, help" these words describe you.

I appreciate Dr. Guillermo Rodriguez-Navas, who is my friend and advisor. Thanks for spending many days for discussing, leading, consolidating the ideas. I hope we will work together more and more.

I deeply appreciate my teachers, Sharaa, Munkhjargal, Erdene Natsagdorj, Professor Motomu Takeshige, Professor Shimizu, and Professor Yasushi Kato. Your words supported me to coming back to the academia.

I would like to thank Prof. Sasikumar Punnekkat and Prof. Masoud Daneshtalab for valuable suggestions and comments on the thesis and dissertation.

Special thanks to my friends and advisors, Jakob Danielsson, Marcus Larsson, Tobias Andersson, Uyanga Ganbaatar, Batbuyan Batchuluun, Dr. Guillermo Rodriguez-Navas, Dr. Predrag Filipovikj, Filip Markovic, Ashalatha Kunnappilly, Mirgita Frasheri, Dr. Gabriel Campeanu, Leo Hatvani, Dr. Momo, Dr. Svetlana Girs, Prof. Kristina Lundqvist, Prof. Cristina Seceleanu, Prof.

Masoud Daneshtalab, Prof. Micke, Dr. Sara Abbaspour, Prof. Tiberiu Seceleanu, Dr. Marcus Jägemar, Stefan Karlsson, Aldin Berisa, Zenepe Satka, Tugu, Batya, Mitsuteru Kaneoka, Koji Yamaguchi, and Prof. Ryu Funase. Your words, cheered me up a lot.

```
if ($id =~ /([a-z][a-z][a-z][0-9][0-9])/) {
    print <<MDH;
    To $1,
    Thank you for spending time with me at MDH.
    MDH
}
```

I appreciate the DPAC project for funding my doctoral study. I also appreciate Advanced Micro Devices, Inc. (AMD) and Unibap AB (publ.) for donating and providing the test platforms. In addition, I am thankful to Volvo CE, Saab, and SaraniaSat Inc. for providing the test data. Furthermore, I would like to thank TESO Corporation, MOOCHA co-authors, DPAC members, HERO project team members, and family members.

Finally and foremost, I would like to express my greatest gratitude to my wife Bolormaa, son Ananda, parents, sister Nandin, and Jouni for your continuous love, support, and encouragement.

Nandinbaatar Tsog
Sala, Oct 12, 2021

List of Publications

Papers Included in the Doctoral Thesis[1]

Paper A: *Intelligent Data Processing using In-Orbit Advanced Algorithms on Heterogeneous System Architecture* – Nandinbaatar Tsog, Moris Behnam, Mikael Sjödin, Fredrik Bruhn. In the Proceedings of the 39th International IEEE Aerospace Conference, AeroConf 2018.

Paper B: *A Trade-Off between Computing Power and Energy Consumption of On-Board Data Processing in GPU Accelerated In-Orbit Space Systems* – Nandinbaatar Tsog, Saad Mubeen, Mikael Sjödin, Fredrik Bruhn. In the Transactions of the Japan Society for Aeronautical and Space Sciences, Aerospace Technology Japan, ATJ 2020.

Paper C: *Enabling Radiation Tolerant Heterogeneous GPU-based Onboard Data Processing in Space* – Fredrik C. Bruhn, Nandinbaatar Tsog, Fabian Kunkel, Oskar Flordal, Ian Troxel. In the CEAS Space Journal, CEAS 2020.

Paper D: *Simulation and Analysis of In-Orbit Applications under Radiation Effects on COTS Platforms* – Nandinbaatar Tsog, Saad Mubeen, Mikael Sjödin, Fredrik Bruhn. In the Proceedings of the 42nd International IEEE Aerospace Conference, AeroConf 2021.

Paper E: *Static Allocation of Parallel Tasks to Improve Schedulability in CPU-GPU Heterogeneous Real-Time Systems* – Nandinbaatar Tsog, Matthias Becker, Fredrik Bruhn, Moris Behnam, Mikael Sjödin. In the Proceedings of the 45th Annual Conference of the IEEE Industrial Electronics Society, IECON 2019.

Paper F: *Offloading Accelerator-intensive Workloads in CPU-GPU Heterogeneous Processors* – Nandinbaatar Tsog, Saad Mubeen, Fredrik Bruhn,

[1] The included papers have been reformatted to comply with the doctoral thesis settings.

Moris Behnam, Mikael Sjödin. In the Proceedings of the 26th International Conference on Emerging Technologies and Factory Automation, ETFA 2021.

Additional Peer-Reviewed Publications, not Included in the Doctoral Thesis

1. *A Trade-Off between Computing Power and Energy Consumption of On-Board Data Processing in GPU Accelerated Real-Time Systems* – Nandinbaatar Tsog, Mikael Sjödin, Fredrik Bruhn. In the Proceedings of the 32nd International Symposium on Space Technology and Science, ISTS 2019.

2. *Using Docker in Process Level Isolation for Heterogeneous Computing on GPU Accelerated On-Board Data Processing Systems* – Nandinbaatar Tsog, Mikael Sjödin, Fredrik Bruhn. In the Proceedings of the 12th IAA Symposium on Small Satellites for Earth Observation, IAASmallSat 2019.

3. *Moon Cubesat Hazard Assessment (MOOCHA) - An International Earth-Moon Small Satellite Constellation* - Alexandros Binios, Janis Dalbins, Sean Haslam, Rusnė Ivaškevičiūtė, Ayush Jain, Maarit Kinnari, Joosep Kivastik, Fiona Leverone, Juuso Mikkola, Ervin Oro, Laura Ruusmann, Janis Sate, Hector-Andreas Stavrakakis, Nandinbaatar Tsog, Karin Pai, Jaan Praks, René Laufer. In the Proceedings of the 12th IAA Symposium on Small Satellites for Earth Observation, IAASmallSat 2019.

4. *Using Heterogeneous Computing on GPU Accelerated Systems to Advance On-Board Data Processing* - Nandinbaatar Tsog, Mikael Sjödin, Fredrik Bruhn. In the European Workshop on On-Board Data Processing, OBDP 2019.

5. *A Systematic Mapping Study on Real-time Cloud Services* - Jakob Danielsson, Nandinbaatar Tsog, Ashalatha Kunnappilly. In the Proceedings of the 1st Workshop on Quality Assurance in the Context of Cloud Computing, QA3C 2018.

6. *Advancing On-Board Big Data Processing Using Heterogeneous System Architecture* - Nandinbaatar Tsog, Mikael Sjödin, Fredrik Bruhn. In the Proceedings of the ESA/CNES 4S Symposium 2018, 4S 2018.

7. *Real-Time Capabilities of HSA Compliant COTS Platforms* - Nandinbaatar Tsog, Matthias Becker, Marcus Larsson, Fredrik Bruhn, Moris Behnam, Mikael Sjödin. In the Proceedings of the 37th IEEE Real-Time Systems Symposium (WiP) , WiP RTSS 2016.

Contents

Part I

Thesis

Chapter 1

Introduction

In the last two decades, the exploitation of small satellites and CubeSats[1] has rapidly increased for academic, commercial, and government intelligence applications [1, 2]. Space applications like for earth observation, deep space explorations, and communications, exploit on-board processing and reconstitute the role and use of small satellites from being a simple node of sensing data to a generator of big data and provider of efficient storage, and intelligent on-board decision making using the data, i.e., space computing. Numerous studies show that the intelligent on-board processing improves the use of the limited communication link bandwidth on small satellites [3, 4, 5], while it requires high-performance computing capability under the size, weight, and power (SWaP), radiation and real-time constraints.

As high-performance space computing technology is a key for the future of space missions, the interest in commercial-off-the shelf (COTS) heterogeneous platforms for space computing is growing vastly. Heterogeneous platforms employ different types of processing units on the same chip such as Central Processing Unit (CPU), Graphics Processing Unit (GPU), Field-Programmable Gate Array (FPGA), Digital Signal Processor (DSP), to mention a few. These platforms provide massive computation capabilities. However, these platforms make the systems more complex. For instance, the systems that utilize heterogeneous computing platforms are more prone to unpredictable timing behaviours compared to the systems that use single-core computing platforms [6]. The work presented in this thesis tackles the challenges to make efficient use of the different compute resources considering various bottlenecks and difficulties in heterogeneous computing platforms.

Furthermore, COTS platforms are usually wobbly against radiation effects

[1] https://www.nasa.gov/content/what-are-smallsats-and-cubesats

without handling any radiation hardening. Besides, understanding of how COTS platforms react under radiation effects is less studied on the level of software and applications. Therefore, this thesis deals with synthetic simulations of radiation effects in order to assess timing predictability of real-time systems using COTS heterogeneous platforms.

As discussed earlier, heterogeneous platforms include multiple processing units and each of them can consist of multiple cores. Thus, this thesis provides a technique to support balanced use of heterogeneous processing units, while improving timing predictability of these platforms. Furthermore, we focused on an investigation of the trade-off between computing performance and power consumption while supporting schedulability of task sets. This investigation shows that our proposed techniques fit well with the embedded systems that are constrained by real-time and energy constraints.

1.1 Thesis Goal and Research Challenges

The overall goal of the work in this thesis is

> "to improve the timing predictability of real-time applications on heterogeneous computing platforms under harsh environments for on-board data processing (e.g., space computing) without degrading their computing performance and energy efficiency."

To achieve the goal, we target three core research challenges as follows:

> **Research Challenge 1:** To identify the key factors that affect timing predictability and power consumption in real-time applications on heterogeneous CPU-GPU platforms without degrading their computing performance.

> **Research Challenge 2:** To improve schedulability of real-time applications running on heterogeneous computing platforms by manipulating the identified factors.

> **Research Challenge 3:** To provide techniques and tools to schedule, analyze and simulate real-time applications running on heterogeneous computing platforms, in particular, under harsh environments.

1.2 Outline of the Thesis

The thesis is based on a collection of six peer-reviewed publications. The thesis consists of two parts:
Part I includes the first five chapters. In Chapter 1, an introduction to the thesis has been provided. The thesis goal and research challenges are introduced in this chapter. Chapter 2 presents the preliminary concepts that are considered throughout the thesis. In Chapter 3, we discuss the contributions and research methodology in the thesis. We present the related work in Chapter 4, and the conclusion and the future work are presented in Chapter 5.
Part II provides the collection of six peer-reviewed publications considered as the scientific contributions of the thesis. This part constitutes Chapters 6-11.

Chapter 2

Background

This chapter provides the required background information following the motivation of the thesis that starts with space computing, and continues with necessary hardware/software solution (heterogeneous computing) with required real-time constraint.

2.1 Space Computing

Space computing started to gain considerable attention of the research computing with the advent of the IEEE Space Computing Conference (SCC) [1] since 2006. Previously, this event was know as Fault Tolerant Space Computing conference/workshop. As space computing performs under harsh environments with radiation, it is closely related to the research introduced in IEEE Nuclear & Space Radiation Effects Conference (NSREC) [2]. Based on active topics covered by the SCC and NSREC conferences, we illustrate space computing in Figure 2.1. Among these topics, this thesis concerns more about COTS components, on-board data processing, radiation, heterogeneous space processors using machine learning benchmarks for intelligent decision making. In this section, we provide more information about on-board data processing and radiation.

2.1.1 On-Board Data Processing

Space is a perfect example for considering real-time system applications as many on-board functions in spacecraft and satellites are constrained by soft

[1] https://spacecomputing.ecs.baylor.edu/

[2] http://www.nsrec.com/

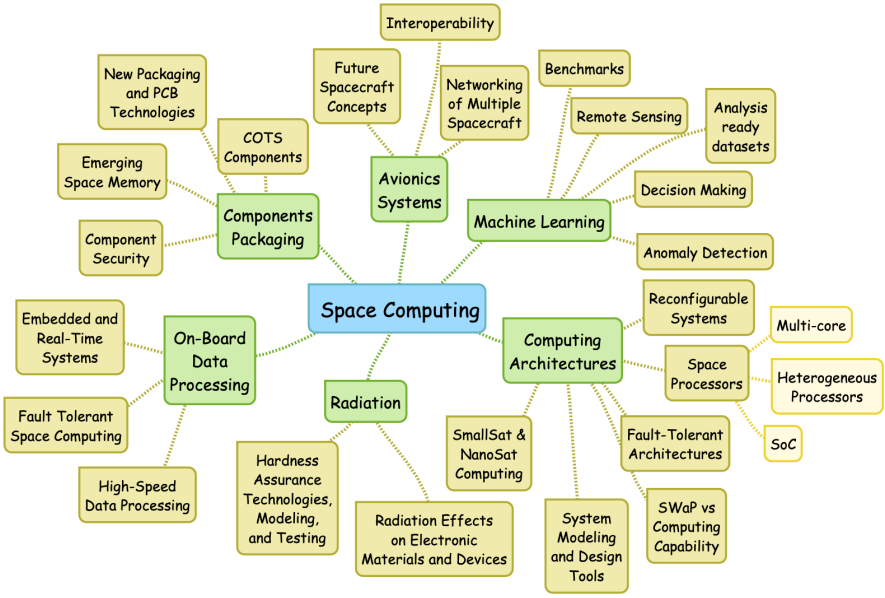


Figure 2.1: Space computing

and hard real-time requirements (see Section 2.3.2 for more details about soft and hard real-time systems). Any failure such as losing control of a system or unable to transfer data may end up in a catastrophic result as it is not possible to fix the devices in orbit or deep space. In order to reduce delays and have more predictable activities, the role of on-board data processing becomes significant. However, due to SWaP (size, weight and power) constraints along with radiation hardiness problem in space, the development of space systems usually encounter limitations which are not always experienced on the earth. The design and development of the on-board computer need to overcome these limitations.

As a heterogeneous architecture, the combination of CPU + FPGA and/or CPU + DSP is broadly employed for on-board computers in space [7]. However, these combinations could not support massive amount of computations required by the intelligent on-board data processing systems. These heterogeneous architecture combinations complemented by GPUs can overcome the above mentioned limitation by running multiple parallel executions and faster memory accesses. Thus, heterogeneous architectures that include CPU, GPU and FPGA can offer an efficient computation solution for on-board data processing systems. In such an architecture, an FPGA can be used for receiving sensors' data with shorter delays, a CPU can act as a controller between the

FPGA and GPU, and the GPU can process heavy computations.

2.1.2 Radiation Effects

Radiation effects increase the complexity of space systems. As radiation effects are cumulative on the one hand, although the dose of space radiation is mostly low, its risk increases by the total time traveled in space [8, 9]. This characteristic is described by total dose of radiation, i.e., total ionizing dose (TID). Developing shielding materials or radiation-hardened products in order to mitigate radiation effects in orbit components could worsen the other limitations such as size, weight, and power, cost, and development time. On the other hand, particles with high energy such as electrons cause electrostatic discharge, single-event effects (SEEs). Thus, radiation effects can hinder the usage of commercial off-the-shelf (COTS) technologies that have been successful in the systems used the earth, such as COTS system on chip (SoC), including the use of integrated graphics processing units (GPUs), which improve the quality of onboard data processing [10]. We introduce safety allowable range of both particle energy and TID as radiation tolerance in this thesis and consider that an idea of radiation tolerance improves space computing.

2.2 Heterogeneous Computing

The role of heterogeneous computing has been growing dramatically in industrial applications [11]. Employing multiple types of processing units makes the embedded systems robust. Freund and Conwell define heterogeneous computing as follows:

Definition 2.1: *"Heterogeneous computing is the well-orchestrated and coordinated effective use of a suite of diverse high-performance machines (including parallel machines) to provide super speed processing for computationally demanding tasks with diverse computing needs."*
R. F. Freund and D. S. Conwell [12]

We re-define the above definition using a new term "heterogeneous processor" instead of machines and consider it in this thesis. *"Heterogeneous computing is the well-orchestrated and coordinated effective use of a suite of on- and/or off-chip heterogeneous processing units to provide high-performance processing for computationally demanding tasks with diverse computing needs."* With this definition, the focus of computation shifts from a machine to a processing unit (PU), and the location of them distinguished by either on- or off-chip.

In case of off-chip, processing units communicate with each other via diverse types of interconnection and network, while processing units employed in the same die for on-chip. The thesis mostly touches on-chip heterogeneous processing units as heterogeneous processing units.

Heterogeneous processing units can employ the same type of naming with different instruction set architecture and/or even different clock speed (e.g., ARM Cortex CPU + Nvidia Denver CPU), while commonly considered to employ the combinations of different types of naming processors such as CPU + GPU, CPU + FPGA and etc. In order to study heterogeneous computing, not only processor types, but also heterogeneous architectures, memory management and interconnection are crucial.

2.2.1 Heterogeneous Architectures

As defined in Section 2.2, heterogeneous computing appears with different architectures, as described heterogeneous architectures. Figures 2.2, 2.3, 2.4, and 2.5 illustrate the well-known heterogeneous architectures as some of them appear in satellites and autonomous vehicles.

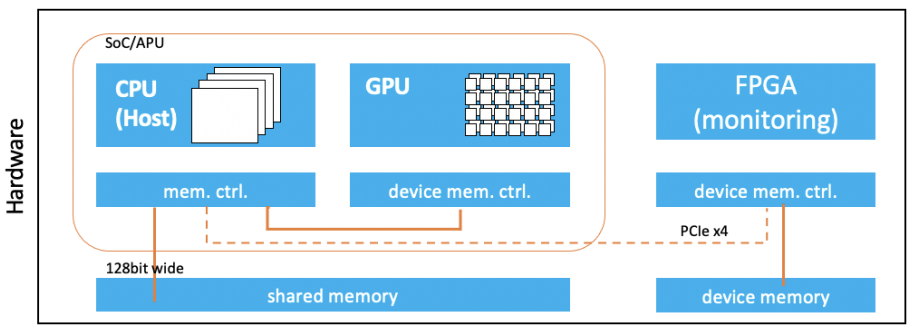


Figure 2.2: GIMME3 and GIMME4 platforms by Unibap AB and Mälardalen University based on AMD APU (CPU and integrated GPU).

The heterogeneous architecture described in Figure 2.2 consists of three different processing units, CPU, GPU, and FPGA. CPU and GPU are in a system on a chip (SoC) and connect to shared memory. The purpose of external processing unit, FPGA, is to monitor the health of SoC. FPGA interconnects with CPU via PCIexpress and has its own memory. The SoC illustrated in Figure 2.2 is called Accelerated Processing Units (APU), that is commonly known as integrated GPU. In this platform, APU considers a computer architecture specialized for heterogeneous computing, i.e., Heterogeneous System Architecture (HSA).

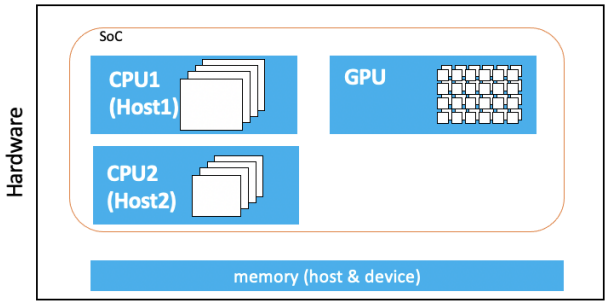


Figure 2.3: Jetson TX platform employs ARM big.LITTLE CPUs (ARM Cortex CPUs and Nvidia Denver CPUs) with Nvidia GPU.

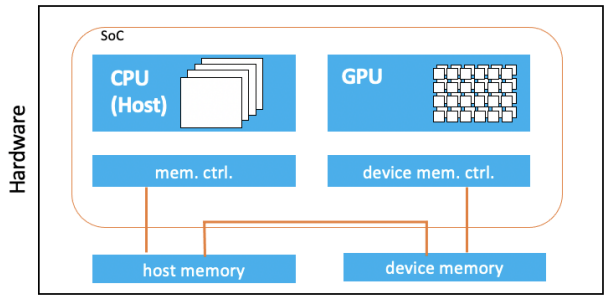


Figure 2.4: GPU4S by Barcelona Supercomputing Center based on Jetson Xavier platform (ARM CPU and Nvidia GPU).

Both heterogeneous architectures illustrated in Figures 2.3 and 2.4 contain only SoCs. Figure 2.4 describes that homogeneous CPU cores and GPU are employed in the same SoC and connect to independent memories. On the other hand, the SoC illustrated in Figure 2.3 consists of heterogeneous CPU cores and GPU. In other words, two CPUs from different vendors and different frequencies are employed in the SoC. This architecture is known as the big.LITTLE CPUs architecture. In order to improve heterogeneous computing, the Big.LITTLE CPUs can be used with clustered switching, in-kernel switching, and heterogeneous multi-processing task migration and scheduling mechanisms [13].

The heterogeneous architecture illustrated in Figure 2.5 consists of two SoCs, which are connected through Ethernet. One of them includes two CPUs with different computing capacities and GPU. Another SoC comprises CPU and FPGA.

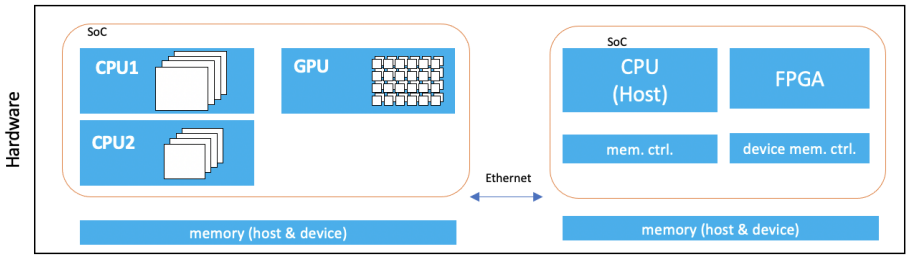


Figure 2.5: On-board platform for the University of Georgia's satellite (Jetson TX2i + Smart Fusion).

2.2.2 Memory Model & Interconnection

Memory Model

In Figures 2.6, 2.7, and 2.8, most common memory models for heterogeneous architectures are described. In these models, we do not focus on the interconnection between PUs and between PUs and memories, while the relation between memories of different PUs are crucial. As illustrated in Figure 2.6, each PU has own allocated memory and independent memory address, i.e., it is called Multi Memory Model (MMM). In this case, data should be transferred between different memories and addresses should be handled as well. Heterogeneous architectures with external PUs tend to have this memory model such as the platform depicted in Figure 2.5. In unified (virtual) memory model (UVM) as illustrated in Figure 2.7, PUs connect to a physical memory. However, each PU should be connected to allocated area only. This means, data should still be transferred/copied between memory areas when different PUs need to access the same data. UVM uses a memory address system for all the PUs and data will be copied in the same physical memory. Hence, data transfer time is less compared to UVM.

As depicted in Figure 2.8, Unified Shared Memory (USM) allows to access to the same data from different PUs. This eliminates unnecessary data transfer and copy. However, it requires advanced interconnection techniques.

Interconnection

Interconnection is the main key to the challenges of heterogeneous memory management, although both software and hardware solutions are required. There exist the following four specifications focused on the interconnection

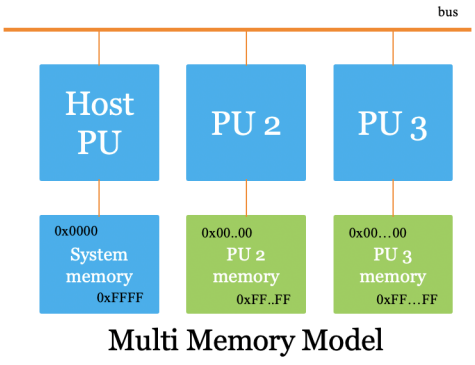


Figure 2.6: Multi Memory Model.

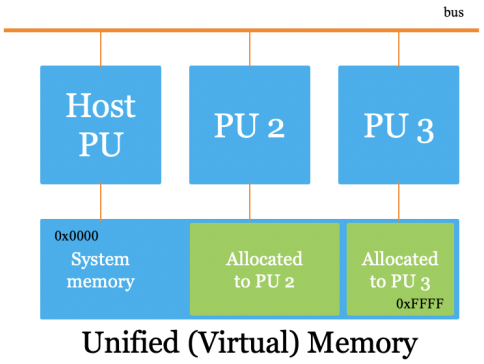


Figure 2.7: Unified (Virtual) Memory.

and buses. CCIX[3] (Cache Coherent Interconnect for Accelerators), OpenCAPI[4] (Open Coherent Accelerator Processor Interface), Gen-Z[5] and CXL[6] (Computer Express Link). Only CXL is proposed by an Intel-driving consortium, and both AMD and ARM are announced to join this consortium. In this sense, CXL has a vast potential that could be upgraded to the industry de-facto standard for the interconnection between host and devices.

[3]CCIX https://www.ccixconsortium.com/
[4]OpenCAPI https://opencapi.org/
[5]Gen-Z https://genzconsortium.org/
[6]CXL https://www.computeexpresslink.org/

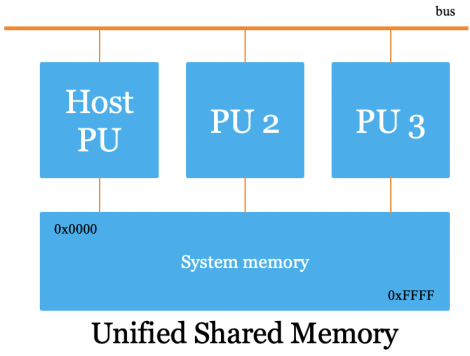


Figure 2.8: Unified Shared Memory.

2.2.3 Heterogeneous System Architecture (HSA)

Different types of specifications and designs of the processing units bring complexities for the development process from cost and timing perspective. To tackle these problems, HSA Foundation [14] has been established by multiple leading hardware vendors to develop the Heterogeneous System Architecture (HSA) specification for reducing the complexity of heterogeneous computations and providing the developer-friendly environments. The main properties of HSA are described with memory handling, software stack, and interconnection.

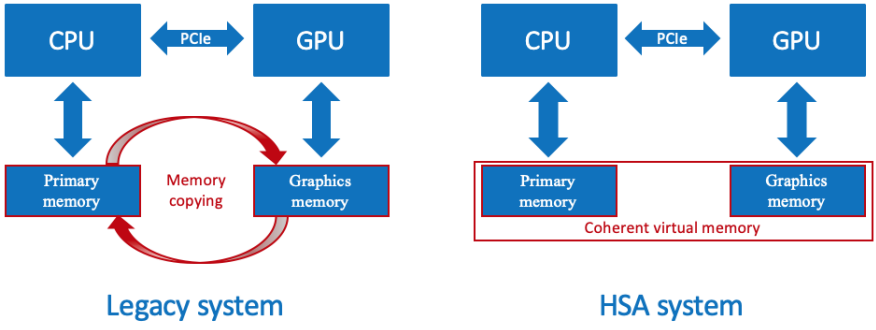


Figure 2.9: Memory structure between a non-HSA system and an HSA system.

Memory handling: The HSA aims to ease the development process on the heterogeneous platform by providing a development environment to the programmers that is similar to the environment for traditional systems, i.e., homogeneous systems. The HSA provides unified coherent memory for host

and devices that saves time for transferring data between different physical memories, i.e., there is no memory copy between different physical memories, e.g., primary and graphics memories (see Figure 2.9).

Software stack: As a part of the HSA, AMD introduces an initiative GPUOpen, an open source software stack, including, but not limited to, kernel level driver, runtime environment, profiling tools, computer vision and machine learning libraries such as ROCm[7], CodeXL[8], AMD OpenVX[9], MIOpen[10] as well as Tensorflow[11] on AMD GPUs. From TensorFlow 2.0, AMD has fully upstreamed their support.

Interconnection: While HSA covers interconnection between different devices, it needs to be collaborated with the interconnection specifications. As CXL is an open standard interconnection and only accepted by all the main vendors such as Intel, AMD, and ARM, the relation between HSA and CXL is crucial and it could co-exist as follows. While HSA is located at a high level, which is close to the developers for reducing the complexity of the development of heterogeneous systems, CXL is directly focused on hardware devices, which is at a low level. This indicates the possibility of the co-existence of HSA and CXL.

2.3 Real-Time Embedded Systems

2.3.1 Embedded Systems

Embedded systems are found in almost all electronic products covering diverse domains such as consumer products, business, military, aerospace and so on. As they are embedded, embedded systems are mostly remain hidden from the end users. A definition of an embedded system is described as follows:

Definition 2.2: *"An embedded system is a combination of computer hardware and software – and perhaps additional parts, either mechanical or electronic – designed to perform a dedicated function."*
M. Barr and A. Massa [15]

[7] ROCm https://github.com/RadeonOpenCompute/ROCm
[8] CodeXL https://gpuopen.com/compute-product/codexl/
[9] OpenVX https://gpuopen.com/compute-product/amd-openvx/
[10] MIOpen https://gpuopen.com/compute-product/miopen/
[11] Tensorflow https://rocm.github.io/tensorflow.html

2.3.2 Real-Time Systems

A real-time system is a system that reacts to external events in a timely manner. This means that not only the accuracy of the result, but also the timeliness is a crucial factor for the accuracy of the system. Hence, a real-time embedded system is an embedded system, which reacts to its environment in a timely manner.

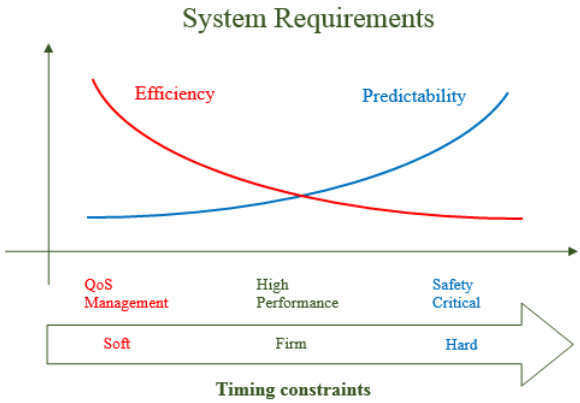


Figure 2.10: A real-time system requirements.[12]

As illustrated in Figure 2.10, real-time systems can be divided into a hard, firm and soft [16] real-time system from perspective of the timing constraints. The hard real-time system must pass all specified timing constraints. If the system misses a constraint (e.g., a deadline) once, it results in failure leading to a fatality and/or big financial or environmental damage. Therefore, many hard real-time systems are considered to be safety critical. In a soft real-time system, one or more deadline misses may be tolerated at the cost of lower quality of service. A firm real-time system is between hard and soft real-time systems.

2.4 System Model and Architecture

We consider a system, which consists of applications comprising of a task set (i.e., a set of applications, software stack), an operating system (kernel including drivers), and a hardware platform as illustrated in Figure 2.11. In

[12] http://www.artist-embedded.org/docs/Events/2008/RT-Kernels/SLIDES/s1-Intro.pdf

addition, in the system architecture, we consider an error to hardware platform, which appears as an interference task to the task set. In this thesis, an error is a radiation effect that is generated from the surrounding harsh environment and interferes with the hardware platform including its devices such as processing units, memory and so on.

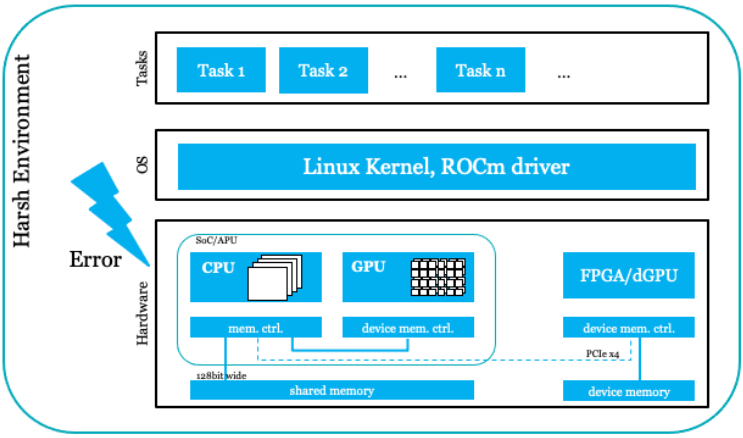


Figure 2.11: System Architecture.

A task in the task set consists of parallel and sequential segments and is represented by the fork/join task model [17]. We assume that sequential segments should be executed only on CPU while parallel segments can be executed in parallel on GPU or on CPU sequentially. Moreover, in this thesis, we consider an extension of the fork-join task model by adopting the notion of heterogeneous segments. A heterogeneous segment can either be mapped to a GPU for parallel execution (alternative B) or to a CPU for sequential execution of the same code segment (alternative A), see Figure 2.12.

The thesis considers that the hardware platform employs a heterogeneous architecture, which may include three types of processing units; host device as CPU, integrated accelerators such as integrated GPU (iGPU), and discrete accelerators such as discrete GPU (dGPU) and/or FPGA (see Figure 2.11). We assume that the hardware platform is HSA-compliant. In some cases, to narrow the setting, at least processing units in system-on-chip (SoC) side should be compliant with the HSA. From the power utilization perspective, on one hand, host device and integrated accelerators share the same power controller, i.e., both turn on and off at the same. On the other hand, discrete accelerators have a dedicated power controller for each, which means that the combinations, ”Host-and-Integrated-Accelerators” and ”Host-and-Discrete-Accelerators”, have

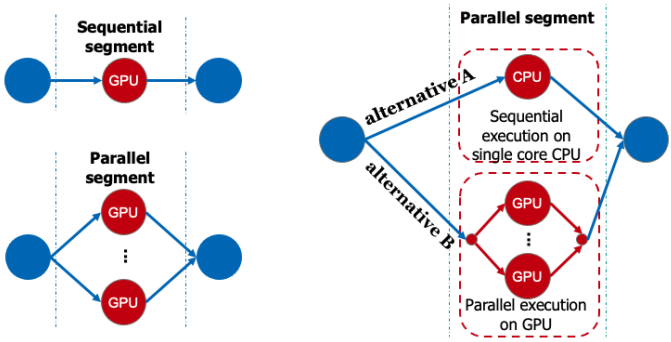


Figure 2.12: Sequential, parallel and alternative executions of parallel segments of tasks.

different amount of power consumption. Furthermore, from the memory structure perspective, "Host-and-Integrated-Accelerators" connects to the shared memory while "Host-and-Discrete-Accelerators" accesses to the virtual shared memory.

In this thesis, the following three reference platforms are considered: CPU + iGPU, CPU + iGPU + FPGA, and CPU + dGPU. Although it is available to use FPGA, our focus on the thesis is only CPU and GPU. CPU scheduling is realized by a partitioned fixed-priority scheduler and we further assume that the execution on CPUs is preemptable. In contrast, the execution on GPU is not preemptable. GPU allows to execute tasks with non-preemptive fixed priority scheduling.

2.5 Metrics

In this thesis, the following metrics are considered in the investigation and experimental evaluation of heterogeneous computing architectures: computing performance, energy efficiency, timing predictability, and radiation tolerance. Use of these metrics in different research methods is explained in Section 3.3.

Computing performance describes how fast tasks are calculated on the given processing units. Hence, this metric presents computing potential of platforms/processing units. A unit of time, second (s), is used for this metric and less computation time shows faster computing performance. There are other ways to express this metric such as FLOPS (floating-point operations per second). In the contributed papers, these expressions are used to give information about the reference platforms at a glance.

Energy efficiency introduces how less power is consumed by the given

processing units to compute tasks. A unit of energy, joule (J), is chosen to describe this metric. A smaller value of power consumption corresponds to more energy efficiency. Energy efficiency is a metric to consider power consumption on the systems/platforms.

Timing predictability presents how systems fulfill the given timing constraints. We consider a system is predictable if all tasks in this system meet their timing requirements [18]. Timing predictability of a system is related to proving, demonstrating or verifying the fulfillment of the timing requirements (e.g., deadline miss) that are specified on the system [19]. In order to consider the timing requirements of tasks in task sets, we conduct schedulability analysis of task sets.

Radiation effect and tolerance express how much high energy particles are present in the environment and what is tolerance of the given components in the systems against the radiation effects. A unit of both radiation effect and tolerance can be either [eV] or [rad] in the cases based on particle energy or total ionizing dose, respectively.

Chapter 3

Research Description

3.1 Scientific Contributions

To achieve the thesis goal, the thesis provides five scientific contributions (SCs), which address the research challenges presented in Section 1.1. These contributions are encapsulated in six peer-reviewed research publications (Papers A-F). Mapping of the research challenges, contributions and publications is shown in Figure 3.1.

Scientific Contribution 1: Investigation of the fundamental characteristics of different types of heterogeneous architectures focusing on computation-time and power-consumption efficiency.
Motivation and summary of the contribution: This contribution helps to understand the current characteristics of heterogeneous platforms. By using the computing performance and energy efficiency metrics, we characterize various computing units in heterogeneous computing platforms. According to our prior knowledge, some tasks are suitable for parallel computing while some tasks are executable only in a sequential manner. Computer vision and machine learning applications are representations of parallelizable applications, and we consider these type of applications in Papers A and C. Our investigations aim to understand what kind of applications (i.e., under what kind of conditions, applications) are suitable to run on GPU compared to CPU or vice versa. As a result, we experienced that the execution of parallel applications on GPU boosts up to 238 times computing performance and consumes 13.5 times less energy, compared to CPU. Although applications that include the smaller numbers of parallel executions are suitable to run on CPU from the computing performance aspect, all parallel applications consumed less energy on GPU compared to CPU in our reference platforms. This contribution ad-

dresses Research Challenge 1.

Scientific Contribution 2: Proposing and evaluating a task model and allocation algorithms based on the model for real-time applications that run on CPU-GPU heterogeneous computing platforms.
Motivation and summary of the contribution: Based on the results acquired by addressing SC 1 and literature review, we extract and propose a novel task model for applications that run on heterogeneous processors, which allows alternative executions of parallel segments of tasks on heterogeneous processing units. By allowing alternative executions, parallel segments can be allocated to an appropriate processing unit with higher computation performance and less power consumption. Alternative executions of parallel segments are investigated in Papers B and E. While Paper B presents the idea of alternative executions of parallel segments of tasks, Paper E considers effects of the alternative executions on the timing behaviours of the applications. As an achievement, the appropriate use of heterogeneous processing units for parallel segments can decrease the total energy consumption of systems up to 64.3%. This contribution addresses Research Challenges 1 and 2.

Scientific Contribution 3: Proposing and evaluating a new technique that utilizes the task model to improve schedulability of real-time applications running on COTS heterogeneous computing platforms.
Motivation and summary of the contribution: Based on the achievements of SC 2 and a literature review, we propose solutions which aim at improving schedulability of the task sets on heterogeneous computing platforms. We integrate these solutions to the timing analysis proposed for CPU-GPU heterogeneous platforms. The solutions increase the number of schedulable task sets up to 90% compared to the existing solutions, while mitigating accelerator intensive loads. The contribution covers both static and dynamic scheduling of task sets. This contribution is discussed in Papers E and F in detail and addresses Research Challenge 2.

Scientific Contribution 4: Proposing a server-based scheduling technique that utilizes the proposed task model to improve the schedulability of real-time applications on CPU-GPU heterogeneous platforms.
Motivation and summary of the contribution: This contribution utilizes the task model and proposes a server-based scheduling technique for real-time systems that run on CPU-GPU heterogeneous computing platforms. Papers E and F cover this contribution. Paper E focuses on static allocation of tasks, and Paper F considers dynamic allocation of tasks. The evaluation results indicate

that one of the proposed techniques based on dynamic scheduling can schedule up to 16% more task sets compared to the traditional non-offloading technique. This contribution addresses Research Challenges 2 and 3.

Scientific Contribution 5: Developing a simulation framework and a tool considering the effects of different radiation tolerance of heterogeneous processing units on the schedulability of real-time applications under harsh environments.

Motivation and summary of the contribution: As radiation effects limits the usability of most of the state-of-the-art COTS technologies from use in space, this contribution presents a simulation framework and a tool to simulate real-time applications under harsh environments using heterogeneous processing units with different radiation tolerances. The simulation tool supports processing units with different characteristics of computing potential, radiation tolerance, as well as different type of processing units such as CPU and GPU. Artificial patterns of single-event effects can be introduced, for example, the patterns generated by using Poisson distribution, and other well-known mathematical distributions. In addition, the simulation tool employs the task model proposed in SC2 and helps to extend the study of heterogeneous processing units under the environments with radiation effects. This contribution is discussed in Paper D and addresses Research Challenge 3 together with SC 4.

As illustrated in Figure 3.1, Paper E proposes a solution based on Papers A and B. Papers A, B, and C focus on practical experiments on real physical platforms, while Papers E and F propose solutions for heterogeneous computing. Paper F is extended work to Paper E. Paper D is based on Papers E and C. It is about a simulation tool to support designers in choosing necessary heterogeneous processing units.

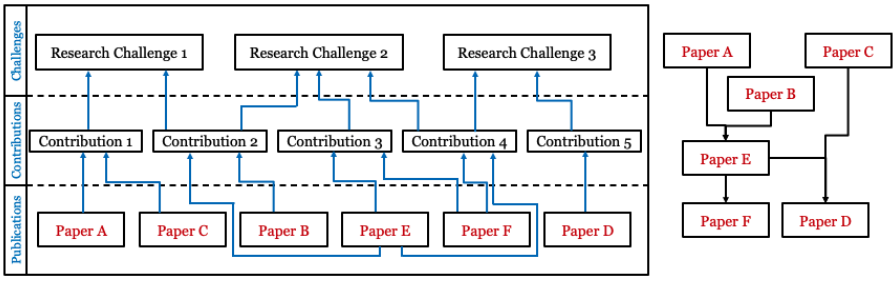


Figure 3.1: Mapping of the research challenges, contributions and publications

3.2 Summary of Included Papers

Paper A: *Intelligent Data Processing using In-Orbit Advanced Algorithms on Heterogeneous System Architecture*
Nandinbaatar Tsog, Moris Behnam, Mikael Sjödin, Fredrik Bruhn.
Published in the Proceedings of the 39th International IEEE Aerospace Conference, March 2018.

Abstract: In recent years, commercial exploitation of small satellites and CubeSats has rapidly increased. Time to market of processed customer data products is becoming an important differentiator between solution providers and satellite constellation operators. Timely and accurate data dissemination is the key to success in the commercial usage of small satellite constellations which is ultimately dependent on a high degree of autonomous fleet management and automated decision support. The traditional way for disseminating data is limited by on the communication capability of the satellite and the ground terminal availability. Even though cloud computing solutions on the ground offer high analytical performance, getting the data from the space infrastructure to the ground servers poses a bottleneck of data analysis and distribution. On the other hand, adopting advanced and intelligent algorithms onboard offers the ability of autonomy, tasking of operations, and fast customer generation of low latency conclusions, or even real-time communication with assets on the ground or other sensors in a multi-sensor configuration. In this paper, the advantages of intelligent onboard processing using advanced algorithms for Heterogeneous System Architecture (HSA) compliant onboard data processing systems are explored. The onboard data processing architecture is designed to handle a large amount of high-speed streaming data and provides hardware redundancy to be qualified for the space mission application domain. We conduct an experimental study to evaluate the performance analysis by using image recognition algorithms based on an open source intelligent machine library 'MIOpen' and an open standard 'OpenVX'. OpenVX is a cross-platform computer vision library.

Paper B: *A Trade-Off between Computing Power and Energy Consumption of On-Board Data Processing in GPU Accelerated In-Orbit Space Systems*
Nandinbaatar Tsog, Saad Mubeen, Mikael Sjödin, Fredrik Bruhn.
Published in the Transactions of JSASS, Aerospace Technology Japan, September 2021.

Abstract: On-board data processing is one of the prior on-orbit activities that

improves the performance capability of in-orbit space systems such as deep-space exploration, earth and atmospheric observation satellites, and CubeSat constellations. However, on-board data processing encounters higher energy consumption compared to traditional on-board space systems. This is because the traditional space systems employ simple processing units such as single-core microprocessors as the systems do not require heavy data processing. Moreover, solving the radiation hardness problem is crucial in space, and adopting a new processing unit is challenging.

In this paper, we consider a Graphics Processing Unit (GPU) accelerated in-orbit space system for on-board data processing. According to prior works, there exist radiation-tolerant GPU, and the computing capability of systems is improved by using heterogeneous computing method. We conduct experimental observations of energy consumption and computing potential using this heterogeneous computing method in our GPU accelerated in-orbit space systems. The results show that the proper use of GPU increases computing potential with 10-140 times and consumes between 8-130 times less energy. Furthermore, the entire task system consumes 10-65% of less energy compared to the traditional use of processing units.

Paper C: *Enabling Radiation Tolerant Heterogeneous GPU-based Onboard Data Processing in Space*
Fredrik C. Bruhn, Nandinbaatar Tsog, Fabian Kunkel, Oskar Flordal, Ian Troxel.
Published in the CEAS Space Journal, June 2020.

Abstract: The last decade has seen a dramatic increase in small satellite missions for commercial, public, and government intelligence applications. Given the rapid commercialization of constellation-driven services in Earth Observation, situational domain awareness, communications including machine-to-machine interface, exploration etc., small satellites represent an enabling technology for a large growth market generating truly Big Data. Examples of modern sensors that can generate very large amounts of data are optical sensing, hyperspectral, Synthetic Aperture Radar (SAR), and Infrared imaging. Traditional handling and downloading of Big Data from space requires a large onboard mass storage and high bandwidth downlink with a trend towards optical links. Many missions and applications can benefit significantly from onboard cloud computing similarly to Earth-based cloud services. Hence, enabling space systems to provide near real-time data and enable low latency distribution of critical and time sensitive information to users. In addition, the downlink capability can be more effectively utilized by applying more onboard processing to reduce the data and create high value information prod-

ucts. This paper discusses current implementations and roadmap for leveraging high performance computing tools and methods on small satellites with radiation tolerant hardware. This includes runtime analysis with benchmarks of convolutional neural networks and matrix multiplications using industry standard tools (e.g., TensorFlow and PlaidML). In addition, a $1/2$ CubeSat volume unit (0.5U) (10×10×5 cm3) cloud computing solution, called SpaceCloud™ iX5100 based on AMD 28 nm APU technology is presented as an example of heterogeneous computer solution. An evaluation of the AMD 14 nm Ryzen APU is presented as a candidate for future advanced onboard processing for space vehicles.

Paper D: *Simulation and Analysis of In-Orbit Applications under Radiation Effects on COTS Platforms*
Nandinbaatar Tsog, Saad Mubeen, Moris Behnam, Mikael Sjödin, Fredrik Bruhn.
Published in the Proceedings of the 42nd International IEEE Aerospace Conference, March 2021.

Abstract: Radiation effects research is crucial as it defines risk to both human bodies and spacecraft. Employing radiation-hardened products is one way to mitigate radiation effects on in-orbit systems. However, radiation effects prohibit most of the state-of-the-art commercial off-the-shelf (COTS) technologies from use in space. Furthermore, radiation effects on software components are less studied compared to hardware components. In this work, we introduce a simulation tool that simulates and performs post-simulation analysis of the impact of radiation effects on schedulability of the software task sets that execute on COTS system-on-chip (SoC) platforms within in-orbit systems. In order to provide a meaningful verification environment, single-event effects (SEEs) are introduced as aleatory disturbances characterized by probability distribution of occurrence using their predefined models. The tool supports interoperability with several other tools as it uses the extensible markup language (XML) model files for input and output, i.e., for importing input task sets and radiation effects and exporting the simulation and analysis results. The proposed tool is extensively by running simulations using a use case of an in-orbit onboard monitoring system.

Paper E: *Static Allocation of Parallel Tasks to Improve Schedulability in GPU Accelerated Real-Time Systems*
Nandinbaatar Tsog, Matthias Becker, Fredrik Bruhn, Moris Behnam, Mikael Sjödin.

Published in the Proceedings of the 45th Annual Conference of the IEEE Industrial Electronics Society (IECON), October 2019.

Abstract: Autonomous driving is one of the main challenges of modern cars. Computer visions and intelligent on-board decision making are crucial in autonomous driving and require heterogeneous processors with high computing capability under low power consumption constraints. The progress of parallel computing using heterogeneous processing units is further supported by software frameworks like OpenCL, OpenMP, CUDA, and C++AMP. These frameworks allow the allocation of parallel computation on different compute resources. This, however, creates a difficulty in allocating the right computation segments to the right processing units in such a way that the complete system meets all its timing requirements. In this paper, we consider pre-runtime static allocations of parallel tasks to perform their execution either sequentially on CPU or in parallel using a GPU. This allows for improving any unbalanced use of GPU accelerators in a heterogeneous environment. By performing several heuristic algorithms, we show that the overuse of accelerators results in a bottle-neck of the entire system execution. The experimental results show that our allocation schemes that target a balanced use of GPU improves the system schedulability up to 90%.

Paper F: *Offloading Accelerator-intensive Workloads in CPU-GPU Heterogeneous Processors*
Nandinbaatar Tsog, Saad Mubeen, Fredrik Bruhn, Moris Behnam, Mikael Sjödin.
Published in the Proceedings of the 26th International Conference on Emerging Technologies and Factory Automation (ETFA), September 2021.

Abstract: Autonomous vehicular systems require computer vision and intelligent on-board decision making functionalities that include a mix of sequential and parallel workloads. The execution times of the workloads and power consumption in these functionalities can be lowered by utilizing the accelerators (e.g., GPU) instead of running the workloads entirely on the host processing units (CPU). However, allocating all the parallelizable workload to accelerators can create a computation bottleneck in the accelerators that, in turn, can have an adverse effect on schedulability of the systems. This paper presents a novel framework that can allocate the accelerate-intensive workloads to the accelerators as well as to the non-accelerated host processing units. Within the context of this framework, the paper introduces five offloading techniques to mitigate the accelerator-intensive workloads by utilizing excess capacity of

non-accelerated processing units under dynamic scheduling in CPU-GPU heterogeneous processors. The proposed techniques are evaluated using simulation experiments. The evaluation results indicate that one of the proposed techniques can achieve up to 16% improvement in schedulability of the task sets compared to the traditional non-offloading technique.

3.3 Research Process and Methodology

A research method is a way to uncover new knowledge or create better understanding of research problems. A scientific method is the logical scheme of the research method that is used to answer to scientific questions posed within science [20]. A research methodology is the primary principle that will guide the research topic conducting a collection of the specific scientific methods [21]. Holz et al. [22] discuss the four major steps (problem definition, idea development, implementation and evaluation) of the research process that we adopt in our research. The research methodology that is used in our research is illustrated in Figure 3.2.

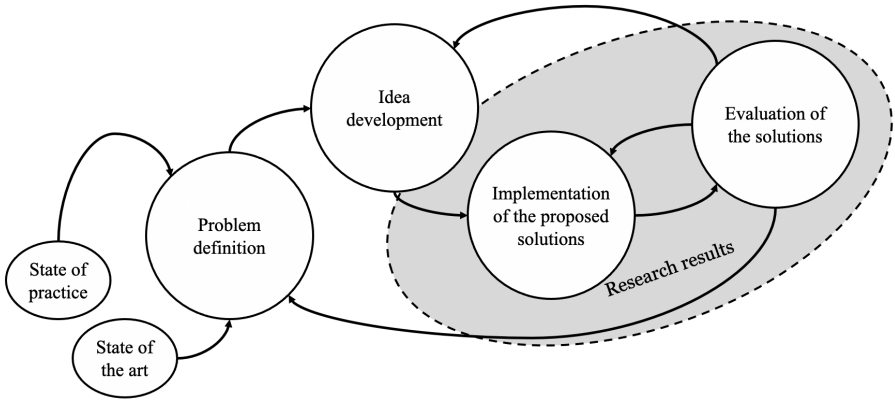


Figure 3.2: Research Methodology.

Problem definition. As first step in our research process, we have done a review of both the state of the art and practice including the reason/problem for initiation of our research which has not been investigated before. In addition to the discussion between the involved parties, the research goals are formulated as an outcome of the problem definition step. In Papers A, B, and C, we have conducted a review of the state of the practice with the help of the indus-

try, including Unibap AB (publ.), Volvo Construction Equipment, SAAB, and Advanced Micro Devices (AMD). Furthermore, we investigate how heterogeneous and parallel computing are introduced in the state of art technologies such as HSA, CUDA, OpenCL, OpenMP and so on. The literature reviews surveys have been conducted in order to understand how different communities deal with heterogeneous computing. In other words, Papers E and F are focused on heterogeneous computing in real-time systems, while Papers A and C survey the relation between heterogeneous computing and space community. As a result, we extract some ideas for the technical contributions presented in this thesis.

Idea development. After the survey study, we have chosen the most relevant works which help to consolidate our ideas. In Papers D, E, and F, we extract models from real applications and propose a solution using models to improve the existing solutions. In Papers A, B, C, and D, we have identified the metrics which are used to understand the characterization of heterogeneous architectures.

Implementation of the proposed solutions. The implementation step results with empirical studies based on either real implementation (Papers A, B, C) and simulation using the state of the art analysis tool (Papers E, F and D). Real implementations help to perform benchmarking study, i.e., measurement based experiments. Some understanding of the proposed solutions using the implementation has been published as work-in progress and workshop papers, and technical reports (listed in this thesis as not included publications) in early stage.

Evaluation of the solutions. In the last part research process, we draw conclusion and determine limitations of our approach by conducting the evaluation of the proposed solution. In the evaluation process, the introduced metrics, tools including MUST (implemented by us during this PhD journey), and research methods are used. Depending on the results of the evaluation step, the problem definition and idea development are revised and continue with the later steps. This process is iterated until the results are acceptable. The finalized results/outcomes are presented as conference and journal publications.

Chapter 4

Related Work

Space missions are constrained to bring the technological advances in COTS platforms due to the radiation effects and other limitations. Many works explore the behaviour of COTS platforms under radiation effects such as [23], [24], [25], and so on. Their focus is the effect of radiation regarding total ionizing dose (TID) and single-event effects (SEEs) on in-orbit hardware and materials used in the spacecraft. Miller et al. [26] and Troxel [23] study the radiation effect on commercial DRAMs. The studies show that the exposed particle can damage hardware, which may end up with data loss as well. Moreover, the authors report the changes of chip revision within each family can be another concern of radiation effects. Therefore, the current state of the art considers how radiation effects can affect materials of hardware that, in turn damage the stored data. This thesis explores how GPU accelerated COTS platforms fit in in-orbit missions, measuring various types of applications considering different limitations such as radiation effect, SWaP, and real-time constrains. However, there is a lack of research on investigation of radiation effects on the execution behavior of applications that are stored in the hardware. Hence, in the thesis, we provide a framework and a tool to simulate how radiation effects influence the execution behavior of applications on the on-board computing platforms including heterogeneous processing units.

Historically, the adoption of heterogeneous processing units is intimately bound to the development of high-performance computing such as supercomputers, especially, in the area of distributed heterogeneous supercomputing [12]. The execution times and manners of the workloads can vary a lot depending on what type of processing units they are executed on. To this end, several existing works focus on how to allocate applications to the appropriate processing units in order to achieve the best-case execution time, i.e., the shortest execution time [27, 28, 29]. In contrast, the work presented in the thesis con-

siders offloading the accelerator-intensive workloads, constrained by real-time requirements, e.g., deadlines on the response times of the workloads, to the available non-accelerated host processing units.

Beside radiation effects, the use of heterogeneous processing units in real-time applications is another main focus of this thesis. The heterogeneous processing units considered in the thesis consist of mainly two parts: (i) a host processing unit, CPU, and (ii) accelerator(s) that include GPUs and FPGAs, among others. There exist several research trends on how to tackle heterogeneous processing units in real-time applications. One of the research trends is to explore the properties of accelerators in heterogeneous processing units since a host processing unit is a well-studied single-core CPU. The existing explorations in this regard include TimeGraph [30], Gdev [31], the black-box method [32], to mention a few.

Another line of existing studies targets resource management in the systems that use heterogeneous processing units. There are several studies [33, 34, 35] that focus on splitting a task on accelerators for improving the schedulability. Moreover, TimeGraph [30], GPUSync [36], and the works by Kim et al. [37] and Biondi et al. [38] consider schedulability analysis of the systems that use heterogeneous processing units. These studies focus on accelerators, which obviously offer better (shorter) execution times of the compute-intensive workloads compared to the executions on the host processing units. On the other hand, this thesis aims at mitigating the accelerator-intensive workload by efficiently offloading it to the non-accelerated host possessing units.

From the real-time perspective, there exist several works that support server-based scheduling on single- and multi-core CPU(s) such as the constant bandwidth server (CBS) [39], total bandwidth server (TBS) [40], polling server (PS), sporadic server (SS) and deferrable server (DS) [41]. Some of the existing works also address the challenge of using the server-based scheduling in accelerators. For example, the works in [42, 43] show that the server-based scheduling on accelerator(s) can improve the schedulability of the systems that use heterogeneous processing units. In comparison to these works, the work presented in the thesis uses the server-based (DS) scheduling in the host processing units instead of accelerators. The rationale behind the decision is that the proposed framework, based on alternative executions of parallel segments, offloads the accelerator-intensive workloads to host processing units to efficiently utilize their excess resources to assist the accelerators.

The idea of using alternative executions of parallel segments of real-time workloads is discussed in a few works [44]. Baruah [44] uses conditional branching by using the if-then-else construct for two or more alternative executions of a workload. Moreover, a scheduling approach employs the condi-

tional DAG model for reserving the necessary amount of computing resources. This thesis discusses a static allocation of real-time tasks using alternative execution of parallel segments of the tasks. Both works construct the fundamental of alternative executions of segments under real-time constraints. However, dynamic allocation of tasks using the alternative executions of parallel segments is missing from the state of the art. Hence, provisioning of such an allocation is the main focuses of this thesis.

Chapter 5

Conclusions

5.1 Summary and Conclusions

This thesis conducts an investigation of COTS heterogeneous architectures under the real-time and space-specific constraints, i.e., space computing using COTS heterogeneous platforms. The work conducted in the thesis addresses the goal of improving timing predictability of real-time applications on heterogeneous computing platforms under harsh environments without degrading their computing performance and energy efficiency.

First, we have investigated the characteristics of processing units in the heterogeneous computing architectures focusing on computing potential and energy efficiency. We confirm that CPU performs better computing performance in the case of small workloads. Otherwise, GPU performs better than CPU in computing performance. In addition, GPU is more energy efficient compared to CPU for any type of workloads. Hence, from the execution-time perspective, we conclude that larger workloads can benefit more compared to smaller workloads by computing on GPU. This conclusion builds the base of an idea about alternative executions of parallel segments. In other words, alternative executions of parallel segments can impact on the timing predictability and energy use of the total system.

Then, the investigations of alternative executions of parallel segments have been conducted in three phases introduced. In the first phase, we have studied the characteristics of alternative executions of parallel segments considering computing performance and energy efficiency. The results indicate that the appropriate use of heterogeneous processing units for parallel segments can achieve up to 64.3% decrease in the total energy consumption by the systems. In the second phase, the study continues with the scheduling of real-time applications applying the alternative executions of parallel segments. The results in-

dicate that the schedulability of task set can be improved up to 90% compared to the existing solutions. In the third phase, we propose a dynamic scheduling technique for alternate execution of parallel segments on CPU-GPU heterogeneous computing platforms. The results indicate that the dynamic scheduling of the proposed task model can improve the schedulability up to 16% compared to the state-of-the-art solutions.

Finally, we have implemented a simulation tool for heterogeneous COTS platforms under harsh environments. In this tool, we applied the proposed task model for the parallel segments of applications. This tool opens a new research direction regarding heterogeneous COTS platforms under harsh environments, when we consider different radiation tolerance levels for the different processing units and peripheral devices.

We believe that the proposed techniques using our novel task model improve the timing predictability of real-time applications on heterogeneous computing platforms without degrading their computing performance and energy efficiency. Moreover, our implemented tool and framework build a base of a new research area for heterogeneous COTS platforms under harsh environments such as electromagnetic environments and radiation environments.

5.2 Future Work

There are several research directions for the future work:

- Validation on industrial case studies is crucial, since the need for the practical use of heterogeneous architectures increases dramatically. Concerning the proposed task model, a real-time GPU scheduler using alternative executions of parallel segments can be useful for the industrial use cases. This scheduler can be implemented by customizing HSA-compliant ROCm. ROCm is adopted already in mainline Linux kernel and the thesis confirms that it performs very stable from Linux kernel 5.0.

- Tackling the radiation effect problems from software level is an interesting research direction. In this field, the thesis has opened a new research direction using different radiation tolerance levels for different processing units, and other peripherals for computation.

- In this thesis, our focus was on the processing units. The investigation can be extended to memory management of platforms that employ heterogeneous processing units. Among the memory models introduced in Section 2.2.2, the research direction can be focused on the shared

memory architectures as it can already be divided as uniform memory access, non-uniform memory access, and heterogeneous uniform memory access. We plan to investigate the different use case scenarios of these architectures with heterogeneous processing units.

- Last, but not least, the consideration of the collaboration of heterogeneous processing units, AI, and big data can be an interesting research direction. This thesis limits AI applications as test-beds that use heterogeneous processing units. However, for example, AI optimized scheduling policy for heterogeneous processing units will be crucial when we consider big data, data inefficiency, extracting the important parts from data and their uses.

Bibliography

[1] J. Bouwmeester and J. Guo, "Survey of worldwide pico-and nanosatellite missions, distributions and subsystem technology," *Acta Astronautica*, vol. 67, no. 7-8, pp. 854–862, 2010.

[2] M. Swartwout, "Cubesats and mission success: 2017 update," in *Electronic Technology Workshop, NASA Electronic Parts and Packaging Program (NEPP), NASA Goddard Space Flight Center*, vol. 27, 2017.

[3] G. Richardson, K. Schmitt, M. Covert, and C. Rogers, "Small satellite trends 2009-2013," 2015.

[4] T. Segert, "Why did Google dump Skybox?" [Online]. Available: https://www.linkedin.com/pulse/why-did-google-dump-skybox-tom-segert/ (Accessed Nov 14, 2019)

[5] M. T. Hicks and C. Niederstrasser, "Small sat at 30: trends, patterns, and discoveries," 2016.

[6] B. Andersson, G. Raravi, and K. Bletsas, "Assigning real-time tasks on heterogeneous multiprocessors with two unrelated types of processors," in *2010 31st IEEE Real-Time Systems Symposium*. IEEE, 2010, pp. 239–248.

[7] G. Lentaris, K. Maragos, I. Stratakos, L. Papadopoulos, O. Papanikolaou, D. Soudris, M. Lourakis, X. Zabulis, D. Gonzalez-Arjona, and G. Furano, "High-performance embedded computing in space: Evaluation of platforms for vision-based navigation," *Journal of Aerospace Information Systems*, vol. 15, no. 4, pp. 178–192, 2018.

[8] L. Walsh, U. Schneider, A. Fogtman, C. Kausch, S. McKenna-Lawlor, L. Narici, J. Ngo-Anh, G. Reitz, L. Sabatier, G. Santin *et al.*, "Research plans in europe for radiation health hazard assessment in exploratory space missions," *Life sciences in space research*, vol. 21, pp. 73–82, 2019.

[9] J. Rask, W. Vercoutere, B. Navarro, and A. Krause, "Space faring: The radiation challenge," *Nasa, Module*, vol. 3, no. 8, p. 9, 2008.

[10] L. L. Bello, R. Mariani, S. Mubeen, and S. Saponara, "Recent advances and trends in on-board embedded and networked automotive systems," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 2, pp. 1038–1051, 2018.

[11] H. Andrade, L. E. Lwakatare, I. Crnkovic, and J. Bosch, "Software challenges in heterogeneous computing: A multiple case study in industry," in *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2019, pp. 148–155.

[12] R. F. Freund and D. S. Conwell, "Superconcurrency: A form of distributed heterogeneous supercomputing," NAVAL OCEAN SYSTEMS CENTER SAN DIEGO CA, Tech. Rep., 1991.

[13] B. Jeff, "Ten things to know about big. little," *ARM Holdings*, 2013.

[14] HSA Foundation, ""Heterogeneous System Architecture.","available: http://www.hsafoundation.com/ [Oct 16, 2018].

[15] M. Barr and A. Massa, *Programming embedded systems: with C and GNU development tools*. " O'Reilly Media, Inc.", 2006.

[16] G. C. Buttazzo, *Hard real-time computing systems: predictable scheduling algorithms and applications*. Springer Science & Business Media, 2011, vol. 24.

[17] C. Maia, M. Bertogna, L. Nogueira, and L. M. Pinho, "Response-time analysis of synchronous parallel tasks in multiprocessor systems," in *Proceedings of the 22Nd International Conference on Real-Time Networks and Systems*, 2014, pp. 3–12.

[18] J. A. Stankovic and K. Ramamritham, "What is predictability for real-time systems?" 1990.

[19] S. Mubeen, E. Lisova, and A. Vulgarakis Feljan, "Timing predictability and security in safety-critical industrial cyber-physical systems: A position paper," *Applied Sciences*, vol. 10, no. 9, p. 3125, 2020.

[20] G. Dodig-Crnkovic, "Scientific methods in computer science," in *Proceedings of the Conference for the Promotion of Research in IT at New Universities and at University Colleges in Sweden, Skövde, Suecia*, 2002, pp. 126–130.

[21] C. Dawson, *A–Z of Digital Research Methods*. Routledge, 2019.

[22] H. J. Holz, A. Applin, B. Haberman, D. Joyce, H. Purchase, and C. Reed, "Research methods in computing: What are they, and how should we teach them?" in *Working group reports on ITiCSE on Innovation and technology in computer science education*, 2006, pp. 96–114.

[23] I. Troxel, "Memory technology for space," *Military and Aerospace Programmable Logic Devices (MAPLD)*, 2009.

[24] D. Sinclair and J. Dyer, "Radiation effects and cots parts in smallsats," 2013.

[25] R. Kingsbury, F. Schmidt, W. Blackwell, I. Osarentin, R. Legge, K. Cahoy, and D. Sklair, "Tid tolerance of popular cubesat components," in *2013 IEEE Radiation Effects Data Workshop (REDW)*. IEEE, 2013, pp. 1–4.

[26] C. Miller, R. Owen, M. Rose, P. M. Rutt, J. Schaefer, and I. A. Troxel, "Trends in radiation susceptibility of commercial drams for space systems," in *2009 IEEE Aerospace conference*. IEEE, 2009, pp. 1–12.

[27] Y. Wen, Z. Wang, and M. F. O'boyle, "Smart multi-task scheduling for opencl programs on cpu/gpu heterogeneous platforms," in *2014 21st International conference on high performance computing (HiPC)*. IEEE, 2014, pp. 1–10.

[28] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *2009 IEEE international symposium on workload characterization (IISWC)*. Ieee, 2009, pp. 44–54.

[29] P. Czarnul and P. Rościszewski, "Optimization of execution time under power consumption constraints in a heterogeneous parallel system with gpus and cpus," in *International Conference on Distributed Computing and Networking*. Springer, 2014, pp. 66–80.

[30] S. Kato, K. Lakshmanan, R. Rajkumar, and Y. Ishikawa, "Timegraph: Gpu scheduling for real-time multi-tasking environments," in *2011 USENIX Annual Technical Conference (USENIX ATC 11)*, 2011, pp. 17–30.

[31] S. Kato, M. McThrow, C. Maltzahn, and S. Brandt, "Gdev: First-class gpu resource management in the operating system," in *2012 USENIX Annual Technical Conference (USENIX ATC 12)*, 2012, pp. 401–412.

[32] N. Otterness, M. Yang, S. Rust, E. Park, J. H. Anderson, F. D. Smith, A. Berg, and S. Wang, "An evaluation of the NVIDIA TX1 for supporting real-time computer-vision workloads," in *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2017, pp. 353–364.

[33] C. Basaran and K.-D. Kang, "Supporting preemptive task executions and memory copies in GPGPUs," in *2012 24th Euromicro Conference on Real-Time Systems*. IEEE, 2012, pp. 287–296.

[34] S. Kato, K. Lakshmanan, A. Kumar, M. Kelkar, Y. Ishikawa, and R. Rajkumar, "Rgem: A responsive gpgpu execution model for runtime engines," in *2011 IEEE 32nd Real-Time Systems Symposium*. IEEE, 2011, pp. 57–66.

[35] E. Rossi, M. Damschen, L. Bauer, G. Buttazzo, and J. Henkel, "Preemption of the partial reconfiguration process to enable real-time computing with FPGAs," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 11, no. 2, pp. 1–24, 2018.

[36] G. A. Elliott, B. C. Ward, and J. H. Anderson, "Gpusync: A framework for real-time gpu management," in *2013 IEEE 34th Real-Time Systems Symposium*. IEEE, 2013, pp. 33–44.

[37] H. Kim, P. Patel, S. Wang, and R. R. Rajkumar, "A server-based approach for predictable gpu access control," in *2017 IEEE 23rd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2017, pp. 1–10.

[38] A. Biondi, A. Balsini, M. Pagani, E. Rossi, M. Marinoni, and G. Buttazzo, "A framework for supporting real-time applications on dynamic reconfigurable FPGAs," in *2016 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2016, pp. 1–12.

[39] L. Abeni and G. Buttazzo, "Integrating multimedia applications in hard real-time systems," in *Proceedings 19th IEEE Real-Time Systems Symposium (Cat. No. 98CB36279)*. IEEE, 1998, pp. 4–13.

[40] M. Spuri and G. C. Buttazzo, "Efficient Aperiodic Service Under Earliest Deadline Scheduling," in *RTSS*, 1994, pp. 2–11.

[41] H. Zhu, S. Goddard, and M. B. Dwyer, "Response time analysis of hierarchical scheduling: The synchronized deferrable servers approach," in *32nd Real-Time Systems Symposium*. IEEE, 2011, pp. 239–248.

[42] S. Kato, K. Lakshmanan, Y. Ishikawa, and R. Rajkumar, "Resource sharing in GPU-accelerated windowing systems," in *2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE, 2011, pp. 191–200.

[43] Y.-S. Chen, H. C. Liao, and T.-H. Tsai, "Online real-time task scheduling in heterogeneous multicore system-on-a-chip," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 1, pp. 118–130, 2012.

[44] S. Baruah, "Resource-efficient execution of conditional parallel real-time tasks," in *European Conference on Parallel Processing*. Springer, 2018, pp. 218–231.

Part II

Included Papers

Chapter 6

Paper A
Intelligent Data Processing using In-Orbit Advanced Algorithms on Heterogeneous System Architecture

Nandinbaatar Tsog, Moris Behnam, Mikael Sjödin, Fredrik Bruhn
In the Proceedings of the 39th International IEEE Aerospace Conference, AeroConf 2018

Abstract

In recent years, commercial exploitation of small satellites and CubeSats has rapidly increased. Time to market of processed customer data products is becoming an important differentiator between solution providers and satellite constellation operators. Timely and accurate data dissemination is the key to success in the commercial usage of small satellite constellations which is ultimately dependent on a high degree of autonomous fleet management and automated decision support. The traditional way for disseminating data is limited by on the communication capability of the satellite and the ground terminal availability. Even though cloud computing solutions on the ground offer high analytical performance, getting the data from the space infrastructure to the ground servers poses a bottleneck of data analysis and distribution. On the other hand, adopting advanced and intelligent algorithms onboard offers the ability of autonomy, tasking of operations, and fast customer generation of low latency conclusions, or even real-time communication with assets on the ground or other sensors in a multi-sensor configuration.

In this paper, the advantages of intelligent onboard processing using advanced algorithms for Heterogeneous System Architecture (HSA) compliant onboard data processing systems are explored. The onboard data processing architecture is designed to handle a large amount of high-speed streaming data and provides hardware redundancy to be qualified for the space mission application domain. We conduct an experimental study to evaluate the performance analysis by using image recognition algorithms based on an open source intelligent machine library "MIOpen" and an open standard "OpenVX". OpenVX is a cross-platform computer vision library.

6.1 Introduction

Small satellites (i.e., satellites defined weighing less than 100 kg) all the way down to nanosatellites (i.e. satellites defined weighing less than 10 kg) are rapidly attracting interest in many areas including the commercial telecommunication, Earth Observation (EO) markets, and the intelligence and defense community [1]. EO satellites are experiencing rapid advancements in optical imaging payload technologies and onboard processing, leading to significantly improved quality and resolution of imagery gathered from spaceborne platforms. The smaller size of satellites together with a lower cost has allowed the use of high performing Commercial-Off-The-Shelf (COTS) electronic parts to be harnessed for image compression and cloud removal using both Graphical Processing Units (GPUs) and Field Programmable Gate Arrays (FPGAs) [2, 3]. The improvements in sensor technology have not been matched with equivalent developments in satellite downlink technologies, and hence the exponential increases in the generated data volume are forming a significant bottleneck onboard the platform. Optical communication holds promise to enable gigabit per second telemetry data transfer for downlinks and intersatellite links [4]. This would decrease the difference between sensor advances and the communication bottleneck. However, latency and storage capacity will still be big challenge since the number of places on Earth with suitable optical stations are limited.

Many emerging missions use constellations of many (e.g., over 100) small satellites to enable rapid revisit times and global coverage [5]. Small satellites are being deployed for many different applications, e.g., communications, space situational awareness, and Intelligence, Surveillance, and Reconnaissance (ISR), precision agriculture (PA), machine to machine communication, and air traffic management.

In order to address the latency issues and communication bottlenecks, more onboard data analytics is required to enable small satellites to accommodate the flexibility needed for autonomous constellation management, information extraction, compression and sensor fusioning with low latency. However, it is important to find data processing solutions that fit the Size, Weight, and Power (SWaP) constraints while collecting mission-critical sensor data. This paper further explores the use, efficiency and performance of HSA capabilities of modern System-on-Chips (SoC) for ISR sensors (e.g., EO/Infrared/Hyperspectral cameras) using the GIMME-series architectures [6].

GIMME-3 is an architecture developed at Mälardalen University to pursue a SWaP optimized onboard computing solution that enables Deep Learning on massively parallel units with advanced Error Code Correction (ECC) for

aerospace application. GIMME-3 has been expanded to GIMME-4 which introduces HSA capability through the AMD® R-series SoC (f.m. named Merlin Falcon) [6].

Radar is an another important sensor which usually requires massive processing. New methods suitable for onboard processing are being developed. Single-frequency transmitted wave-forms with high Doppler resolution nature called Doppler synthetic aperture radar (D-SAR) is one interesting approach for bistatic radar [7].

6.1.1 Contributions

The main contribution of this paper is to investigate the performance of onboard data processing on the heterogeneous architecture by running image processing algorithms which use both MIOpen framework of high performing machine learning primitives and OpenVX vision library. We focus on the fact that the concurrent executions of multiple advanced algorithms affect the worst-case execution time (WCET) of other parallel running tasks which of expresses the quality of the onboard heterogeneous system. Since less energy consumption is an another key factor to increase the quality of the onboard architecture, we have focused on the energy consumption of GPU based heterogeneous computing. We confirmed that an HSA compliant GPU based heterogeneous computing improves the quality of the onboard architecture for intelligent data processing since it either uses less energy consumption and performs better than CPU for the feature tracking with the different workloads.

6.1.2 Organization

In Section 2, we discuss an importance of onboard processing, usage of in-orbit heterogeneous architectures and energy consumption of parallel architectures running advanced algorithms. The architectures and specifications are introduced in Section 3. We describe our benchmark suites in Section4 and the evaluation of the experiments are described in Section 5. Section 6 concludes the paper and discusses future work.

6.2 Related Work

In this work, we consider the contribution of heterogeneous architecture in mission critical applications such as onboard processing. Advanced and resource-intensive computing in new science-mission applications brings a new challenge to the space-computing community as these needs require next-generation

systems that should support a broad potential of processing with low power consumption and high reliability [8]. Certain numbers of the heterogeneous multicore SoC platforms are introduced as a promising architecture for space-computing. In order to increase the reliability of such platforms, Wilson et al. consider a multifaceted strategy (HARFT strategy) for fault-tolerant computing, targeting SoC platforms consists of multicore CPUs and FPGA fabric. The HARFT strategy introduces fault-tolerant schemes by using both compute nodes to achieve a robust, hybrid, hardware redundant and fault-tolerant theme for a hybrid device.

Heterogeneous architectures including FPGA are not only the hybrid solution in space-computing, but also several architectures including GPUs exist. However, fault masking and tolerance on GPUs is less investigated for harsh environments [9]. Milluzzi and George discuss GPU protection on Tegra X1 SoC for space usage, i.e., how to avoid vulnerability of Tegra SoCs against a wide range of single-event upsets (SEUs) since it has complex caching structure which consists of a number of the GPU cores and a custom task scheduler. As a GPU protection, they consider a persistent thread method with triple-modular redundancy (TMR) which provides a strong basis for fault masking on a wide range of platforms. They have succeeded to remove the vulnerability of scheduler faults even when GPUs pose a unique challenge to a general TMR implementation. The paper reports that the NVIDIA Tegra™ K1 and X1 perform over 500 GFLOPS of peak performance at just 10 Watts Thermal Design Power (TDP). Hence, the SoC considered in our paper possesses sufficient computational performance that it employees Radeon™ R6 GPU which has the computational potential up to 614GFLOPS and less than 10Watt in peak performance.

Persistent threads style programming model/method is well-known to protect GPU from the interference of host CPU since it performs the direct communication with the different GPU kernels instead of unnecessary round-trip communications through host CPU. Moreover, this method is useful for FPGA protection as well. Khan et al. present a complete networking switch designed in OpenCL that consists of several high-level constructs which create the building blocks of any network application for FPGAs [10]. In this work, persistent kernel method is used to avoid the intervention of the host to provide the kernels with data processing constantly. Measuring the intervention of the tasks to one another and between the different compute nodes is one of the main challenges in our work to assess the quality of the concurrent executions.

Measuring the energy consumption of the advanced algorithms while running on the onboard computer is another challenge in our work. Liu et al. tackle with an advanced algorithm using the aerosol optical depth (AOD) prop-

erties from the performance and energy efficiency perspective [11]. As a result of a large number of remote sensing data and compute-intensive algorithms, the AOD retrieval is computationally expensive. Two different kinds of parallel architectures, multicore processors and GPU accelerators, are used to run the time-consuming SRAP-MODIS algorithm for the AOD retrieve. This algorithm includes not only a set of nonlinear equations but also requires a large number of input images. In this paper, the performance of parallel computations on both of the architectures and energy consumptions are analyzed in the context of a quantitative remote sensing retrieval application. The difference of the power consumptions between the idle and load conditions [12] is used as the power consumption of the applications for the multicore and GPU in order to evaluate the power consumption. We use this measurement method to determine the real power consumption of the application running.

In order to adapt to unexpected situations, acquirement of cognitive capabilities is important to autonomous control systems in space [13]. Q-learning is a model-free reinforcement learning technique and it is efficient in solving some classes of learning problems. Due to the constraints, SWaP, convergence rate and costs, learning algorithms are rarely implemented in onboard embedded systems in space. Similarly to exploring Convolution Neural Network in our paper, Gankidi and Thangavelautham present Q-learning with Aritificial Neural Network. This method fits well with the parallel computing and it achieved a great reducing processing time by using the fine-grain parallelism of an FPGA hardware. The result shows 43x speed up by Virtex 7 FPGAs compared to Intel i5 2.3 GHz CPUs. They emphasize that the fine grained parallel architectures are competitive considering power consumption.

6.3 Background

6.3.1 AMD A-Series A10-8700P APU

As illustrated in Figure 6.1, the AMD A10-8700P APU maintained in a SoC comprises a quad core 1.8GHz 64bit A10 CPU and Radeon™ R6 GPU. The CPUs consist of 2 compute units (CUs) each of which has 2 cores and shares 1MB L2 cache. The cores integrated into the same CU share 96KB L1 instruction cache, and 32KB L1 data caches are unique to each core. The GPU consists of 6 CUs with 64 cores each, and totally performs up to 614GFLOPS. The APU shares 8GB DDR3 handled by memory controllers with a full hardware cache coherence at 128 bit-wide memory bandwidth between GPU and CPU caches. An Address Translation Cache (ATC) hierarchy brings a fine grained shared virtual memory, i.e., the same virtual address space to all devices. This

feature is known as a foundational aspect of HSA that is merely passing the pointer to data between all the devices and no memory copying required for the different CUs. The AMD Embedded G- and R-Series SoCs are used with the A-series APUs and there exist COTS products such as conga-TR3[1] with the size of 95mm by 125 mm. The AMD Embedded R-series SoC is based on a 28 nm process and compliant with HSA 1.0 thus meaning it supports fine-grain full cache coherency. The power consumption of this APU ranges between from 12 Watt up to 35 Watt while the thermal design power (TDP) is 15Watt.

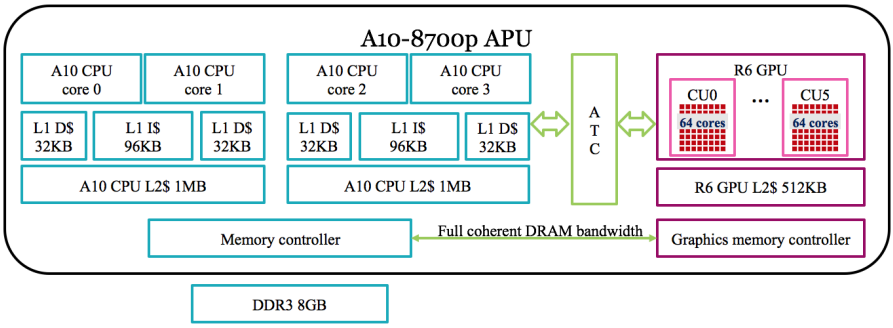


Figure 6.1: AMD A-series A10 APU's architecture

6.3.2 GIMME3 and GIMME4

A fault tolerant heterogeneous architecture, GIMME3, has been designed for high performance computing in mission critical applications [14]. GIMME3 architecture was productized and commercialized by Unibap AB (publ) and was flown into space on May 30th 2016 on the Satellogic NuSat-1 and NuSat-2 [6]. GIMME3 employees the AMD Embedded G-series SoC based on FT3 and FT3(b) footprints (formerly known as Kabini and Steppe Eagle, respectively) that support up to quad core CPU with 2 GPU CUs [15]. Each AMD GPU CU has 64 Arithmetic Logic Units (ALU) and GIMME3 can deliver 77 GFLOPS of GPU performance.

As an expansion of GIMME3, Tsog et al. introduce the next generation heterogeneous computing architecture GIMME4 using HSA for higher computing performance and better redundancy [6]. Similarly to the AMD A10-series APU, GIMME4 architecture is based on the AMD Embedded 2nd Gen

[1] conga-TR3:
http://www.congatec.com/en/products/com-express-type6/conga-tr3.html

R-series SoC [16] with 8 GPU CUs and FP4 footprint, formerly known as Merlin Falcon. Major differences between the FT3(b) and FP4 footprint based products are the shift in CPU architecture from 'Bulldozer' to 'Excavator', updated GPU design, memory controller, and the official support for HSA [6]. The Unibap e2200 family development board based on GIMME4 architecture is 82 mm by 110 mm and 85g, and provides 819GFLOPS of GPU performance. Currently, GIMME4 has not tested yet in radiation environments, however, the previous version GIMME3 is fully confirmed [14] that it operates in radiation environments.

6.3.3 Heterogeneous System Architecture

In modern trend in industrial applications, the role of heterogeneous computing has been increasing dramatically. Employing multiple types of compute nodes, CPU, GPU, FPGA, DSP and so on, according to their strengths makes the embedded systems as robust as much. However, the different types of specifications and designs of the compute nodes bring difficulties for the developing process from cost and timing perspective. To overcome these problems, multiple leading hardware vendors have established HSA Foundation[2] to develop the Heterogeneous System Architecture (HSA) specification for reducing heterogeneous computing complexity and providing the developer friendly environments. The HSA aims to ease the process of developing the heterogeneous platform by providing the similar environment for the developers such as they used for the legacy systems, i.e., homogeneous systems. For example, providing the open-source well-known compilers, LLVM and GCC, and using only pointers in a virtual memory space gives access to the memory spaces of all the compute nodes. The virtual memory space in the HSA provides no memory copying between different physical memories, i.e., the HSA provides unified coherent memory that saves a lot of computation time for transferring data between different physical memories. As a part of HSA, AMD contributes by introducing an initiative GPUOpen[3], an open-source software stack, including, but not limited to, kernel level driver, runtime, tools and libraries such as ROCm, MIOpen, AMD OpenVX and CodeXL.

ROCm

ROCm is an open source software stack and consists of multiple modules which support GPU computing [6].

[2]HSA Foundation: http://www.hsafoundation.com
[3]GPUOpen: https://gpuopen.com

MIOpen

MIOpen, an alternative to CuDNN, is an open-source machine learning library that developed to exert full potential of ROCm software stack as well as heterogeneous computing. In the current release (version 1.0), MIOpen supports Convolution Neural Network (CNN), Pooling, Softmax, Activations, Gradient Algorithms Batch Normalization, and LR Normalization[17] with data described in 4-D tensors. Both OpenCL and HIP frameworks are enabled in MIOpen that HIP includes a tool "hipify" which ports CUDA code into C++.

OpenVX

The Khronos Group has designed an open and royalty-free standard OpenVX that is portable across different vendors and hardware types, and optimized and power-efficient image processing for computer vision applications. OpenVX enables the following use cases; face, body and gesture tracking, smart video surveillance, advanced driver assistance systems (ADAS), object and scene reconstruction, augmented reality, visual inspection, robotics and more [18]. Moreover, OpenVX and OpenCV complement each other to perform as perfect computer vision library since OpenVX has to be implemented by hardware vendors and OpenCV has a strong open source community. Not only, OpenVX is the computer vision library, but also it has great potential to being as a machine learning library in its Neural Network Extension. There are multiple vendors implement their OpenVX libraries for both computer vision and neural network libraries. However, we focus only on computer vision library and use AMD OpenVX (AMDOVX) [19] in this paper. Currently, the released version of the AMDOVX is 0.9.6 and it includes feature tracking "Optical flow" algorithm as well as feature detection algorithms, e.g. Harris, FAST, Canny. Furthermore, the current version interoperates with OpenCV as well.

CodeXL

CodeXL[4] is an open-source development tool suite, debugging and profiling, for the different processors such as CPU, GPU, and APU. Using CodeXL facilitates the HSA development process as it provides debugging functionality for OpenGL, OpenCL and HSA, and profiling functionality for both OpenCL and HSA kernels. CodeXL works on both Windows™ and Linux™ as a Visual Studio™ extension and a standalone user interface with both graphical and command line. In this paper, we deal with CodeXL mainly for power profiling purpose, however, it is used for debugging purpose as well.

[4]CodeXL: https://gpuopen.com/compute-product/codexl/

6.4 Experiment Setup

As illustrated in Figure 6.2, we consider a comparison of the process of a reporting system using satellite data. The traditional reporting system is depicted with solid lines and it consists of storing raw data on the onboard computer of satellite, downlinking to the ground station, processing data on the ground station and creating the report. Meanwhile, the report creating system using onboard processing is illustrated with dash lines. The system begins with onboard processing and storing analyzed data on the onboard computer. The report will be ready once the analyzed data is downlinked to the ground station. In this paper, we consider the onboard processing only.

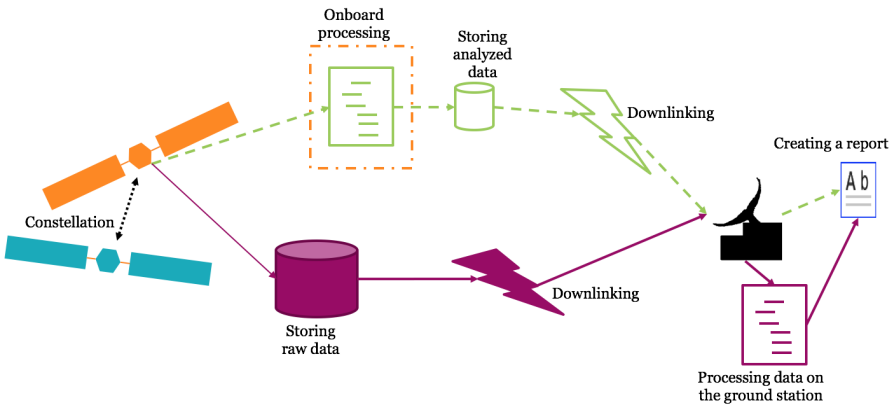


Figure 6.2: A reporting system using satellite data

The SWaP is the key to evaluate advancing onboard processing while assessing the quality of the onboard processing. The size constraint is satisfied as both the Unibap e2200 family product and the conga-TR3 fit in the 1.5U CubeSats or more. From the weight perspective, the Unibap e2200 family product takes less than 10% of the entire weight of 1U CubeSat as specified by the CubeSat standard and less than 3% of 3U CubeSat which is the preferred size of CubeSats for the advanced missions. By the specification of the board/system, the power consumption is known as between 15-35Watt that fits for the 3U CubeSats [20]. However, we investigate the detail of power consumption to find the real power consumption of the advanced onboard processing while accessing the quality of the onboard processing.

6.4.1 Benchmark Suites

We design the following experiment scenarios by using MIOpen and OpenVX with the CodeXL in order to assess the HSA compliant onboard computer for advanced processing.

- *ExpA - An investigation of the computational performance and power consumption in CPU and GPU.* The goal of this experiment is to investigate the computational performance and power consumption of advanced algorithms on CPU and GPU devices of the HSA compliant onboard computer. We use the CodeXL profiler to measure the power consumption of the advanced algorithms. To the best of our knowledge, CodeXL is an optimal tool to measure the power consumption since it omits to equip physical measuring tools.

- *ExpB - Assessing a quality of the concurrent executions of advanced tasks.* In this experiment, we aim to investigate the quality of the HSA compliant onboard computer by executing the multiple concurrent advanced algorithms. We consider a comparison of the measurement-based worst case execution time (WCET) of a task with different workloads on the CPU and GPU.

6.4.2 Configuration of Test Scenarios

In the experiment ExpA, we consider the following 7 tasks (in Table 6.1) on both CPU and GPU computations; OVX1, OVX2, ML1-1, ML1-2, ML1-3, ML1-4 and ML1-5. OVX1 and OVX2 are based on a tracking algorithm with the different test data [21] and [22], respectively. ML1-1, ML1-2, ML1-3, ML1-4 and ML1-5 are the following machine learning applications of MIOpen with the default configurations, respectively; Activations, Batch Normalization, CNN, LR Normalization and Pooling.

Shortened name	Detailed name
OVX1	Tracking algorithm with test data 1 [21]
OVX2	Tracking algorithm with test data 2 [22]
ML1-1	Activations
ML1-2	Batch Normalization
ML1-3	CNN
ML1-4	LR Normalization
ML1-5	Pooling

Table 6.1: Tasks' shortened and detailed name list

The combinations of the following 5 tasks are used in the experiment ExpB; OVX1, OVX2, ML10, ML100 and ML1000. OVX1 and OVX2 are the same as we defined for the experiment ExpA. ML10, ML100 and ML1000 are the concurrent executions of the machine learning applications Activations, Batch Normalization, CNN, LR Normalization and Pooling with a custom configurations such as 10, 100 and 1000 iterations, respectively. We measure the WCET of the task OVX1 while we consider the following tasks as the workloads; OVX2, ML10, ML100 and ML1000. The task OVX1 converts a 4K image to 1280x720 RGB image for every frame and tracks the features.

6.4.3 Test Data

In this paper, we use the International Space Station (ISS) Expedition 42's time lapse videos of earth as test data. The resolution of the videos is 4K 3840x2160 and the size of each frame is around 6MB. The test data 1 and 2 have 180 and 600 frames, respectively. The frame rate of both the data is 60fps.

Tasks	Computation time			Energy consumption		
	GPU [ms]	CPU [ms]	Ratio=CPU/GPU	GPU [Joules]	CPU [Joules]	Ratio=CPU/GPU
OVX1	79.33	137.35	1.73	4.41	4.78	1.08
OVX2	31.18	93.62	3.00	3.92	4.34	1.11
ML1-1	1.12	0.66	0.58	1.09	1.14	1.05
ML1-2	0.19	22.34	119.67	0.73	0.87	1.19
ML1-3	12.06	2873.56	238.20	1.63	22.01	13.52
ML1-4	0.57	86.82	153.23	0.75	1.43	1.89
ML1-5	1.73	29.65	17.16	0.76	0.99	1.31

Table 6.2: The computation time and energy consumption of the tasks per frame or iteration in the ExpA.

6.4.4 Evaluation Environment

The experiments are performed on A10-8700P APU employed Acer E15 E5-552-T99R model notebook. The following software stacks are installed in the environment; Ubuntu 16.04, Linux Kernel 4.14.rc3 , ROCm 1.6.3, CodeXL 2.5-25, MIOpen 1.0 and OpenVX 0.9.6. To reproduce the experiments, we have implemented the patches[5] to the CodeXL, since the current version of CodeXL is not suitable for the newer versions of the Linux Kernel than 4.10.

[5]The patches to CodeXL 2.5-25:
https://github.com/GPUOpen-Tools/CodeXL/issues/161#issuecomment-337128790

6.5 Experiment Results

The accuracy of the tasks (algorithms) are confirmed by the comparison of calculations of both CPU and GPU. We have confirmed the optical flow and Harris feature algorithms of OpenVX as shown in Figure 6.6 in Appendix 6.7. The results of the ExpA are shown in Table 6.2. Both the results of computation time and energy consumption are measured per frame and iteration for OpenVX and MIOpen applications, respectively. We can see that GPU computes faster than CPU except in case of the ML1-1 task. The speed up ratio reaches up to 238 times in the ML1-3 task which is based on one the most well-known algorithm CNN. From the energy consumption perspective, GPU leads CPU for all the cases. Moreover, GPU consumes surprisingly 13.52 times less energy in case of CNN algorithm as GPU reaches 6 Watts for the full performance in contrast with the usage of CPU reaches 3 Watts. As we know the results are per unit (frame and iteration), this experiment shows that GPU is a potential candidate for the onboard processing.

Processor	Task / Workloads				
	OVX1	OVX2	ML10	ML100	ML1000
GPU [s]	7.11	9.34	0.25	1.92	21.51
CPU [s]	11.52	24.32	27.57	320.05	5406.53
CPU/GPU	1.62	2.60	111.94	166.82	251.30

Table 6.3: The computation time of each workload running alone (no concurrent executions of any other tasks as well as the workloads).

The aim of the expB is to investigate the measurement-based WCET of the task (OVX1) while running with the different workloads. The computation time of the workloads in the expB is the total computation time of the tasks, meanwhile we considered the computation time per unit (frame or iteration) similar to the case in the ExpA. In Table 6.3, we can see that the computation time of the workloads are different. Moreover, the computation time of the workloads is measured when each workload runs alone, i.e., no concurrent executions of any other tasks at the same time. We can see that it is better to run the machine learning algorithms on GPU than CPU since the ratio of CPU/GPU increases when the iteration number of a running algorithm increases.

Figure 6.3 shows the variation of the computation time of OVX1 while running together with the different workloads. Detail information are shown in Table 6.4. We can see that the computation time of the OVX1 slows down to 2.82 times from 11.52s (no workload) to 32.56s (OVX1+OVX2+ML1000) on

CPU. Meanwhile, the increase of the computation time stays 2.23 times from 7.11s (no workload) to 15.86s (OVX1+OVX2+ML1000) on GPU. Therefore, GPU takes from 40ms (7.11s/179) to 89ms for the calculation of each frame, and CPU takes from 64ms to 182ms for each frame. As shown in Figure 6.3, we confirm that the computation time of the GPU is more stable than CPU. The task OVX1 uses the data1 which is 180frames, 4K resolution and 60fps. In other words, it takes 1/60s=17ms per frame. Therefore, the computation time of CPU is approximately 10.71 times more time compare to the frame rate time of the video. However, GPU consumes approximately 5.24 times more time of the frame rate time of the video with 4K resolution, and it could be considered reasonable with the lower frame rate (20-30fps) and the lower resolution of the images.

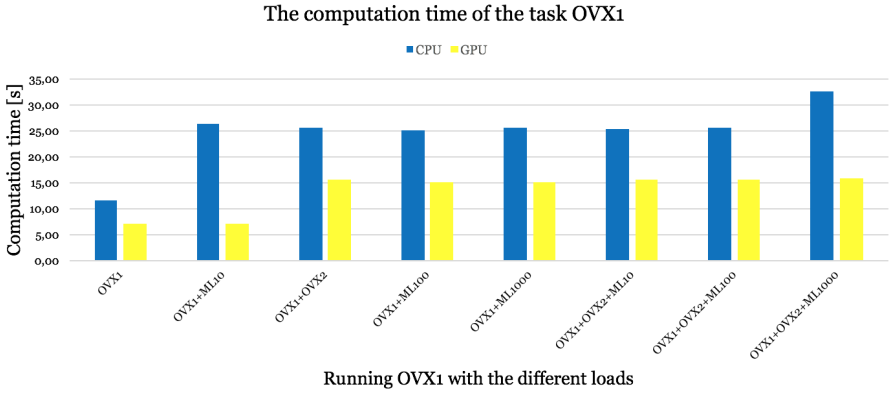


Figure 6.3: The variation of the computation time of OVX1 while running together with the different workloads.

Concurrent executions	The computation time of the task OVX1	
	CPU [s]	GPU [s]
OVX1 + no workload	11.52	7.11
OVX1 + ML10	26.36	7.24
OVX1 + OVX2	25.50	15.59
OVX1 + ML100	25.17	15.17
OVX1 + ML1000	25.64	15.17
OVX1 + OVX2 + ML10	25.39	15.55
OVX1 + OVX2 + ML100	25.74	15.66
OVX1 + OVX2 + ML1000	32.56	15.86

Table 6.4: The computation time of the task OVX1 while running together with the different workloads in the ExpB.

6.6 Conclusion / Future Work

First of all, we have come to conclusion that GPU is a potential candidate for the onboard computer processing of the CubeSat as we performed two experiments over 7 different machine learning and computer vision algorithms. From our experimental study, we have confirmed that the HSA compliant GPU computes up to 238 times faster and consumes between 13.5 times less energy, compared to the CPU calculation. Moreover, we have confirmed that the computation time of GPU is more stable than CPU while running together with the different workloads. Therefore, we conclude that GPU can be a highly potential candidate in the onboard computer processing of the CubeSat. For future work, we would like to continue developing combined usage of the intelligent applications for the onboard computer of the CubeSat that allows more usability and reliability.

6.7 Test Data

6.7.1 Source of the Test Data

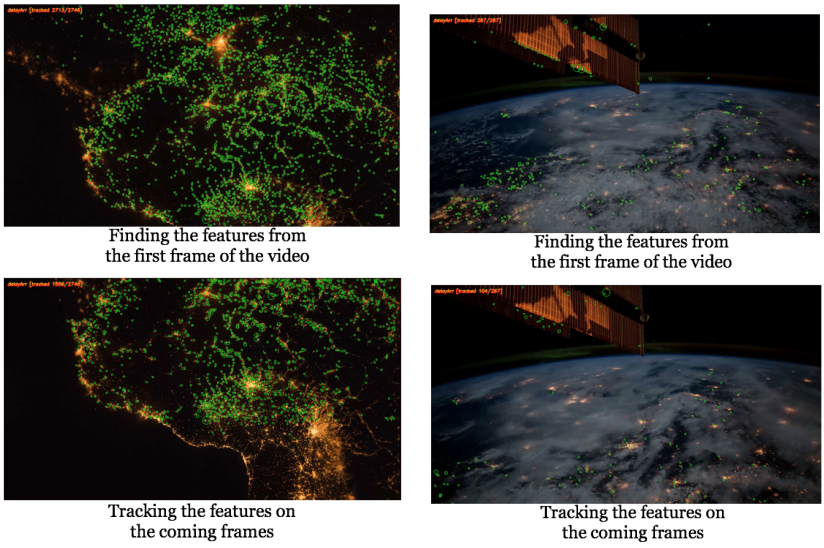
The first frames of ISS Expedition 42's time lapse videos with the id number "jsc2015m000221" and "jsc2015m000226" are shown in Figures 6.4, and 6.5, respectively. These time lapse videos are assembled from JSC still photo collections (still photos iss042e255412 - iss042e255592 and iss042e283240 - iss042e283840).

Figure 6.4: The first frame image from ISS Expedition 42 Time Lapse Video of Earth with the id "jsc2015m000221"

Figure 6.5: The first frame image from ISS Expedition 42 Time Lapse Video of Earth with the id ”jsc2015m000226”

6.7.2 Tracking Results

The results of the tasks OVX1 and OVX2 are shown in Figure 6.6.



(a) Applying OVX1 to the data1 (b) Applying OVX2 to the data2

Figure 6.6: Applying OVX1 and OVX2 to the data1 and data2, respectively

6.8 Pseudo Code for the Measurements of the Computation Time

A guide to measure the computation time is shown in the following pseudo code.

Algorithm 1 How to set a timestamp to the MIOpen code

```
* Include header files located in MIOpen/driver/.
- #include "InputFlags.hpp"
- #include "timer.hpp"

* Declare a timestamp variable
- Timer t;

* Set the timestamp for START
- START_TIME;

* Set the timestamp for END and calculate the computation time
if inflags.GetValueInt("time") == 1 then
    STOP_TIME;
    if WALL_CLOCK then
        print t.gettime_ms();
    end if
end if
```

Acknowledgments

We would like to express our sincere gratitude to Dr Harris Gasparakis, an AMD GPGPU, Computer Vision and Machine Learning technical expert and project manager, for his great knowledge in computer vision, machine learning and HSA related areas. He has helped us a great deal by providing an extensive amount of support whenever necessary.

Bibliography

[1] P. Hershey, B. Wolpe, J. Klein, and C. Dekeyrel, "System for small satellite onboard processing," in *2017 Annual IEEE International Systems Conference (SysCon)*. IEEE, 2017, pp. 1–6.

[2] R. Davidson and C. Bridges, "Gpu accelerated multispectral eo imagery optimised ccsds-123 lossless compression implementation," in *2017 IEEE Aerospace Conference*. IEEE, 2017, pp. 1–12.

[3] L. Santos, L. Berrojo, J. Moreno, J. F. López, and R. Sarmiento, "Multispectral and hyperspectral lossless compressor for space applications (hyloc): A low-complexity fpga implementation of the ccsds 123 standard," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 9, no. 2, pp. 757–770, 2015.

[4] R. P. Welle, S. Janson, D. Rowen, and T. Rose, "Cubesat-scale laser communications," in *31st space symposium*, 2015.

[5] C. Boshuizen, J. Mason, P. Klupar, and S. Spanhake, "Results from the planet labs flock constellation," 2014.

[6] N. Tsog, H. Gasparakis, M. Behnam, M. Sjödin, and F. Bruhn, "Technical Report: Advancing Onboard Computer Data Processing in CubeSats," Tech. Rep., 2017.

[7] L. Wang and B. Yazici, "Bistatic synthetic aperture radar imaging using ultranarrowband continuous waveforms," *IEEE transactions on image processing*, vol. 21, no. 8, pp. 3673–3686, 2012.

[8] C. Wilson, S. Sabogal, A. George, and A. Gordon-Ross, "Hybrid, adaptive, and reconfigurable fault tolerance," in *2017 IEEE Aerospace Conference*. IEEE, 2017, pp. 1–11.

[9] A. Milluzzi and A. George, "Exploration of tmr fault masking with persistent threads on tegra gpu socs," in *2017 IEEE Aerospace Conference*. IEEE, 2017, pp. 1–7.

[10] J. Khan, P. Athanas, S. Booth, and J. Marshall, "Opencl-based design pattern for line rate packet processing," in *2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2017, pp. 190–194.

[11] J. Liu, D. Feld, Y. Xue, J. Garcke, and T. Soddemann, "Multicore processors and graphics processing unit accelerators for parallel retrieval of aerosol optical depth from satellite data: implementation, performance, and energy efficiency," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 8, no. 5, pp. 2306–2317, 2015.

[12] S. Hong and H. Kim, "An integrated gpu power and performance model," in *Proceedings of the 37th annual international symposium on Computer architecture*, 2010, pp. 280–289.

[13] P. R. Gankidi and J. Thangavelautham, "Fpga architecture for deep learning and its application to planetary robotics," in *2017 IEEE Aerospace Conference*. IEEE, 2017, pp. 1–9.

[14] F. Bruhn, K. Brunberg, J. Hines, L. Asplund, and M. Norgren, "Introducing radiation tolerant heterogeneous computers for small satellites," in *2015 IEEE Aerospace Conference*. IEEE, 2015, pp. 1–10.

[15] AMD, "Embedded G-Series SoC Processors — AMD," (accessed 2017-10-12). [Online]. Available: http://www.amd.com/en-us/products/embedded/processors/g-series (accessed 2017-10-12)

[16] ——, "Application Brief of AMD Embedded R-Series SoC," 2017. [Online]. Available: http://www.amd.com/Documents/merlin-falcon-product-brief.pdf (accessed 2017-10-12)

[17] ——, "MIOpen." [Online]. Available: https://github.com/ROCmSoftwarePlatform/MIOpen (accessed 2017-10-05)

[18] Khronos, "OpenVX." [Online]. Available: https://www.khronos.org/openvx/ (accessed 2017-10-09)

[19] AMD, "AMD OpenVX." [Online]. Available: https://gpuopen.com/compute-product/amd-openvx/ (accessed 2017-10-09)

[20] S. S. Arnold, R. Nuzzaci, and A. Gordon-Ross, "Energy budgeting for cubesats with an integrated fpga," in *2012 IEEE Aerospace Conference*. IEEE, 2012, pp. 1–14.

[21] NASA, "ISS Expedition 42 Time Lapse Video." [Online]. Available: https://images.nasa.gov/%5C#/details-jsc2015m000221.html; (accessed: 2017-10-19)

[22] ——, “ISS Expedition 42 Time Lapse Video.” [Online]. Available: https://images.nasa.gov/%5C#/details-jsc2015m000226.html (accessed 2017-10-19)

Chapter 7

Paper B
A Trade-Off between Computing Power and Energy Consumption of On-Board Data Processing in GPU Accelerated In-Orbit Space Systems

Nandinbaatar Tsog, Saad Mubeen, Mikael Sjödin, Fredrik Bruhn
In the Transactions of the Japan Society for Aeronautical and Space Sciences,
Aerospace Technology Japan, ATJ 2020

Abstract

On-board data processing is one of the prior on-orbit activities that improves the performance capability of in-orbit space systems such as deep-space exploration, earth and atmospheric observation satellites, and CubeSat constellations. However, on-board data processing encounters higher energy consumption compared to traditional on-board space systems. This is because the traditional space systems employ simple processing units such as single-core microprocessors as the systems do not require heavy data processing. Moreover, solving the radiation hardness problem is crucial in space, and adopting a new processing unit is challenging.

In this paper, we consider a Graphics Processing Unit (GPU) accelerated in-orbit space system for on-board data processing. According to prior works, there exist radiation-tolerant GPU, and the computing capability of systems is improved by using heterogeneous computing method. We conduct experimental observations of energy consumption and computing potential using this heterogeneous computing method in our GPU accelerated in-orbit space systems. The results show that the proper use of GPU increases computing potential with 10-140 times and consumes between 8-130 times less energy. Furthermore, the entire task system consumes 10-65% of less energy compared to the traditional use of processing units.

Key Words: On-board data processing, Heterogeneous computing, Energy efficiency, GPU accelerated on-board computer

Nomenclature

C	:	worst case execution time (WCET), sec
T	:	period of task, sec
D	:	deadline of task, sec
R	:	response time (RT), sec
t	:	time instance, sec
E	:	consumed energy, Joule
P	:	consumed power, Watt

7.1 Introduction

In the space community, technological advances make it possible to work on a new challenge for on-orbit activities [1] including in-orbit servicing and in-situ experiments. On-board data processing is one of the prior on-orbit activities that improves the performance capability of in-orbit space systems such as deep-space exploration, earth and atmospheric observation satellites, and CubeSat constellations. We consider that the advanced on-board data processing solves the current communication limitation which is low-speed connections between satellites and ground stations with limited access time intervals. Furthermore, there exist Size, Weight, and Power (SWaP) and radiation limitations for space systems as well as on-board data processing. Due to these limitations, the traditional and small scale space systems employ simple processing units such as micro-controllers or a single-core processor even though the systems end up with limited on-board processing capabilities in orbit. The rapid development of technology makes advanced on-board data processing possible for small scale space systems using heterogeneous processing units that meet the requirements of size and weight limitations. Moreover, there exist many radiation hardened and/or tolerant processing units including Field Programmable Gate Arrays (FPGA), Digital Signal Processor (DSP) and Graphics Processing Unit (GPU) [2]. However, these processing units consume more energy [3]. In addition, the use of GPUs in the context of space is not well studied yet, due to the prior concern that GPUs are not suitable for the radiation-hardened environments. Therefore, in this paper, we consider a trade-off between computing power and energy consumption focusing on the entire task set with different use scenarios in GPU accelerated systems.

The interest of using heterogeneous computing in real-time and low-end embedded systems is increasing along with advanced on-board processing such as machine learning and computer vision algorithms. However, in real-time and low-end embedded systems, heterogeneous processing units are less-studied compared to single- and multi-core processing units, although, heterogeneous computing is well-known in High-Performance Computing (HPC), especially in supercomputers [4, 5]. The main reasons that hinder the usage of heterogeneous processing units in embedded systems are difficulties of parallel programming and complexity of heterogeneous systems. In order to address these problems, some industry vendors (AMD, ARM, Imagination, MediaTek, Qualcomm, and Samsung) established HSAFoundation [6] which has proposed a new standard, the Heterogeneous System Architecture (HSA), for the advancement of heterogeneous computing. In this paper, we conduct experimental observations of HSA compliant GPU accelerated on-board processing platforms using heterogeneous computing methods introduced in prior works [5, 7]. These platforms are commercialized by Unibap AB[1] with flight heritage and selected by NASA for high-performance on-board data processing for the "HyTI" thermal hyperspectral mission [8].

7.1.1 Contributions

The overall goal of our research is to develop a real-time system which could provide more computing potential to its tasks under energy limited conditions. This work is part of understanding suitable mapping from heterogeneous processors to tasks under limited energy budget. Prior works [7, 5, 9] report that the balanced use of heterogeneous processors improves the schedulability of the task sets in real-time systems when tasks are allowed to choose to run on different processors in different instances. Hence, our contribution in this paper is to conduct observations of energy consumption in GPU accelerated real-time systems while using the mapping method for the balanced use of heterogeneous processors. These observations provide us the fundamental understanding to perform the dynamic allocation of tasks to the heterogeneous processors under limited energy budget.

7.1.2 Organization

In the rest of this paper, we provide needed related work in Section 7.2. Section 7.3 presents detailed explanations about real-time systems, heterogeneous computing as well as advanced applications in satellite. A description of our

[1] https://unibap.com/

system model is discussed in Section 7.4. Section 7.5 reports experimental evaluation. Lastly, we conclude in Section 7.6.

7.2 Related work

In high performance computing, the research of heterogeneous processors and heterogeneous computing is very active [4]. Especially, in supercomputers, the impact of GPU is indispensable. However, the balanced use of GPU and Central Processing Unit (CPU) is significant, since not all the applications are suitable for parallelism [5]. The nature of Open Computing Language (OpenCL) [10, 5] makes heterogeneous computing easier as it is possible to prepare the different kernels on the different devices. Furthermore, heterogeneous computing is considered as part of distributed computing in sense of distributing the data/kernels to the distributed computing resources when applications use data-parallelism. However, satellites as being low-end embedded system applications need to perform under limited budgets of the different resources (location, SWaP); therefore, considering the distributed computing resources is challenging in the satellite. Moreover, the research of heterogeneous computing in real-time embedded systems is less studied compared to high performance computing.

There exist several approaches to utilize GPU in real-time systems. Shinpei et al. introduced TimeGraph[11], Responsive GPGPU Execution Model (RGEM)[12] and Gdev[13] along with zero-copy Input/Output (I/O) processing for low-latency GPU computing[14]. Furthermore, the works of Elliott et al. [15, 16] and Kim et al. [17, 18] consider worst-case timing behavior in GPU accelerated real-time systems. Most of these works consider compensating the limitation of early existing GPU hardware and device drivers such as a zero-copy technique for accelerators' memory and splitting tasks into smaller chunks for allowing preemption. However, these limitations are considered to be solved by new technologies such as unified memory, zero-copy and preemption support in Compute Unified Device Architecture (CUDA)[19] and Heterogeneous System Architecture (HSA)[20, 3, 21].

There are several works that have focused on modeling sequential and parallel tasks such as fork-join[22, 23] and Directed Acyclic Graph (DAG)[24, 25]. Recently, Baruah[26] introduced $if\text{-}then\text{-}else$ concept using conditional DAG task modeling, which is useful for the heterogeneous computing. The topology of this model is considered in this study.

In order to maintain a sustainable system in space, energy efficiency is the crucial factor that should be considered. There are many techniques used to improve energy efficiency[4] such as workload partitioning based techniques

[27], Dynamic Voltage and Frequency Scaling (DVFS) based techniques [28], and resource scaling based techniques. The combination of these techniques has also considered to save energy efficiency [29]. Our experiments in this paper focus more on the energy consumption of the entire system compared to specific tasks, since the power budget for the entire system is most important in the low-end embedded systems.

Employing heterogeneous processors in On-Board Computer (OBC) of a satellite is common when the scale of the satellite size is larger. For example, FPGA accelerated on-board computers are well known in satellites, as FPGAs are robust in the radiation-hardened environments. Since FPGAs are good for image and video processing, they are considered for on-board processing in an advanced imaging system [30], Digital Video Broadcasting - Satellite - Second Generation (DVB-S2) transport stream [31] and real-time cloud detection [32]. On the other hand, the use of GPUs in the context of space was not appreciated, due to the prior concern of GPUs for the radiation-hardened environments. Recently, GPUs are being considered more and more in the on-board computer is increasing [33, 2]. In this paper, we conduct experimental observations of utilizing GPU in the context of space.

7.3 Background

The advanced on-board data processing should be predictable in order to make a decision in orbit, while it is considered as a way to solve the limitation of communication between the satellite and the ground station. To consider a predictable system, we introduce the background knowledge of real-time systems in this section. Then, we present how the heterogeneous computing techniques are implemented in the current state of the art environments. Furthermore, we discuss the use of advanced applications in satellite in this section.

7.3.1 Real-time system

A real-time system is a system that reacts to external events. The system executes a function based on the external events and returns a response within a finite and required time. Therefore, not only the accuracy of the result, but also the timeliness is a crucial factor for the accuracy of the system.

The real-time system can be divided into a hard, firm and soft [34] real-time system from perspective of the timing constraints (see Figure 7.1). The hard real-time system must pass all specified timing constraints. If the system misses a constraint (e.g., a deadline) once, it results in failure leading to a

fatality and/or big financial or environmental damage. Therefore, hard real-time systems are often considered to be safety critical. In a soft real-time system, one or more deadline misses may be tolerated at the cost of lower quality of service. A firm real-time system is between hard and soft real-time systems.

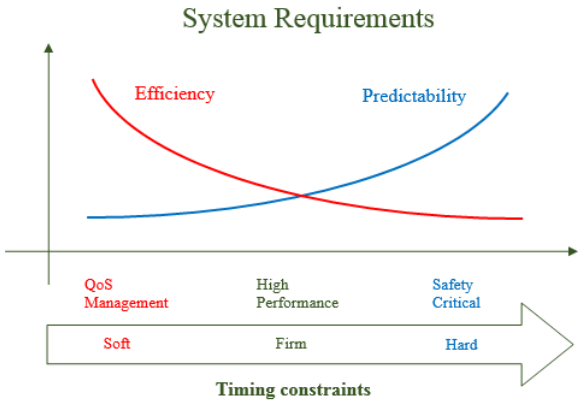


Figure 7.1: A real-time system requirements.[2]

7.3.2 Heterogeneous computing

In regards to parallel computation, technology developments that have been pursued actively cover many environments such as operating systems, programming languages/libraries, heterogeneous processing units and so on. Here we look through four programming languages which are pushing the heterogeneous computation research a lot.

Open Multi-Processing (OpenMP) [35], a specification implementing Application Programming Interface (API), is a well-known candidate when it comes to parallel computation and consists of compilers directives, runtime library routines, tool support and environment variables used in Fortran and C/C++ programs. OpenMP allows a program to run its parallel part regardless of whether it is on a host device or target devices. Regardless of how an executable is assigned to the processors, the host device is set as a default/spare processor and is possible to run the executable implicitly when the assigned target device is not able to run it. In other words, a parallel part of programs has a heterogeneous variety of the execution contexts on processors.

[2] http://www.artist-embedded.org/docs/Events/2008/RT-Kernels/SLIDES/s1-Intro.pdf

OpenCL is an open and royalty-free standard for parallel programming in heterogeneous systems including smartphones, personal computers, servers and embedded systems. In OpenCL[36], computing systems are considered as a collection of a number of computing devices which consist of a host processor (host device in OpenMP) and accelerators (target devices in OpenMP). By simply using *clCreateContextFromType* function together with conditional statements like *if*, programmers can develop a heterogeneous nature of executions explicitly.

CUDA is a (Nvidia's GPU centered) parallel computing platform and a heterogeneous programming model. CUDA consists of a host and devices which stand for CPU and Nvidia's GPUs, respectively. In CUDA [37, 38], three qualifiers/space-specifiers (*__global__*, *__device__*, and *__host__*) are prepared to run code regardless of it is on host or devices. The space-specifiers allow programmers to write executions explicitly with a heterogeneous nature.

In order to gain computational performance using GPU in systems, Microsoft implemented a native programming model and open specification called "C++ Accelerated Massive Parallelism (C++ AMP)" which extends to programming language C++ and its runtime library [39]. Moreover, C++ AMP is supported by HSA using its intermediate language Heterogeneous System Architecture Intermediate Language (HSAIL). HSA allows virtual shared memory between different devices such as host (CPU) and target (e.g. GPU, DSP). Similar to OpenMP, C++ AMP runs an executable implicitly on a host device when it is assigned to target device which is not able to run the executable at the same time.

7.3.3 Advanced applications in satellite

Satellite image analysis presents a fertile ground for applying cutting edge computer vision algorithms. In contrast to other fields of application of computer vision, such as Advanced Driver Assistance Systems (ADAS), satellite images are quasi-static, in the time-scale defined by the image acquisition frequency: the satellite does not move very fast (if at all) in relation to the imaged landscape, weather conditions such as cloud formations do not change rapidly, and neither do the lighting conditions. Nonetheless, for a variety of applications, we still need to be able to compensate for all these factors, and deduce a normalized image where further inference can be performed. As it is common in the computer vision/machine learning space, it is beneficial to know in advance what questions one wishes to answer. Possible interesting questions include: is a forest on fire, and if yes, how is it evolving over time? Or, is there a hurricane system developing? Or, how fast is ice melting in Antarctica? Or,

how fast is traffic flowing in highway I-90?

Cloud identification is best viewed as deducing a cloud density distribution over the image to accommodate for the varying degree of apparent cloud thickness. In this scheme, a "cloudness" value of 1 would be interpreted as perfect confidence of complete obstruction of the ground by clouds (per pixel). Similarly, a "cloudness" value of 0 would be interpreted as perfect confidence of no obstruction. There are multiple ways to define such a model, while the problem is of some complexity, in the following sense: clouds over snow appear significantly different compared to clouds over an ocean, or over a city during the day, or during the night. One can condition a cloudness model over the various relevant backgrounds, and train either a generative model that will generate clouds and apply them on various backgrounds, or directly train a discriminative model that will deduce the cloudness distribution. There is merit in both approaches, however, in keeping with the current state of the art, it is beneficial to train a deep network, to automatically discover the salient feature maps. There are two approaches in training such a network: one approach would stream the data to a ground data-center, which would combine imagery from multiple satellites. Such an approach would be the most fruitful, as it could be enhanced by user-assisted classification. There are multiple machine learning frameworks that can accomplish this, such as Tensorflow[3], Torch[4], Caffe[5], and Microsoft Cognitive Toolkit (CNTK)[6]. Of course, one could also employ unsupervised learning approaches, such as a (deep) auto-encoder scheme. Having a trained model, one can execute it on the satellite's GPU, and produce a scene classification. The forward problem consists of a cascade of convolution filters, activation functions, subsampling, and normalization. One would execute a forward pass on multiple and possibly overlapping regions of interest, for local scene classification. Such a problem is well suited for GPU acceleration. If the training takes into account the various variability factors (terrain kinds, atmospheric conditions, light conditions), the classification will also succeed in classifying the scene according to these factors, for example deducing that the scene represents a city at night with clouds.

Having performed a classification on the scene, one can then have a solid ground in performing further analysis: for example, by knowing what features in the scene are persistent, as opposed to transient noise (such as a cloud), one can select robust keypoints (or robust regions of interest), for image registration (either based on keypoints, or based on functional minimization). Such

[3]https://www.tensorflow.org/

[4]http://torch.ch/

[5]https://caffe.berkeleyvision.org/

[6]https://github.com/Microsoft/CNTK

robust registration would be beneficial for creation of panoramas, or for enhancing image quality by combining multiple images of the same scene (super-resolution). Conversely, one can apply tracking algorithms to the transient features, for example following a cloud or a car, or set of cars, using correlation tracking. Depending on image resolution, it may be beneficial to apply pre-processing algorithms on the image, such as image sharpening (e.g. via unsharp mask with local contrast enhancement, or anisotropic heat diffusion).

7.4 System Model

We consider a task model which is described by Fork-Join task model. As shown in Fig. 7.2, the parallel segment (starts with a Fork and ends with a Join) of tasks could be executed in two manners, parallel and sequential [9]. Parallel execution could be performed on GPU, multi-core CPUs, or single CPU using parallelization techniques such as Single Instruction, Multiple Data (SIMD), multithreading, etc. Sequential execution is executed on CPU sequentially.

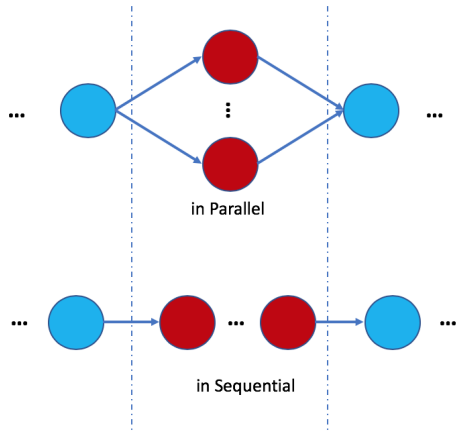


Figure 7.2: Execution manner of parallel segment of Fork-Join task model

In order to study a trade-off between computing power and energy consumption, we consider a task set Γ, which consists of n independent periodic tasks $\{\tau_1, .., \tau_n\}$ expressed with the introduced task model. Each task τ_i has a period T_i, deadline D_i, and worst case execution time (WCET) C_i. The response time (RT) R_i of task τ_i is measured by experimental observations in this paper.

We consider that the system consists of two different processing units such

as CPU and GPU. The system energy consumption can be calculated with either $E_{system} = E_{CPU} + E_{GPU} + E_{other}$ or

$$E_{system} = \sum_{1 \leq t \leq max(R_i), 1 \leq i \leq n} P_{system}(t) * \Delta t$$

Here, E_{system} is the system's energy consumption. E_{CPU}, E_{GPU}, and E_{other} are the energy consumptions of CPU, GPU, and other peripherals, respectively. $P_{system}(t)$ is power consumption of the system at timing instance of t. Δt is unit value of time sample.

Algorithm 2 Algorithm of the 2D Anisotropic Diffusion.

* Initialise the variables; num_iter, $option$, $kappa$ and $lambda$.
* Set an initial condition of Partial Differential Equation (PDE); $diff\_im$.
* Set step for all directions. 1 pixel for horizontal: $d[x]$ and vertical: $d[y]$, sqrt(2) pixels for diagonal: $d[d]$.
* Define 2D convolution masks - finite differences.
* Main calculation of Anisotropic diffusion.Looping given number of iterations

for num_iter **do**

* Calculate finite differences nabla[] for all directions N, S, W, E, NE, SE, SW and NW, where N, S, W, E describe north, south, west, and east, respectively.
* Calculate coefficients for all directions:
– Choose a diffusion function from 2 original functions.
– if $option$ == 1
Calculate $c[] = exp(-(nabla[]/kappa)^2)$ for all 8 directions
– if $option$ == 2
Calculate $c[] = 1/(1 + (nabla[]/kappa)^2)$ for all 8 directions
* A solution for Discrete PDE
– $diff\_im = diff\_im + lambda * sum\{c[] * nabla[]/(d[]^2)\}$

end for

7.5 Experimental design

In this section, we introduce algorithms and discuss the evaluation of the energy consumption and computing potential of the task set which consists of these algorithms.

7.5.1 Algorithms

In this paper, we consider two on-board algorithms, namely Anisotropic Diffusion [40] and Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics (LULESH) [41]. The different combinations of the algorithms are used in the different purposes of the experiments.

Anisotropic Diffusion

We perform Anisotropic Diffusion algorithm to evaluate the on-board computer processing since this algorithm is used to sharpen images. As we mentioned in Section 7.3.3, sharpening the satellite images and detecting objects such as clouds and forest fires from the satellite images are significantly useful. The pseudo code of the Anisotropic Diffusion algorithm is shown in Alg. 2, as we have ported the 2D Anisotropic Diffusion code from MATLAB to C++ AMP, OpenCL and OpenMP in order to execute the application on HSA compliant platform. In this study, we only deal with the code, since the quality of Anisotropic Diffusion is well-known from the previous studies [40, 42, 43].

Table 7.1: Detailed information about the test machines.

Test Machine	Type	Product Specification	Clock	Cores / compute units	Energy consumption (Watt)
A10	CPU	A10-8700P APU, Excavator	1800MHz	4	12-35
	iGPU	Radeon™ R6, GCN Gen3	800MHz	6	
R&R	CPU	Ryzen™ 7 1800x, Zen	4GHz	8	95
	dGPU	Radeon™ R9 nano, GCN Gen3	1GHz	64	175

LULESH

LULESH is created as a result of the project, The Shock Hydrodynamics Challenge Problem, which is originally defined and implemented by Lawrence Livermore National Laboratory (LLNL) as one of five challenge problems in Defense Advanced Research Projects Agency (DARPA)'s Ubiquitous High Performance Computing (UHPC) program. LULESH is a highly simplified shock hydro application in order to solve only a simple Sedov blast problems [44]. Modeling hydrodynamics is significant in computer simulations as it is used to understand the motion of materials relative with each other under the force. Furthermore, these kind of simulations are preferable to use the parallel computing. In order to achieve parallelism, LLNL provides an open-source version

of LULESH[7][8], which is ported to the different environments such as MPI, OpenMP, OpenCL and C++AMP.

7.5.2 Testbeds

Test platforms Two test machines, A10 and R&R, are used for this experiment. A10 is Acer's laptop that is maintained with AMD A10-8700P Accelerated Processing Unit (APU), which consists of 4 core CPUs and 6 compute unit GPUs in a chip. APU is AMD's product name of a new type of processing unit, which integrates CPU and GPU in a die. APU is normally termed as "integrated GPU". R&R is a custom made desktop computer and consists of AMD Ryzen™ 7 1800x8 core CPUs and AMD Radeon™ R9 nano GPU. More details are shown in Table 7.1. As the test machines are general-purpose computers, the range of the energy consumption differs from the embedded systems, especially R&R. Both A10 and R&R are used for experimental observations 1 and 2, while only A10 is used for observation 3.

Test application 1 This application performs Anisotropic Diffusion algorithm in order to measure a computation time on the following three combination of the accelerators; HSACalc, CPUCalc, and OMPCalc. HSACalc is about the computation of the algorithm using GPU with the HSA extension. CPUCalc and OMPCalc are the computation of the algorithm on single-core CPU and multicore CPUs, respectively. OpenMP's loop parallel technique is used in OMPCalc for multicores. The aim of this test application is to confirm the computation time improvements of GPU/HSA instead of a single core CPU.

Test application 2 There are three applications which run Anisotropic Diffusion algorithm in three different programming manners independently. The intention of this experiment is to monitor the energy consumptions of the different compute units in different programming manners.

Test application 3 This application performs two algorithms (Anisotropic Diffusion and LULESH) concurrently with different sets. The intention of this experiment is to monitor how the energy consumption of the system changes with respect to the different settings of tasks.

7.5.3 Experimental observations

Observation 1: Compiler vs Computing potential. First, we consider the relation between computing potential with respect to the different compiler ver-

[7] https://github.com/LLNL/LULESH

[8] https://github.com/AMDComputeLibraries/ComputeApps

sions. Test application 1 is compiled by three different versions (GCC5.4.0, GCC6.2.1[9] and GCC 7.1.0[10]) of GCC compiler toegther with 2 different options, "non-optimised" and "optimised". "Non-optimised" is compiled with "-O0" flag, and "optimised" is compiled with "-O3"[11] flag. Each measurement is performed 100 times continuously.

Observation 2: Energy consumption vs Programming manner. Then, we conduct an experiment about energy consumption of test application 2 implemented in different programming manners (using HSA for GPU, normal sequential execution on CPU, using OpenMP for parallelization on CPU). While the experiments of "optimised" and "non-optimised" versions of the applications are conducted in "Observation 1", we consider the worst case scenario in this observation. Hence, we use only non-optimised versions of the applications in this observation.

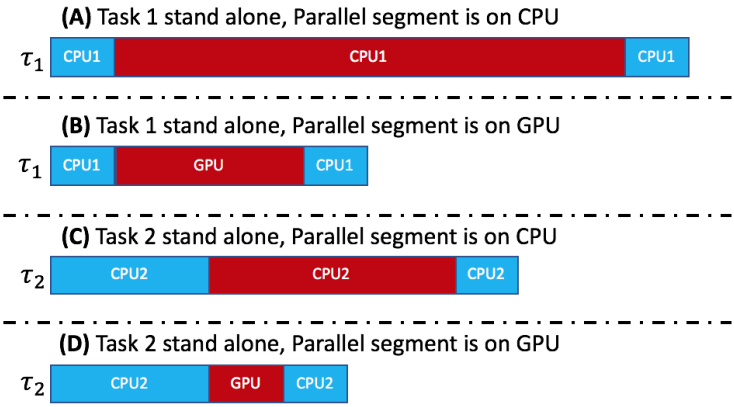


Figure 7.3: Execution manners of a stand-alone parallel segment in terms of considering worst case execution time (WCET). Blue bars express sequential segments of tasks that should be executed sequentially only. Red bars are parallel segments of tasks. Parallel segments could be executed either in sequential or in parallel manner. The length of each task/bar describes its worst case execution time. The cases (A) and (B) are the execution manners of parallel segment of Task 1 (τ_1) in sequential and parallel, respectively. Similarly, (C) and (D) represent the execution manners for Task 2 (τ_2).

[9]Untrusted PPA:*ppa:jonathonf/gcc-6.2*

[10]Untrusted PPA: *ppa:jonathonf/gcc-7.1*

[11]Combining the "-O3" flag with the following machine architecture specific flags is possible; "-march=bdver4" and "-march=znver1". However, we consider only "-O3" flag in this paper.

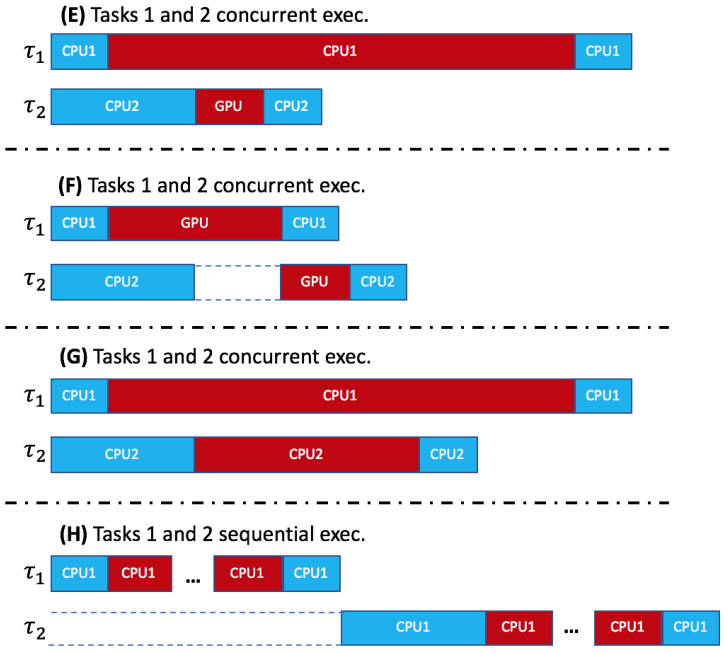


Figure 7.4: Execution manners of concurrent parallel segments in terms of considering response time (RT). The combinations of the tasks, Task 1 (τ_1) and Task 2 (τ_2), described in Fig. 3 are considered. White space bordered with dot-lines expresses no execution of tasks, e.g., τ_2 in cases (F) and (H). Three dots means that the pointed parallel segment has been shortened compared to its actual execution. In cases (E) and (F), the parallel segment of τ_1 is executed either in sequentially or parallel, while the parallel segment of τ_2 is executed in parallel manner only. In case (G), Tasks τ_1 and τ_2 are executed on CPU1 and CPU2, respectively. Both τ_1 and τ_2 are allocated to CPU1 only in case (H).

Observation 3: Energy consumption vs Execution manner. Finally, we consider experiments in which the tasks in the task set are allocated to the different processing units. In order to generate the worst case scenario, we consider non-optimised codes on A10 machine in this observation. By conducting these experiments, we can monitor how the balanced use of the processing units affects the energy consumption of the systems. The allocations of the tasks are illustrated in Figs 7.3 and 7.4. We express sequential and parallel segments with blue and red bars, respectively. Text inside the bars describe where this segment should be allocated. For example, in Fig 7.3-(A), the parallel segment of task τ_1 is allocated to CPU1. On the other hand, in Fig7.3-(B), we

can see that this parallel segment is allocated to GPU. Stand alone executions of the tasks are illustrated in Fig. 7.3. Allocations of concurrent executions of tasks τ_1 and τ_2 are shown in Fig. 7.4. Here, we consider that τ_1 and τ_2 implement LULESH and Anisotropic Diffusion algorithms, respectively.

7.5.4 Evaluation and Results

Observation 1: The results of running the test application 1 are shown in Tables 7.2, 7.3, 7.4, 7.5, 7.6 and 7.7. In this observation, we consider the combinations of two machines, A10 and R&R, and three accelerators, HSACalc, CPUCalc, and OMPCalc, named as A10 HSACalc, A10 CPUCalc, A10 OMPCalc, R&R HSACalc, R&R CPUCalc, and R&R OMPCalc. In non-optimised experiments, we can see that the computation times are in the following order: R&R OMPCalc, A10 OMPCalc, R&R CPUCalc and A10 CPUCalc, from the smallest to the largest respectively. This result is obvious since we have used Ryzen™ 7 1800x CPU which is one of the best CPUs in the market now. However, when we turn a spotlight to both A10 and R&R HSACalc, HSACalc shows between 122 to 152 times faster than the calculation which uses 1 core CPU with non-optimised version. This ratio improves in the case of 1 core CPU with optimised version, however, HSACalc shows still between 11 to 22 times faster computation time than 1 core CPU. Moreover, A10 HSACalc shows the best computation time among others, even better or similar R&R HSACalc. As we explained in Table 7.1, Ryzen™ CPU and discrete R9 nano GPU are connected through PCIe 3.0 in R&R machine. In A10-8700P APU, CPU and GPU are located in the same silicon with coherent fabric connection. Hence, A10-8700P APU is gaining benefits of HSA more than R&R for this particular workload. In general, a high end discrete GPU has significantly more compute cores than an integrated GPU. While data needs to be physically transferred to the discrete GPU over a typically slower bus, once the transfer is performed (optimally employing the multiple concurrent asynchronous Direct Memory Access (DMA) engines typically available), the data is available on the discrete GPU over a very fast memory (e.g. DDR5). Therefore the relative performance of a discrete GPU over an integrated GPU is workload dependent. The more the data that need to be densely processed, the higher the desired frame-rate, and the more processing steps they will undergo, the more likely it is that a discrete GPU will outperform an integrated GPU. However, as argued in this paper, they will both typically significantly outperform even a very powerful CPU.

As we mentioned in the previous section, we have used ”-O3” flag for the compiler level optimisations. Moreover, there is a machine architecture spe-

cific optimisation flag "-march=znver1" for Ryzen™. However, this flag is not available to optimise with GCC 5.4 compiler. Therefore, we have focused only on "-O3" flag in this paper. We confirm 6.5-12.7 times faster improvements for the optimised versions of both A10 and R&R CPUCalc compared to the non-optimised versions. Moreover, about similar times faster improvements have confirmed for the optimised versions of both A10 and R&R OMPCalcs compared to the non-optimised versions. In addition, the optimised R&R CPUCalc calculates the similar computation time as A10 OMPCalc.

Table 7.2: Anisotropic Diffusion, gcc version 5.4.0 20160609, non optimised

Computation time of Anisotropic Diffusion						
	A10 HSACalc	A10 CPUCalc	A10 OMPCalc	R&R HSACalc	R&R CPUCalc	R&R OMPCalc
max (msec)	2.692	128.843	50.631	0.913	94.292	36.225
min (msec)	0.487	112.960	43.897	0.556	89.341	27.726
avg (msec)	0.889	116.663	47.223	0.679	91.978	30.372
ratio[12]	131.265	1	2.470	135.468	1	3.028

Table 7.3: Anisotropic Diffusion, gcc version 5.4.0 20160609, optimised

Computation time of Anisotropic Diffusion						
	A10 HSACalc	A10 CPUCalc	A10 OMPCalc	R&R HSACalc	R&R CPUCalc	R&R OMPCalc
max (msec)	2.580	22.579	14.874	0.914	11.983	4.869
min (msec)	0.516	16.206	6.674	0.567	7.764	2.085
avg (msec)	0.923	17.819	8.752	0.658	8.247	2.933
ratio	19.307	1	2.036	12.532	1	2.812

Observation 2: The energy consumption results of the test application 2 for the different environments, HSACalc, CPUCalc and OMPCalc, with optimised and non-optimised are shown in Figure 7.5. The preparation, initializing variables and loading images, part of the entire calculation is marked with orange colour, and the execution which is the essential calculation of the algorithm is marked with blue colour. Here, we consider the total energy

[12]The ratio of the average values to the average value of CPUCalc. For example, $\frac{avg(A10CPUCalc)}{avg(A10HSACalc)}$ or $\frac{avg(R\&RCPUCalc)}{avg(R\&ROMPCalc)}$.

Table 7.4: Anisotropic Diffusion, gcc version 6.2.1 20161215, non optimised

Computation time of Anisotropic Diffusion						
	A10 HSACalc	A10 CPUCalc	A10 OMPCalc	R&R HSACalc	R&R CPUCalc	R&R OMPCalc
max (msec)	2.796	123.160	48.415	0.903	94.328	35.914
min (msec)	0.485	112.515	42.630	0.545	88.889	24.633
avg (msec)	0.963	117.759	45.092	0.608	91.380	26.593
ratio	122.302	1	2.612	150.408	1	3.436

Table 7.5: Anisotropic Diffusion, gcc version 6.2.1 20161215, optimised

Computation time of Anisotropic Diffusion						
	A10 HSACalc	A10 CPUCalc	A10 OMPCalc	R&R HSACalc	R&R CPUCalc	R&R OMPCalc
max (msec)	2.703	22.354	13.924	0.837	11.694	5.696
min (msec)	0.485	15.885	6.572	0.552	7.963	2.199
avg (msec)	0.770	17.524	8.233	0.640	8.318	2.560
ratio	22.762	1	2.129	12.993	1	3.249

Table 7.6: Anisotropic Diffusion, gcc version 7.1.0, non optimised

Computation time of Anisotropic Diffusion						
	A10 HSACalc	A10 CPUCalc	A10 OMPCalc	R&R HSACalc	R&R CPUCalc	R&R OMPCalc
max (msec)	2.504	124.866	48.461	1.135	90.652	37.356
min (msec)	0.508	113.488	41.570	0.535	85.848	24.244
avg (msec)	0.782	119.108	45.348	0.684	88.440	26.210
ratio	152.290	1	2.627	129.330	1	3.374

consumption, E_{system}, as a summation of the energy consumption of the execution, $E_{execution}$, and preparation, $E_{preparation}$, i.e., $E_{system} = E_{execution} + E_{preparation}$. In case of HSACalc, the total energy consumption, E_{system}, of the system is 24.08 Joules and 17.46 Joules for non-optimised and optimised versions, respectively. In other words, the energy consumption of the execution ($E_{execution}$) for HSACalc is 2.24 and 3 Joules, and the energy consumption of the preparation ($E_{preparation}$) for HSACalc is 21.84 and 14.46

Table 7.7: Anisotropic Diffusion, gcc version 7.1.0, optimised

Computation time of Anisotropic Diffusion						
	A10 HSACalc	A10 CPUCalc	A10 OMPCalc	R&R HSACalc	R&R CPUCalc	R&R OMPCalc
max (msec)	2.673	18.415	12.675	0.899	10.425	4.189
min (msec)	0.489	14.304	5.597	0.538	6.459	1.873
avg (msec)	0.753	15.668	7.158	0.620	6.923	2.265
ratio	20.800	1	2.189	11.158	1	3.056

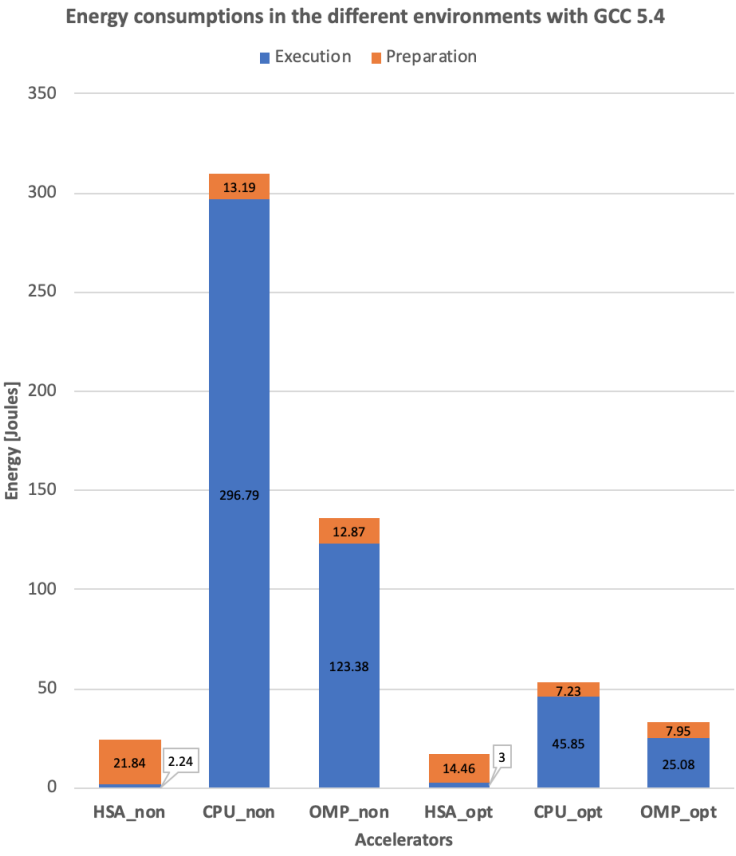


Figure 7.5: Comparison of energy consumption between different programming manners with optimised and non-optimised codes.

Joules. The energy consumption of the preparation part of CPUCalc and OMPCalc decrease by half of HSACalc to approximately 13 and 7.5 Joules for non-optimised and optimised versions, respectively. From Figure 7.5, we can see that the energy consumption of the experiment part in OMPCalc (123.38 Joules for non-optimised and 25.08 Joules for optimised) is around half of CPUCalc (296.79 Joules for non-optimised and 45.85 Joules for optimised). Hence, we see that the execution part ($E_{execution}$) of HSACalc uses between 15 ($= 45.85/3$) to 132($= 296.79/2.24$) times less energy consumption compared to the execution part of CPUCalc. Similar to the comparison of energy consumption regarding the execution parts of HSACalc and OMPCalc is between 8 to 55 times difference, which means HSACalc consumes that much less energy. In other words, adapting an HSA complaint GPU uses between 8 to 132 times less energy compared to the CPU cores.

Table 7.8: System's energy consumption (stand alone execution)

	(A)	(B)	(C)	(D)
Measured WCET of τ_1 (s)	15.21	9.1		
Measured WCET of τ_2 (s)			11.94	6.11
Energy consumption of the system (Joules)	136.81	88.82	107.14	60.81

Table 7.9: System's energy consumption (concurrent executions)

	(E)	(F)	(G)	(H)
Measured RT of τ_1 (s)	15.63	9.41	15.93	15.78
Measured RT of τ_2 (s)	6.36	11.62	12.64	32.1
Energy consumption of the system (Joules)	150.32	136.29	152.47	382.41

Observation 3: The results of different allocations of tasks τ_1 and τ_2 are illustrated in Tables 7.8 and 7.9. The result in Table 8 corresponds to the execution manner described in Figure 3, while the result in Table 9 relates to the execution manners in Figure 4. Since the stand alone execution manner of the tasks are explained in Figure 3, the results in (C) and (D) for tasks τ_1 and in (A) and (B) for tasks τ_2 are not available. As described in Figure 4, we consider an execution manner of concurrent execution of two tasks, hence, the response time (RT) will be the key in these measurements. In case of (F), we can see the system consumes less energy (at least 10% and up to 65%) compared to other allocations, although the RT of τ_2 gets almost two times longer ($R_2 = 11.62sec$) than the stand alone version, i.e, its worst case execution time (WCET). ($C_2 = 6.11sec$). The RT of τ_2 in (F) is shorter than the stand alone version ($C_2 = 11.94sec$) of τ_2 when the parallel segment is allocated to CPU sequentially. This means that the proper use of GPU shows better results

of both computing potential and energy efficiency.

The energy consumptions of the systems in the cases of (E)($E_{system} = 150.32 Joules$) and (G)($E_{system} = 152.47 Joules$) are close to each other. The parallel segment of task τ_1 is allocated to CPU in both cases. The difference here is that the parallel segment of task τ_2 is allocated to GPU in parallel and CPU sequentially. This means that we can choose the allocation of (E) in case GPU is idle. Otherwise, it is good to choose the allocation of (G) when GPU is busy with other tasks.

The allocation (H) shows longest RTs ($R_1 = 15.78 sec$ for τ_1 and $R_2 = 32.1 sec$ for τ_2) and consumes most energy ($E_{system} = 382.41 Joules$). However, we have to note that the system did not use the GPU in this case at all. Hence, we could say that the system has more space for GPU computation.

7.6 Conclusion

In this paper, we have focused on the energy consumption and computing potential of GPU accelerated in-orbit space systems. Further, both programming manner (how to compile a task) and executing manner (how to allocate a task) are considered in the experiments. From the experimental study, we have confirmed that the execution part of HSA compliant GPU computes the calculation between 10 to 140 times faster and consumes between 8 to 130 times less energy, compared to the execution part of CPU-based (including single- and multi-core processors) calculations. The use of GPU is supported even when we consider the entire system as allocation of the workload to GPU is most energy efficient compared to the other allocations. Therefore, we conclude that GPU can be a highly potential candidate in the on-board data processing of the small satellites.

For future work, we would like to continue developing a system with real-time GPU scheduler, which can dynamically allocate the tasks under limited power budget.

Acknowledgments

The work presented in this paper is supported by the Swedish Knowledge Foundation (KKS) through the research profile DPAC. We would like to express our sincere gratitude to Dr. Moris Behnam for sharing his great knowledge in real-time embedded systems. Further, this work is supported in part by a corporate scholarship of the TESO Corporation for Nandinbaatar Tsog.

The authors would also like to express our sincere gratitude to Dr. Harris Gasparakis, a computer vision expert at AMD, for his great knowledge in computer vision and HSA related areas. Dr. Gasparakis has helped us a great deal by providing an extensive amount of support whenever necessary, especially, sections 7.3 and 7.5 would not be improved without his advice and valuable discussions.

Bibliography

[1] T. Master, "Consortium for Execution of Rendezvous and Servicing Operations (CONFERS)." [Online]. Available: https://www.darpa.mil/program/consortium-for-execution-of-rendezvous-and-servicing-operations

[2] F. Bruhn, K. Brunberg, J. Hines, L. Asplund, and M. Norgren, "Introducing radiation tolerant heterogeneous computers for small satellites," in *2015 IEEE Aerospace Conference*. IEEE, 2015, pp. 1–10.

[3] N. Tsog, M. Behnam, M. Sjödin, and F. Bruhn, "Intelligent data processing using in-orbit advanced algorithms on heterogeneous system architecture," in *2018 IEEE Aerospace Conference*, March 2018, pp. 1–8.

[4] S. Mittal and J. S. Vetter, "A survey of cpu-gpu heterogeneous computing techniques," *ACM Computing Surveys (CSUR)*, vol. 47, no. 4, pp. 1–35, 2015.

[5] Y. Wen, Z. Wang, and M. F. O'boyle, "Smart multi-task scheduling for opencl programs on cpu/gpu heterogeneous platforms," in *2014 21st International conference on high performance computing (HiPC)*. IEEE, 2014, pp. 1–10.

[6] HSAFoundation, "HSA Foundation - ARM, AMD, Imagination, MediaTek, Qualcomm, Samsung, TI." [Online]. Available: http://www.hsafoundation.com

[7] N. Tsog, M. Sjödin, and F. Bruhn, "Using heterogeneous computing on gpu accelerated systems to advance on-board data processing," in *European Workshop on On-Board Data Processing 2019 OBDP2019, 25 Feb 2019, Amsterdam, Netherlands*, 2019.

[8] R. Wright, T. George, and ..., "Hyperspectral Thermal Imager (HyTI)." [Online]. Available: https://esto.nasa.gov/files/solicitations/INVEST_-17/ROSES2017_InVEST
_A49_awards.html#george

[9] N. Tsog, M. Becker, F. Bruhn, M. Behnam, and M. Sjödin, "Static allocation of parallel tasks to improve schedulability in cpu-gpu heterogeneous real-time systems," in *IECON 2019-45th Annual Conference of the IEEE Industrial Electronics Society*, vol. 1. IEEE, 2019, pp. 4516–4522.

[10] P. Czarnul and P. Rościszewski, "Optimization of execution time under power consumption constraints in a heterogeneous parallel system with gpus and cpus," in *International Conference on Distributed Computing and Networking*. Springer, 2014, pp. 66–80.

[11] S. Kato, K. Lakshmanan, R. Rajkumar, and Y. Ishikawa, "Timegraph: Gpu scheduling for real-time multi-tasking environments," in *2011 USENIX Annual Technical Conference (USENIX ATC 11)*, 2011, pp. 17–30.

[12] S. Kato, K. Lakshmanan, A. Kumar, M. Kelkar, Y. Ishikawa, and R. Rajkumar, "Rgem: A responsive gpgpu execution model for runtime engines," in *2011 IEEE 32nd Real-Time Systems Symposium*. IEEE, 2011, pp. 57–66.

[13] S. Kato, M. McThrow, C. Maltzahn, and S. Brandt, "Gdev: First-class gpu resource management in the operating system," in *2012 USENIX Annual Technical Conference (USENIX ATC 12)*, 2012, pp. 401–412.

[14] S. Kato, J. Aumiller, and S. Brandt, "Zero-copy i/o processing for low-latency gpu computing," in *Proceedings of the ACM/IEEE 4th International Conference on Cyber-Physical Systems*, 2013, pp. 170–178.

[15] G. A. Elliott, B. C. Ward, and J. H. Anderson, "Gpusync: A framework for real-time gpu management," in *2013 IEEE 34th Real-Time Systems Symposium*. IEEE, 2013, pp. 33–44.

[16] G. A. Elliott and J. H. Anderson, "Globally scheduled real-time multiprocessor systems with gpus," *Real-Time Systems*, vol. 48, no. 1, pp. 34–74, 2012.

[17] H. Kim, P. Patel, S. Wang, and R. R. Rajkumar, "A server-based approach for predictable gpu access control," in *2017 IEEE 23rd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2017, pp. 1–10.

[18] ——, "A server-based approach for predictable gpu access with improved analysis," *Journal of Systems Architecture*, vol. 88, pp. 97–109, 2018.

[19] M. Harris, ""Unified Memory for CUDA Beginners." June 19, 2017." available: https://devblogs.nvidia.com/unified-memory-cuda-beginners/ [Oct 16, 2018].

[20] HSA Foundation, “”Heterogeneous System Architecture.”,” available: http://www.hsafoundation.com/ [Oct 16, 2018].

[21] N. Tsog, M. Sjödin, and F. Bruhn, “Advancing on-board big data processing using heterogeneous system architecture,” in *ESA/CNES 4S Symposium 4S 2018, 28 May 2018, Sorrento, Italy*, 2018.

[22] C. Maia, M. Bertogna, L. Nogueira, and L. M. Pinho, “Response-time analysis of synchronous parallel tasks in multiprocessor systems,” in *Proceedings of the 22Nd International Conference on Real-Time Networks and Systems*, 2014, pp. 3–12.

[23] A. Saifullah, D. Ferry, J. Li, K. Agrawal, C. Lu, and C. D. Gill, “Parallel real-time scheduling of dags,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 12, pp. 3242–3252, 2014.

[24] J. D. Ullman, “Np-complete scheduling problems,” *Journal of Computer and System sciences*, vol. 10, no. 3, pp. 384–393, 1975.

[25] A. Melani, M. Bertogna, V. Bonifaci, A. Marchetti-Spaccamela, and G. C. Buttazzo, “Response-time analysis of conditional dag tasks in multiprocessor systems,” in *2015 27th Euromicro Conference on Real-Time Systems*. IEEE, 2015, pp. 211–221.

[26] S. Baruah, “Resource-efficient execution of conditional parallel real-time tasks,” in *European Conference on Parallel Processing*. Springer, 2018, pp. 218–231.

[27] Q. Liu and W. Luk, “Heterogeneous systems for energy efficient scientific computing,” in *International Symposium on Applied Reconfigurable Computing*. Springer, 2012, pp. 64–75.

[28] H. Aydin, R. Melhem, D. Mossé, and P. Mejía-Alvarez, “Power-aware scheduling for periodic real-time tasks,” *IEEE Transactions on computers*, vol. 53, no. 5, pp. 584–600, 2004.

[29] M. Rofouei, T. Stathopoulos, S. Ryffel, W. Kaiser, and M. Sarrafzadeh, “Energy-aware high performance computing with graphic processing units,” in *Workshop on power aware computing and system*, 2008.

[30] C. D. Norton, T. A. Werne, P. J. Pingree, and S. Geier, “An evaluation of the xilinx virtex-4 fpga for on-board processing in an advanced imaging system,” in *2009 IEEE Aerospace conference*. IEEE, 2009, pp. 1–9.

[31] R. Varsha, R. Arora, T. Ram, and A. Patel, "Design and implementation of dvb-s2 transport stream for onboard processing satellite," in *2015 19th International Symposium on VLSI Design and Test*. IEEE, 2015, pp. 1–6.

[32] J. A. Williams, A. S. Dawood, and S. J. Visser, "Fpga-based cloud detection for real-time onboard remote sensing," in *2002 IEEE International Conference on Field-Programmable Technology, 2002.(FPT). Proceedings.* IEEE, 2002, pp. 110–116.

[33] R. Davidson and C. P. Bridges, "Adaptive multispectral gpu accelerated architecture for earth observation satellites," in *2016 IEEE International Conference on Imaging Systems and Techniques (IST)*. IEEE, 2016, pp. 117–122.

[34] G. C. Buttazzo, *Hard real-time computing systems: predictable scheduling algorithms and applications*. Springer Science & Business Media, 2011, vol. 24.

[35] A. OpenMP, "Openmp application programming interface version 4.5," *OpenMP Architecture Review Board*, 2015.

[36] K. O. W. Group *et al.*, "The opencl specification, version: 1.2, document revision: 19.(nov. 2012)," *URL: https://www. khronos. org/registry/OpenCL/specs/opencl-1.2. pdf*, 2012.

[37] NVIDIA, "NVIDIA CUDA C Programming Guide, Version 4.2, April 16, 2012." [Online]. Available: https://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf

[38] ——, "CUDA C PROGRAMMING GUIDE, Version PG-02829-001_v9.2, Design Guide, August 2018." [Online]. Available: https://docs.nvidia.com/cuda/archive/9.2/pdf/CUDA_C_Programming_Guide.pdf

[39] Microsoft, "C++ AMP : Language and Programming Model, Version 1.0, August 2012." [Online]. Available: https://download.microsoft.com/download/4/0/E/40EA02D8-23A7-4BD2-AD3A-0BFFFB640F28/CppAMPLanguageAndProgrammingModel.pdf

[40] P. Perona and J. Malik, "Scale-space and edge detection using anisotropic diffusion," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 12, no. 7, pp. 629–639, 1990.

[41] L. L. N. Laboratory, "LULESH." [Online]. Available: https://computation.llnl.gov/projects/co-design/lulesh (accessed April 21, 2019)

[42] A. Schwarzkopf, T. Kalbe, C. Bajaj, A. Kuijper, and M. Goesele, "Volumetric nonlinear anisotropic diffusion on gpus," in *International Conference on Scale Space and Variational Methods in Computer Vision*. Springer, 2011, pp. 62–73.

[43] D. Lopes, "A set of filters that perform 1D, 2D and 3D conventional anisotropic diffusion, 2007." [Online]. Available: http://se.mathworks.com/matlabcentral/fileexchange/14995-anisotropic-diffusion–perona—malik- (accessed April 22, 2019)

[44] L. Sedov, "Similarity and dimensional methods in mechanics [in russian]," 1957.

Chapter 8

Paper C
Enabling Radiation Tolerant Heterogeneous GPU-based Onboard Data Processing in Space

Fredrik C. Bruhn, Nandinbaatar Tsog, Fabian Kunkel, Oskar Flordal, Ian Troxel
In the CEAS Space Journal, CEAS 2020

Abstract

The last decade has seen a dramatic increase in small satellite missions for commercial, public, and government intelligence applications. Given the rapid commercialization of constellation-driven services in Earth Observation, situational domain awareness, communications including machine-to-machine interface, exploration etc., small satellites represent an enabling technology for a large growth market generating truly Big Data. Examples of modern sensors that can generate very large amounts of data are optical sensing, hyperspectral, Synthetic Aperture Radar (SAR), and Infrared imaging. Traditional handling and downloading of Big Data from space requires a large onboard mass storage and high bandwidth downlink with a trend towards optical links. Many missions and applications can benefit significantly from onboard cloud computing similarly to Earth-based cloud services. Hence, enabling space systems to provide near real-time data and enable low latency distribution of critical and time sensitive information to users. In addition, the downlink capability can be more effectively utilized by applying more onboard processing to reduce the data and create high value information products. This paper discusses current implementations and roadmap for leveraging high performance computing tools and methods on small satellites with radiation tolerant hardware. This includes runtime analysis with benchmarks of convolutional neural networks and matrix multiplications using industry standard tools (e.g., TensorFlow and PlaidML). In addition, a ½ CubeSat volume unit (0.5U) ($10 \times 10 \times 5$ cm^3) cloud computing solution, called SpaceCloud™ iX5100 based on AMD 28 nm APU technology is presented as an example of heterogeneous computer solution. An evaluation of the AMD 14 nm Ryzen APU is presented as a candidate for future advanced onboard processing for space vehicles.

8.1 Introduction

There are numerous studies and argumentation for increased onboard autonomy and data information processing to provide more efficient use of the relatively limited communication link bandwidth on small satellites [1, 2, 3]. Expanding on the needs of intelligent processing, it is especially relevant to study the rapidly evolving field Earth Observation driven by advances in sensor technologies. ESA's Φ-lab at the ESRIN facility has led several workshops in the context of artificial intelligence (AI) for Earth Observation (AI4EO) and written a European AI research agenda [4]. The agenda identifies a range of challenges and opportunities for ensuring European pooling of resources, talent supply, digital environment for rapid prototyping, and development of solutions to capture the opportunities. The landscape formed around the transformative AI technology is today dominated by United States and China. ESA have formulated several candidate projects in the mid-technology readiness level (TRL) range within the General Study Technology Programme (GSTP) Element 1 "Develop" AI 2019 compendium [5]. These candidate projects are of strategic importance to current and future space systems and space exploration and cover both data exploitation and operations. The proposed developments are categorized in these areas:

- Smart payload data
- AI in data exploitation
- AI in operations
- Guidance, navigation, and control
- Edge/onboard AI

This paper covers architectural and software aspects of Edge/onboard AI and Smart Payload Data but uses toolchains common with cloud architectures on ground and hence AI in data exploitation and operations. The presented architecture can also be applied to guidance, navigation, and control (GNC). The commonality with hardware and software development environments on ground is important to simplify deployment of AI in space systems. It is furthermore important for cost and resource sharing reasons, where existing code from industry or consumer business can be reused and a wider access to talent is possible.

To make a difference in the information market it is important to provide an infrastructure and ecosystem that is generic while still offering specialization

at the same time in order to minimize the size, weight, and power (SWaP) for small satellites. This is especially important, since small satellites are driving many new products and services [6].

The authors have explored edge computing and especially onboard AI data processing since 2013, leading up to a scalable radiation tolerant heterogeneous architecture first implemented using AMD 1st generation (28 nm) G-series System-on-Chip (SOC) paired with MicroSemi FPGA on an Input/output (IO) expanded industrial Qseven form factor board [6]. AMD denotes their SOCs as accelerated processing units (APUs). This paper expands on the previous work to include a full heterogeneous computer architecture also for AMD 2nd generation (28 nm) G-series SOC, AMD R-series (28 nm) SOC, and the latest AMD V1000 Series (14 nm) SOC [7, 8].

8.2 Related work

Due to increasing demands of onboard sensor and autonomous processing, research has long focused on high performance and reliability. The adoption of graphical processing units (GPU) in space is emerging rapidly due to the necessity of handling massive data in-orbit or in deep space. One example of a CubeSat with heterogeneous architecture is the NASA Hyperspectral Thermal Imaging (HYT) mission being integrated by University of Hawaii [9].

Processing capabilities on CubeSat has been limited due to available SWaP and novel computer architectures have been explored like hybrid and reconfigurable computing. George and Wilson present an overview of different architectures, methods, and alternatives for onboard space computing in an overview paper [10]. The authors also describe the radiation effects that are shared between all space computers including the presented architecture in this paper. The reconfigurable computing part is defined in a field programmable gate array (FPGA) while hybrid computing is synonym with heterogenous computing, i.e., the combination of CPU + GPU, CPU + FPGA on the same chip or board. Fault tolerant computing is needed for space computers due to the radiation background effects and uses a combination of techniques also common with the presented architecture. These include information redundancy exemplified by error detection and correction coding (EDAC), error correcting codes (ECC), cyclic redundancy check (CRC), algorithm-based fault tolerance (ABFT), and parity checking. Checkpoint and exception handling are prominent examples of software redundancy.

Adams et al. of University of Georgia have investigated a similar approach of hybrid processing as the authors with a combination of Nvidia Tegra TX2i

and Microsemi SmartFusion2 [11]. It shares similar features with the architecture presented in this paper, including the physical form factor of PC/104, stacking connector, and standard protocols. However, there is a big difference in radiation performance behaviour between the Nvidia TX2 and the AMD SOC, which is further discussed below. Very similarly, Adams et al. use the SmartFusion2 as a trusted control node and watchdog of the larger CPU + GPU SOC. In the heterogeneous architecture the FPGA use is a bit expanded as it has redundant communication paths to the SOC and can have isolated hybrid compute tasks separate from the watchdog functionality as further described below.

ESA has investigated GPU for space applications through analysis of different low-end and high-end GPUs from radiation and power consumption perspectives in the GPU4Space project [12].

NASA has conducted several studies from a radiation perspective on different GPUs including from both Nvidia and AMD [10, 11, 13]. Notable, Salazar et al. have conducted radiation testing on five COTS graphic cards, of which two AMD GPUs and three Nvidia GPUs, aiming for application on the International Space Station (ISS) in low earth orbit (LEO) radiation environment [13]. Top three among five GPUs were chosen to test under the total dose of 6 krad. However, 6 krad is very low and ISS is not a representative environment for most missions. An expanded description of radiation effects is discussed in Sect. 4. None of the cards failed in a permanent failure, all the cards have several failures, i.e., functional interrupts which are required a reboot or power cycle to get the control again. MSI HD6450 employed with AMD's GPU has performed the best and recorded 43.1 days of MTTFI (the mean time to functional interrupt).

An important rationale for the use of GPUs in space is the energy efficiency for a computing task. Kosmidis et al. and Tsog et al. have shown that GPUs can have a significantly higher power efficiency compared to CPU for the same computation [12, 14].

8.2.1 Heterogeneous computing architecture overview

Building on the initial Qseven standard derived heterogeneous design described in refence 7, Unibap AB and Troxel Aerospace Industries, Inc have coordinated to develop a next-generation onboard heterogeneous/hybrid computing platform for intelligent processing, e.g., Big Data analytics and Artificial Intelligence (AI) processing to address the need of onboard data processing using the AMD V1000 Series 14 nm embedded family of SOCs [8] and the Microchip PolarFire FPGA [15].

The initial compute architecture laid the foundation to the presented x86 embedded computer using SOC/APUs from AMD. The SOC devices are from the FT3/FT3b footprint compatible 1st and 2nd generation G-series SOCs featuring multi-core 64-bit CPU cores and integrated Graphical Processing Unit (GPU). and paired with a Microchip/Microsemi SmartFusion2 FPGA, which includes an ARM Cortex M3 Microcontroller and high-speed IO. The industrial standard Qseven interfaces are supported together with a wide range of IO expanded through the FPGA. Figure 8.1 illustrates the heterogeneous/hybrid architecture combining x86 SOC, ARM-based FPGA and optionally additional accelerators (e.g., Intel Movidius Myriad ASICs).

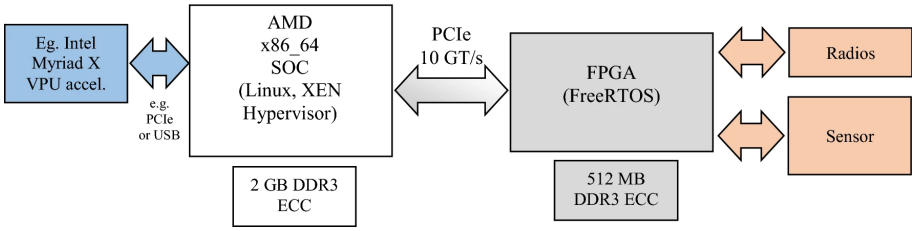


Figure 8.1: Illustration of the heterogeneous compute architecture as implemented on the Qseven industrial form factor compute board

Figure 8.1 illustrates the initial heterogenous architecture as described above. From a raw theoretical performance view, the AMD G-series SOCs have up to 87 GFLOPS GPU FP32, single precision performance. Common space interfaces such as SpaceWire, SpaceFibre and RapidIO can be supported through the FPGA or external circuits. The data rate limitation in the heterogeneous SOC-FPGA link is 10 Giga Transfer per second (GT/s) (bidirectional) over 2 lanes PCIexpress generation 2. The DDR3 memory support Error Correction Code (ECC) on both the AMD SOC and the FPGA and operate at 1066 or 1333 MHz on the AMD and 667 MHz on the FPGA. To simplify integration of new functions in the FPGA, Unibap developed a custom Direct Memory Architecture (DMA) for the heterogeneous computing architecture interaction between the AMD SOC and the FPGA over PCIexpress. Theoretically using two lanes of PCIe, an actual real data flow of 8 Gigabit/s (Gbps) is theoretically possible without the protocol overhead. Unibap has demonstrated a sustained heterogeneous bandwidth of 5.7 Gbps (i.e., 720 MB/s) using DMA over the PCIe interface.

The heterogeneous PCIe link between the AMD SOC and the FPGA is used in the NASA HYTI mission by integrating a DMA Camera Link sensor interface and providing DMA interfaces to S- and X-band radios [9].

For the purpose of demonstrating a real implementation of the architecture, a Qseven compute solution in a ½ CubeSat volume unit (0.5U) (10 × 10 × 5 cm3) called SpaceCloud™ iX5 is presented as an example of a heterogeneous computing solution suitable for spaceflight that provides advanced onboard processing for space systems.

8.2.2 High performance computing tools in space

The AMD V1000 Series SOC and AMD R-Series SOCs advances the concept of heterogeneous computing by integrating hardware features for rapid IO memory translation (IOMMU) and instructions from Heterogeneous System Architecture (HSA) standard led by HSA Foundation [16].

Supporting HSA has significant benefits to the compute architecture as the V1000 and R-Series can be made to leverage AMD's high-performance computing (HPC) software stack called Radeon Open Compute (ROCm) [17]. A particularly interesting aspect of the ROCm stack is that is can convert and execute Nvidia CUDA code and hence provide an avenue for radiation tolerant execution of CUDA code. This is also of interest, since large algorithm investments have been made in the CUDA framework and ROCm offer an avenue to leverage these investments on an open source platform.

ROCm is an open source high performance computing platform for GPU accelerated platforms. The open source aspect of the ROCm stack is important from a software radiation hardening perspective as it allows for injection of time and executing state monitoring in the code, as well as review and modifications of choice. The main development of ROCm is done by AMD and currently it is targeting mainly Linux operating systems. As illustrated in Figure 8.2, ROCm provides a complete software stack, from the Linux kernel driver, to compiler support and common libraries and software for machine learning. The core Linux driver of ROCm is accepted upstream in the Linux kernel. Currently, the latest release of ROCm is version 3.3.0 which have upstream support in Linux kernel 4.15 and 5.3 respectively for R-series and V-series. ROCm supports important features for heterogeneous computing, including:

- multi-GPU coarse-grain shared virtual memory,
- process concurrency and pre-emption,
- large memory allocations,
- HSA signals and atomics,
- user-mode queues and DMA,

- standardized loader and code-object format,
- dynamic and offline-compilation support,
- peer-to-peer multi-GPU operation with RDMA support,
- profiler trace and event-collection API,
- systems-management API and tools.

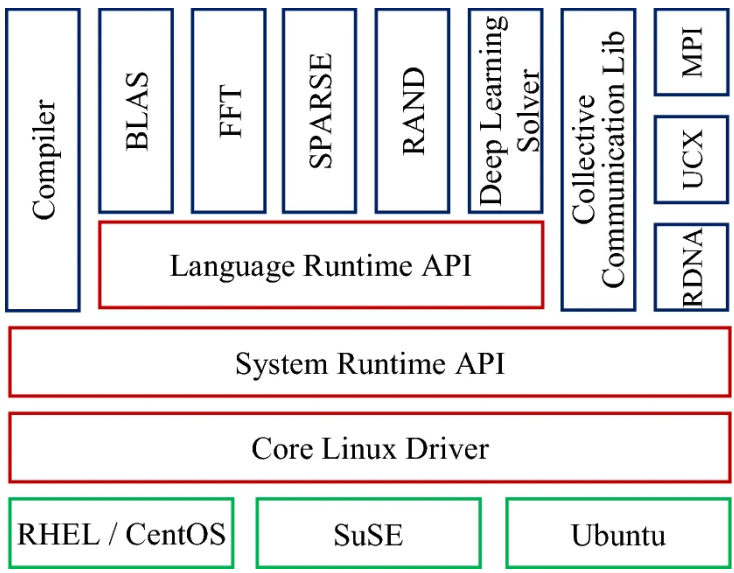


Figure 8.2: Overview of the ROCm HPC software stack

ROCm is the first HSA-compliant HPC software stack and is designed to allow other hardware vendors to adopt and develop their drivers to extend the ROCm ecosystem. The aim of HSA generally is to decrease the development complexity of applications on heterogeneous processing units (e.g., CPU, GPU, FPGA, etc.) for developers. Moreover, it allows to handle coherent shared memory through the entire heterogeneous processing units. For example, by allowing this, developers do not need to care about the different memory structures of CPU and GPU. Furthermore, the processing units see data in coherent shared memory in the same way. It reduces the mechanical data copying process between the memories of different processing units.

Comparing the 1st and 2nd generation G-series SOCs to the V1000 series reveal a significant performance uplift in performance, partially due to HSA

but mostly because of a new manufacturing process and new CPU + GPU architecture. The performance uplift and use of ROCm HPC software stack is demonstrated with benchmarks in this paper. Overall GPU compute performance of the V1000 family in 16 bit (half) floating point (FP16) is up to 3.7 TFLOPs and the SOC support up to 8 CPU threads execution using simultaneous multithreading (SMT) on quad x86 CPU cores from the AMD ZEN microarchitecture.

8.3 Stacking interface for modularity and form factor

The SpaceCloud™ iX5 is modularized by providing a core compute board and a common stacking interface based on the Samtec LSHM-150-04.0-L-V-A-S-K-TR connector [18]. The physical outline form factor of the printed circuit board (PCB) is aligned to the Pumpkin PCB Specification [19] with all PC-104 related connectors removed and replaced. Figure 8.3 shows a photograph of the Unibap e2160 heterogeneous compute module and the iX5 CORE carrier board on the left. On the right the stacking connectors are highlighted. The system is designed to operate on 12 V DC voltage and fit within a 0.5 U (10 × 10 × 5 cm) volume.

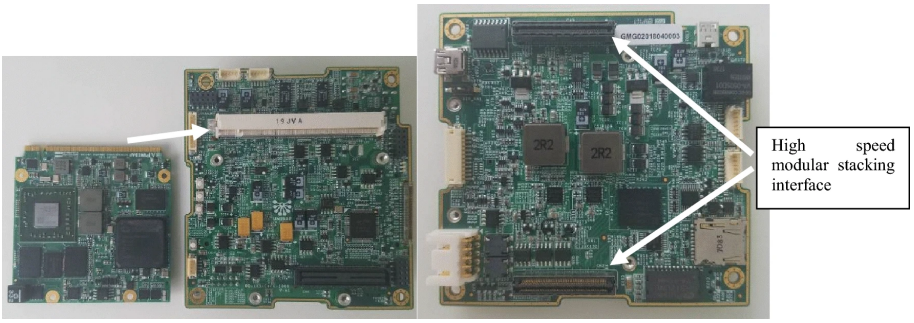


Figure 8.3: Photograph of SpaceCloud™ iX5 core components. Left, IO expanded Qseven compatible compute core (Unibap e2160) with CORE- 1000 carrier board. Right: Photograph of SpaceCloud™ iX5 CORE module with high speed stacking interface for expansion

Figure 8.4 shows a photograph of the iX5 compute module, CORE carrier board and EXTENSION board stacked together.

The signal partitioning and capabilities in the stacking connectors on the iX5 CORE module are defined in Table 8.1.

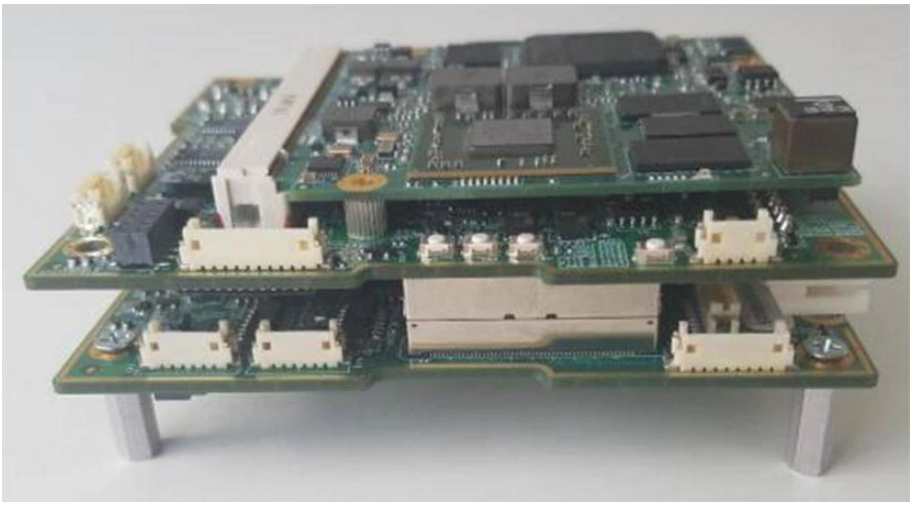


Figure 8.4: Photograph of SpaceCloud™ iX5 CORE and EXTENSION module stacked with compute module attached at the top

Table 8.1: Summary of the electrical interfaces in the high-speed expansion stacking interface (level 1 and level 2)

Interface module (level)	LSHM-150-06.0-F-DV-S-K-TR (100 pin, height 12 mm)			
	DD-iX5 CORE (level 1)		DD-iX5 EXTENSION (level 2)	
	Signal definition	Device	Signal definition	Device
CORE Module Stacking Connector Extension	16xLVDS pair @ 700Mbps	FPGA	2xI^2C	FPGA
	CAN v2.0b	FPGA	2xUSB v2.0	AMD SOC
	2xI^2C	FPGA	SPI	FPGA
	SPI	FPGA	I^2C	AMD SOC
	2xSERDES (10Gbps)	FPGA		
	12xGPIO 3.3V	FPGA		
	PCIexpress x1 (5GT/s)	AMD SOC		
CORE Module Stacking Connector Base	3.3V	DC	3.3V	DC
	3.3V on/off	FPGA	5V	DC
	5V	DC	5V on/off	FPGA
	5V on/off	FPGA	12V	DC
	12V	DC	12V on/off	FPGA
	12V on/off	FPGA	Reset	FPGA
	Reset	FPGA	12xGPIO	FPGA
	6xGPIO 3.3V	FPGA	PCIexpress x1 (5GT/s)	AMD SOC
	2xSATA v3	AMD SOC	PCIexpress x4 (20GT/s)	AMD SOC
	PCIexpress x4 (20GT/s)	AMD SOC		
	2xUSB v3	AMD SOC		
	2xUSB v2	AMD SOC		
	I^2C	AMD SOC		
	Ground			

8.4 Single-event effect mitigation middleware (SMM)

A brief discussion of radiation effects is required to understand the value of Troxel Aerospace's single-event effect mitigation middleware (SMM) and its

relevance to enabling the use of COTS processors in space applications. Several types of radiation effects have the potential to damage or create incorrect operating conditions in electronics, e.g., processors, while operating in a space environment. Total ionizing dose (TID) can be thought of as a build-up of absorbed radiation over time that changes the electrical characteristics of transistors. A transistor's ability to effectively switch without increasing leakage current degrades as dose increases to a point, where the transistor will either no longer switch and/or becomes stuck in a closed or open state. TID effects are a combination of numerous particles strikes and/or gamma irradiation over time. Other effects occur based on a single particle strike and are generally grouped into the category of single-event effects (SEEs). Within this category of radiation effects, single-event latchup (SEL) is a destructive event, whereby a single particle, typically a heavy ion, concentrates enough charge within a transistor to cause a charge path between two of the three contact points of the transistor causing an un-designed current to flow between them. If the current flow is sufficiently large or flows in an inappropriate direction, the transistor suffers permanent damage such that it becomes a current short or in some other way no longer functions properly. Other types of SEE, such as single-event upsets (SEUs) and single-event functional interrupts (SEFIs), are non-destructive events caused by a single particle (typically a proton, neutron, or heavy ion) that either causes a memory bit to "flip", i.e., change from 1 to 0 or 0 to 1, or cause the device to enter an incorrect state of operations, respectively.

Radiation hardened processors (rad-hard processors) are designed with various techniques at the basic silicon transistor layer to provide some level of immunity to the radiation effects previously described. Typical TID immunity levels exceed 100 krad up to over 1 Mrad and SEL immunity is typically above 75 MeVcm2/mg (Si). Non-destructive SEEs are designed to be so rare in these devices that they typically occur only once in 20 years. Rad-hard processors such as the BAE RAD750 [20], BAE RAD5545 [21], Cobham/Geisler LEON3FT [22], and Moog Broad Reach BRE440 [23], form the basis of many satellite control systems that require a high degree of radiation effects immunity, especially large/expensive spacecraft, human-rated vehicles, and exoplanetary missions like the Mars rovers. However, there is a large performance price paid for such radiation immunity with rad-hard processors being typically tens to hundreds of times less capable in processor performance compared to modern COTS processors [20]. If chosen carefully and validated through extensive radiation testing, COTS processors can be selected that have favourable destructive radiation effect characteristics—indeed, the AMD processors mentioned in this paper have been shown to have favourable TID and SEL characteristics [21]. However, all COTS processors exhibit high rates

of non-destructive SEEs compared to rad-hard processors and thus typically require frequent rebooting to mitigate these effects. The frequency of time between reboot vary greatly based on the underlying technology and the radiation environment in which the processor is operating. In benign environments such as the International Space Station, the time between reboot can be weeks to months while in more stringent environments such as polar orbits, GEO stationary, MEO, or HEO orbits, or exoplanetary missions, SEFI rates, and time between reboots, can be multiple per day. For many missions, particularly expensive or human-rated ones mentioned previously, where rad-hard processors are typically used, such reboot rates, and moreover, any reboot at all is unacceptable. Additionally, the observed SEFI susceptibility of COTS processors has been increasing as their feature sizes have decreased (i.e., moving from 32 to 28 nm to 14 nm, etc.) reducing times between reboot for a given mission.

To overcome this limitation and provide a means to make COTS processors viable for space applications that require both improved processing capability and reduced radiation susceptibility (i.e., less frequent reboots) Troxel Aerospace developed an SEE Mitigation Middleware that greatly improves non-destructive SEE upset rates. Troxel Aerospace's SEE Mitigation Middleware (SMM) provides core-, device-, and system-level fault tolerance by implementing multicore checking in the background in Linux. This robust middleware for heterogeneous multicore processors provides resource-aware configuration and execution management, and fault detection and mitigation. The SMM is designed to operate as either a background "scrubbing" task or as an interactive fault correction mechanism directed by missions software. The middleware software layer primarily resides between the application layer and the Operating System (OS), with extensions into and below the OS, to provide intelligent resource, fault, and power management. The middleware provides a consistent computing environment and application programming interface (API) for fault management that allows mission software to be largely agnostic to the specific underlying hardware, thereby reducing development and integration cost, complexity, and schedule.

A functional block diagram illustrating where the middleware resides within a multicore processor software stack is shown in Figure 8.5. The SMM provides an abstraction layer on which mission software, be it command and data handling (C&DH) software or applications, execute to increase portability and fault tolerance. The SMM is largely processor-agnostic and supports multiple processor architectures with core management functions fully portable across processor and Linux variants. A relatively small portion of the middleware is required to be OS- and processor-specific to support resource and fault status collection, and to execute commands to manage resources, deploy applica-

tions, and mitigate faults through processor-specific interfaces and technology. The technology-agnostic central management features of the SMM communicate to the technology-specific components (i.e., OS and hypervisor kernel extensions) through standard interfaces allowing the design to be common across processor and OS architectures.

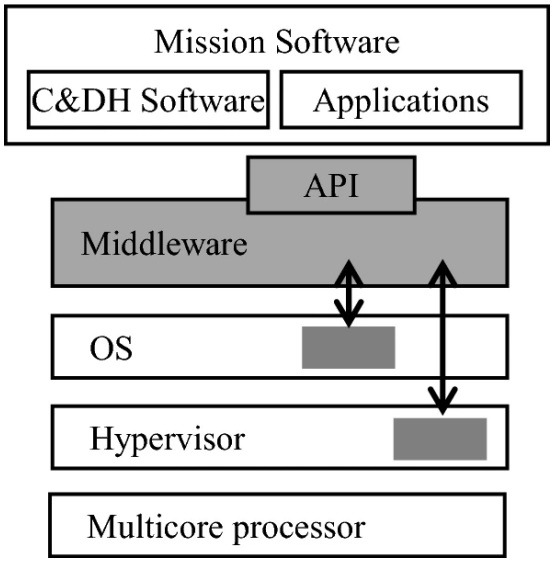


Figure 8.5: Proposed middleware software architecture

The SMM has been deployed on a homogenous quad-core ARM processor, the Unibap e2000 and e2100 family featuring AMD 1st Gen SOC fm. "eKabini" and 2nd Gen SOC fm. "Steppe Eagle" Series GPU, the Unibap V1000 series AMD, and a Digital Signal Processor (DSP), demonstrating the middleware's flexibility across platforms. Through the completion of a NASA JPL SBIR Phase II program, the SMM implementations on the Steppe Eagle and DSP were irradiated with 32 h of heavy ions using Texas A&M's Cyclotron in November and December 2019 and demonstrated SEU (bit-flip) and SEFI immunity for all error events observed demonstrating a 720× increase in non-destructive SEE susceptibility. The 720× increase takes a conservative approach by assuming that the next test observation would have resulted in an uncorrectable error, which is possible but very unlikely. Even so, this is a dramatic increase in upset rate. As mentioned in the radiation discussion above, this improvement provides a varied benefit depending on the mission orbit. To provide two examples, if the processor would otherwise suffer a SEFI (reboot) every 3 days, i.e., a relatively harsh mission, the system would

instead suffer a SEFI once every 5.9 years with Troxel Aerospace's SMM enabled. In another mission scenario, if the processor would suffer a SEFI every 30 days, i.e., a moderately harsh mission, the system would instead suffer a SEFI once every 59 years with the SMM enabled. These results demonstrate a substantial improvement in SEFI rate and would make these processors viable for a wide range of otherwise inappropriate missions such as autonomous operations, docking, exoplanetary landings, and other missions described in Section 8.5.

8.5 Mission scenarios and application

The latest available NASA crosscutting technology roadmap lists key avionics goals to include improved reliability and fault tolerance, increased autonomy, reduced size, weight, and power (SWaP), and commonality across spaceflight and ground processing systems [22]. Long-duration crewed missions, space-based observatories, and solar system exploration will require highly reliable, fault-tolerant systems. Communication delays, the challenging orbital dynamics of Near-Earth Asteroids (NEAs), and extreme science missions require increased autonomy for on-board decision infrastructures [23]. Future robotic missions will involve greater complexity and reactivity, which will require increased reliance on autonomy (i.e., advanced onboard processing). Deep-space missions that target active, dynamic, or time-varying phenomena will need robots that can adaptively adjust their configurations and behaviour to changing circumstances, and robustly handle uncertainty. Robotic missions to NEAs will require the decision-making and monitoring processes—currently performed by ground control—to be performed by onboard autonomous systems [24]. Advanced avionics technologies and approaches are needed to support these challenging missions.

Subsection TA11.1.1 of the Chief Technologists Office Technology Roadmap lists the three areas of flight computing that are critical to next-generation needs for science and exploration to include processors, memory, and high-performance flight software [22]. Scalable, multicore processors, co-processors, and memory that have a range of capabilities for fault tolerance and recovery are needed for use in radiation fields to support an increasingly software-intensive onboard environment. Flight software, called on to perform a range of functions, including increasing autonomy, will require techniques for state-based design and verification techniques to manage complexity at design time and ensure reliability and safety in operations. Historically, flight computing has focused on tight-loop operations.

Onboard experiments with intelligent onboard processing on CubeSats took

a significant step forward in 2013 when the IPEX CubeSat was launched as a secondary payload. IPEX validated a range on board instrument data-processing algorithms and autonomy [24, 25].

ESA's Earth Observation directorate have been pushing AI for small satellites through the Φ-Sat-1 satellites. The Φ-Sat-1 mission was formulated in response to an ESA challenge and consists of two 6U CubeSats. The mission will demonstrate on-orbit image filtering using AI of hyperspectral images [26]. This mission represent the comprehensive approach ESA is taking to identify and deploy AI on space mission as discussed in the introduction [4, 5]. Varile et al. have explored Convolutional Neural Networks (CNN) for autonomous image analysis [27].

Future trends show generalization toward varied requirements for flight computing, including hard real-time, mission-critical calculations that often involve vision-based algorithms such as those for entry, descent, and landing; high-data-rate instrument throughput imperatives, such as those for hyperspectral and synthetic aperture radar; and the increasing use of model-based reasoning techniques like those for mission planning and fault management. Future flight computing systems must provide heterogeneous architectural support across this spectrum of computational drivers, including uncertainty, distribution, concurrency, and operations. As more capable science instruments observe and capture larger volumes of data, there is a need to develop methods for data reduction and triage at the point of collection. The introduction of intelligent machine-learning algorithms onboard is a critical technology area that is important for helping to address the entire end-to-end observing path in data-driven environments. Furthermore, the need to respond to and update observation plans is a critical part of moving towards more autonomous operations. This paradigm shift will require new onboard capabilities as demands for computation, storage, and software continue to grow to enable more autonomous operations coupled with onboard data services.

Additionally, new paradigms for fleet management and sustainment, such as the Digital Twin, which are enabling to extended autonomous operations, amplify the need for robust onboard computing [28]. Pinpoint landing, hazard avoidance, rendezvous-and-capture, and surface mobility are directly tied to the availability of high-performance space-based computing. In addition, multicore architectures have significant potential to implement scalable computing, thereby lowering spacecraft vehicle mass and power by reducing the number of dedicated systems needed to implement onboard functions. These requirements are equally important to space science and human exploration missions. In addition, power-efficient, high-performance, radiation-tolerant processors and the peripheral electronics required to implement functional

systems could also benefit commercial aerospace entities and other governmental agencies that require high-capability spaceflight systems. Advances in middleware to support cooperative processing in combining high-performance multicore general-purpose processors (GPPs) and niche co-processors, such as the robust middleware proposed by Troxel Aerospace, and heterogeneous computing architectures by Unibap, is required to achieve planned mission performance requirements.

8.5.1 Applications

There are many mission's scenarios and applications, where massive onboard processing is critical as discussed earlier in the paper. Some mission are prime candidates for advanced onboard computing, including the following types:

- Autonomous rendezvous and docking
- Quick react, low latency science observations, where human time scales are not enough to react
- Exo-planetary avionics and science missions, where message latency is too long
- Downlink bandwidth limited missions (high rate sensors), where intelligent data reduction is required

An example of a bandwidth limited mission that leveraging onboard radiation tolerant heterogenous ×86 computing is the NASA Hyperspectral Thermal Imaging, HyTI mission, due for launch in 2021 [9]. The HyTI mission is a 6U CubeSat that will demonstrate spectral thermal imaging from Low Earth Orbit (LEO) orbit with onboard science data product generation.

8.6 Software overview

The heterogeneous architecture allows for software portioning over different compute nodes in the heterogenous architecture (i.e., multi-core CPU, GPU, and the FPGA in this case). It is possible to extend the heterogenous architecture with more compute nodes using the available peripherals such as PCIe or USB, e.g., Intel Myriad X Vision Processing Units (VPU) with 3 TOPS as illustrated in the figure.

For the purpose of benchmarking and demonstrating AI software in this paper, the software configuration listed in Table 8.1 was used with either CPU

support or both CPU and GPU support. The tools clpeak [29] and mixbench [30] was used to verify the GPU performance of 87 GFLOP for the AMD G-series SOC and 2 TFLOPs for the AMD V1605B SOC from the V1000 family.

It is important to note that the AMD HPC software stack ROCm is not possible to run on SOC/APUs after version 1.7 without modification. The official APU support has been removed from the packages. Hence, it was needed to recompile the entire stack to enable support for the AMD embedded series of devices. Bruhnspace corporation and Mälardalen University performed the ROCm patching and Bruhnspace provide an experimental software build online [31] while Unibap has patched the latest ROCm v3.3.0.

To illustrate the use of the "hipify" tool we convert a simple squaring CUDA code and execute it.

This example uses a simple squaring example, square.cu[1] to demonstrate the simplicity of using CUDA on AMD ROCm. However, it should be noted that "hipify" cannot parse CUDA assembler which need to be manually converted to AMD GPU assembler.

$ hipify-perl **square.cu**>square.cpp // ROCm "Hipify" Nivida CUDA example code to generic cpp code.

$ hipcc square.cpp -o square_hip // Compile the cpp code with AMD "hip compiler".

$./square_hip // and finally run it on ROCm stack for AMD APU devices.

info: running Square CUDA example on device **AMD Ryzen Embedded V1605B with Radeon Vega GFX**.

info: allocate host mem (7.63 MB) info: allocate device mem (7.63 MB).

info: copy Host2Device info: launch 'vector_square' kernel info: copy Device2Host.

info: check result PASSED!

8.7 Intelligent data processing performance evaluation

To demonstrate the next-generation intelligent processing capabilities of the heterogeneous compute platform, six experiments have been conducted in this paper and executed on AMD A10-8700P (codename "Carrizo") R-series SOC and AMD V1605B part of the V1000 family of SOCs using the ROCm HPC stack (v2.6.0 and v3.3.0).

[1] https://raw.githubusercontent.com/ROCm-Developer-Tools/HIP/master/samples/0_Intro/square/square.cu.

8.7.1 Evaluation environment

The experiments are performed on two refence platforms featuring V-Series V1605B and A10-8700P APUs from AMD. V1605B APU includes gfx902 (Vega) GPU with 1.1 GHz (15 W TDP setting) clock rate and Ryzen CPU with 2 GHz (15 W TDP setting) clock rate [32]. A10-8700P APU consists of Excavator CPU and gfx801 GPU that is employed in Acer E15 E5-552-T99R model notebook [33]. The clock rates of CPU and GPU in A10-8700P APU are 1.67 GHz and 0.8 GHz, respectively. The software used are defined in Table 8.2.

Table 8.2: Verified software AMD G-series SOC, AMD R-series AMD V1000

Software name	G-series	R-series/V-series	V-series
(L)Ubuntu Operating system	18.04.4 6 AMD64	18.04.4 AMD64	18.04.4 AMD64
Linux kernel	5.4.28	5.0.0	5.4.28
AMD gpu kernel driver	amdgpu	amdgpu	amdgpu
AMD IOMMU driver	–	IOMMU2	IOMMU2
AMD HSA driver	amdkfd	amdkfd	amdkfd
AMD ECC memory kernel driver	AMD64 EDAC	–	–
Unibap DMA kernel driver	1.0	–	–
GCC	7.2	8.1	7.2
cmake	3.11	3.11	3.16
LLVM	10.0.0	6.0.0	11.0-git
Mesa, patched by Unibap	20.1-devel	18.2	19.2
Libclc, patched by Unibap	2020–02-22	–	–
ROCm, patched by Unibap	–	2.6.0	3.3.0
OpenCL	1.2	2.0	2.0
OpenGL	4.6	4.6	4.6
Vulcan	1.2	1.2	1.2
Theano	1.0.0	–	–
Caffe	1.0	–	–
OpenCV	3.3.1	4.1.1	4.1.1
Robot Operating System (ROS)	1.12.13 (Kinetic)	–	–
TensorFlow	1.4	1.14.1/2.0	1.15.2/2.2
pyTorch			1.6a
PlaidML		0.6.4	0.6.4
Clpeak [19]	2019–09-05	2019–09-05	2019–09-05
Mixbench (HIP) [20]			2020–05-19

8.7.2 Experimental design

Artificial intelligence (AI) enabled applications are one of the concepts that should be employed for intelligent onboard data processing. TensorFlow[2] is explored as machine learning platform/framework in the experiments. Using TensorFlow, matrix multiplication has been performed for Experiment A on both CPU and GPU with the different sizes of the arrays. TensorFlow is an

[2]https://www.tensorflow.org/

open source machine learning platform involving tensor computations. Matrix multiplication is the fundamental of neural network, hence, we selected it in this experiment. The aim of this experiment is to discuss how the platform gains computing performance using GPU for the advanced parallel algorithms compared to CPU. Furthermore, this experiment indicates the performance of Tensorflow framework. Necessary parameters for Experiment A are described in Table 8.3.

Table 8.3: Parameters for Experiment A

Parameters	Values
Edge size of matrix	10, 20, 50, 100, 200, 500, 1000, 2000, 5000
Experiments number	100 times

Then, in Experiment B, we consider the optimal implementations of matrix multiplication provided by vendors (ROCm) as well as a well-known library (BLAS[3]) to evaluate the performance capabilities of the platforms. We use a code written in C++ for HIP compiler for GPU computing and sgemm from BLAS for CPU cores. For comparison reason, we use the program that used in Experiment A as well (Table 8.4).

Table 8.4: Parameters for Experiment B

Parameters	Values
Edge size of matrix	8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096
Experiments number	10 times

In Experiment C, we evaluate the inference performance when running two different convolutional neural network (CNN) across both CPU and GPU on the v1605b and the A10-8700P. The networks are from the TensorFlow object detection model zoo which are good candidates for transfer learning when running detection networks for earth observation on a satellite. A resolution of 512×512 is used, where multiple overlapping images can be used to cover the typical large sensor sizes seen on satellites.

In Experiment D we benchmark the compute throughput and bandwidth of ROCm 3.3.0 running on AMD V1605B.

In Experiment E we evaluate the possibility to do training on the platform. There are cases when it is impractical to get data to ground and where online learning can be done on self-supervised data such as anomaly detection on sensor readouts. Included is an experiment, where we train a simple Long short-term memory (LSTM) autoencoder on a time series anomaly detection dataset. Given the algorithmic advancement in where classification workloads

[3]BLAS – Basic Linear Algebra Subprograms https://www.netlib.org/blas/

have reduced in FLOP count by 2× every 16 months [34] workloads that is efficient to train on ground today is likely to become easier to train efficiently in orbit during the platforms lifetime.

In Experiment F we test the Intel Movidius Myriad X as neural network accelerator that can be used to offload calculations from the platform. Networks run in FP16 precision compiled through the Intel OpenVINO framework[4]. Given the drop in precision and separate implementations, a slightly different result is given for the numbers quoted for the Myriad X is on a different network.

8.7.3 Results

Experiment A Tables 8.5, 8.6, and 8.7 present the processing time of matrix multiplication on CPU and GPU with respect to edge size of matrices in the reference machines V1605B with TensorFlow 1.14.1, V1605B with TensorFlow 2.0.0, and A10-8700P with TensorFlow 2.0.0, respectively. Tables include minimum, maximum, average and median values of the processing time. For the comparison study of CPU and GPU, we focus on median values of the processing time. We confirm that the processing time on CPU increases rapidly, while edge size of matrix increases. Under edge size of 100, the usage of CPU could be better than the usage of GPU. On the other hand, we see that the processing time (median) on GPU is better than CPU when the edge size is more than 200.

Table 8.5: Processing time on both CPU and GPU in V1605B (TF 1.14.1) with respect to edge size of matrix

Edge size	On CPU (ms)				On GPU (ms)			
	Min	Max	Average	Median	Min	Max	Average	Median
10	0.12	0.20	0.15	0.15	0.39	1.77	0.63	0.58
20	0.12	0.16	0.13	0.12	0.39	1.89	0.66	0.58
50	0.13	1.74	0.24	0.19	0.36	3.82	0.58	0.50
100	0.21	0.92	0.31	0.27	0.35	3.17	0.78	0.68
200	2.43	10.64	4.62	4.05	0.32	8.72	0.64	0.53
500	30.71	49.57	36.23	35.57	1.38	32.31	2.10	1.77
1000	199.05	278.22	213.13	211.69	6.30	142.80	8.29	6.98
2000	96.27	125.78	103.78	104.17	34.82	541.33	46.17	41.36
5000	1566.19	1691.06	1631.44	1633.21	639.78	3804.77	733.65	704.74

Furthermore, we can see about 3.1 times improvements between the processing time for the edge size of 5000 on CPU, as it is 1633.21–1639.05 ms in V1605B and 5077.05 ms in A10-8700P. This result explains that the next generation platform employs a much powerful CPU based on AMD ZEN architecture. In the case of the GPU, we can see 2 times improvements between the GPUs employed in V1605B (about 704 ms for the edge size of 5000) and

[4]https://docs.openvinotoolkit.org/

Table 8.6: Processing time on both CPU and GPU in V1605B (TF 1.14.1) with respect to edge size of matrix

Edge size	On CPU (ms)				On GPU (ms)			
	Min	Max	Average	Median	Min	Max	Average	Median
10	0.13	0.33	0.16	0.16	0.37	2.56	0.77	0.65
20	0.13	0.63	0.24	0.21	0.38	2.35	0.61	0.55
50	0.12	0.21	0.13	0.13	0.40	3.54	0.62	0.56
100	0.23	1.56	0.45	0.39	0.36	3.64	0.54	0.47
200	2.90	14.41	6.49	6.65	0.36	8.93	0.65	0.51
500	17.89	73.60	45.14	45.27	1.14	32.49	1.71	1.31
1000	15.60	342.25	259.27	273.90	5.85	139.83	7.86	6.46
2000	99.04	2075.00	487.88	107.33	35.27	525.53	46.10	41.39
5000	1573.95	7671.68	1696.15	1639.05	653.73	3839.39	733.60	704.28

Table 8.7: Processing time on both CPU and GPU in V1605B (TF 1.14.1) with respect to edge size of matrix

Edge size	On CPU (ms)				On GPU (ms)			
	Min	Max	Average	Median	Min	Max	Average	Median
10	0.29	0.96	0.48	0.44	0.99	2.36	1.27	1.17
20	0.37	1.15	0.46	0.42	0.92	2.44	1.29	1.22
50	0.41	0.89	0.50	0.46	0.85	2.31	1.20	1.11
100	0.45	1.15	0.63	0.56	0.78	1.69	1.04	0.99
200	1.32	2.69	1.63	1.56	0.68	2.18	1.12	1.07
500	6.46	9.92	7.61	7.48	1.86	5.69	2.68	2.66
1000	37.06	41.46	38.80	38.75	11.32	27.72	13.48	13.39
2000	273.60	322.30	283.77	282.65	85.70	124.89	90.36	89.88
5000	4336.91	5259.34	4948.74	5077.05	1458.28	1770.54	1472.15	1464.71

A10-8700P (1464.71 ms). In addition, we do not confirm big difference between the different versions of TensorFlow used for the experiments in V1000.

Experiment B Table 8.8 presents the comparison study of processing times of the different implementations of matrix multiplication. TF-GPU and TF-CPU describe a matrix multiplication code using TensorFlow 2.0.0 on GPU and CPU, respectively. HIP means a code provided in ROCm software stack and is implemented for GPU using HIP compiler. BLAS describes a matrix multiplication code for CPU computation using BLAS library. We consider both HIP and BLAS as optimized codes, since they are provided by vendors or a well-known benchmarking library. TF is our target framework in this paper, and we evaluate it by conducting the comparison study with the optimized codes. Naïve is a naïve implementation of matrix multiplication for CPU written in C. As a note, both HIP and BLAS are written in C/C++, and TF-CPU and TF-GPU are written in Python.

Matrices with edge sizes larger than 128, we see that TF-GPU performs better than HIP for both reference machines. Moreover, TF-CPU on A10-8700P leads BLAS on A10-8700P as well. Only TF-CPU on V1605B performs less compared to BLAS on V1605B. Since BLAS on V1605B leads BLAS on A10-8700P, it can be concluded that TensorFlow 2.0.0 is not optimized well for Ryzen CPU in V1605B. Although GPU in A10-8700P has less performance capability compared to GPU in V1605B, HIP on A10-8700P

Table 8.8: Comparison of processing times on both CPU and GPU in V1605B and A10-8700P

Edge size	Processing time on V1605B (ms)					Processing time on A10-8700P (ms)				
	GPU		CPU			GPU		CPU		
	TF-GPU	HIP	TF-CPU	BLAS	Naïve	TF-GPU	HIP	TF-CPU	BLAS	Naïve
8	0.88	3.257	0.3	0.025	0.004	0.94	0.16	0.46	0.034	0.004
16	1.06	2.935	0.33	0.027	0.035	0.96	0.161	0.53	0.049	0.031
32	0.87	2.262	0.36	0.042	0.27	1.25	0.178	0.48	0.077	0.247
64	0.94	3.238	0.45	0.133	1.957	1.06	0.298	0.5	0.188	1.562
128	0.91	4.175	1.09	0.564	14.967	1.05	1.23	0.92	1.007	17.425
256	0.96	13.307	6.36	2.909	120.50	1.29	2.815	2.26	4.428	116.73
512	2.82	72.74	39.34	21.141	1066.4	2.7	12.926	7.7	19.411	3899.4
1024	15.72	314.85	306.98	110.22	70.457 s	12.58	56.747	42.87	126.08	45.277 s
2048	100.2	1491.3	2109.1	574.4	364.58 s	87.22	319.16	303.18	1034.3	396.45 s
4096	774.51	6738.6	1742.0	4692.9	-	607.42	2100.3	2560.7	8286.7	3339.6 s

performs better than HIP on V1605B. This could be explained that the optimization of ROCm for A10-8700P is better than V1605B, since A10-8700P is one of the oldest platforms started with the ROCm development. In other words, there are room for more improvement in ROCm for gfx902 (Vega) GPU. The results of matrices with edge sizes smaller than 64 are less informative. This is, because, the different programming languages use, and their time measurement methods are slightly different. Hence, we can explain the overhead time influences on the results a lot in these cases. As a conclusion of this experiment, we can emphasize the optimization of TensorFlow fits well with our reference machines.

Experiment C Two pre-trained models, Model A and Model B, are considered in this experiment. Model A is a mobilenet with ssd[5] and Model B is a resnet50 with faster rcnn[6]. The experiment is run on randomized data across 256 images split into 16 batches and on 32-bit floating point (single precision). The APU is set to do automatic thermal management (to the threshold 12 W or 15 W TDP) to get a balanced overall system performance. Adding additional priority to the GPU can yield faster inference as indicated in the GPU high column but with CPU clocks dropping to 400 MHz and the total power increasing to maximum TDP (Table 8.9).

Experiment D Figure 8.6 shows that the maximum throughput of 2.2 TFLOP

[5]https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.mdssd_mobilenet_v1_coco

[6]https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.mdfaster_rcnn_resnet50_coco

Table 8.9: Comparison results of Model A and Model B

Model	V1605B			A10-8700P	
	CPU	GPU	GPU high priority mode	CPU	GPU
A. Mobilenet SSD (512 × 512 pixel images per second)	7.7	6.2	5.3	8.3	5.4
B. Resnet50 faster rcnn (512 × 512 pixel images per second)	0.28	0.28	0.44	0.20	N/A

is reach at 9 GB/s bandwidth and 2 TFLOP throughput at 16.5 GB/s bandwidth is measured using mixbench (HIP, alt mode) for the AMD V1605B embedded APU in single memory configuration. The V1605b support dual memory configuration.

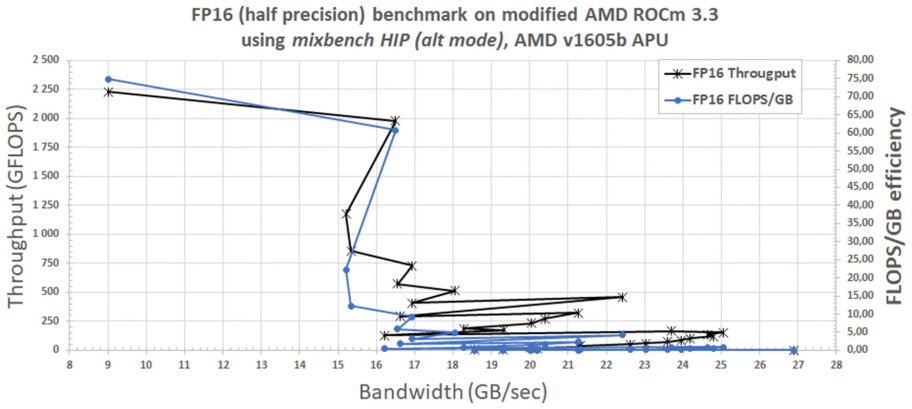


Figure 8.6: Throughput vs bandwidth benchmark of ROCm v3.3.0 on AMD V1605b using mixbench

Experiment E The training benchmark is run using a 128 LSTM run on a single dimensional temperature dataset. While this is limited in scope this mirrors the usefulness of, e.g., monitoring sensors on board and finding out when adjustments needs to be done to various instruments. This training is compared to a typical server as found on the ground, in this case a 24 core AMD ThreadRipper with a Nvidia 2080 RTX GPU. Given the limited size of the network this perform similarly on a server class cpu and gpu and the difference to the embedded platform for this type of workload is smaller making the case for training directly on the V1605B platform stronger.

Experiment F The Myriad X experiment is run over USB 3.0 on a Mobilenet SSD (depth multiplier 1.0) with only 1 output category and 16-bit floating point. The network has a fixed input resolution of 480 × 384.

Model	V1605B		Ground based server	
	CPU	GPU	CPU	GPU
Time series anomaly detection (steps/s)	10	50	175	280

Using both execution slots and queuing up 32 jobs the average rate that can be processed is 29.8 fps.

8.8 Conclusions

A radiation tolerant CubeSat compatible onboard information processing architecture have been prototyped and evaluated. Evaluation of AMD 28 nm and 14 nm embedded products with multicore CPU and GPU have shown significant benefits in acceleration of radiation tolerant and potentially radiation hardened compute tasks.

AMD's high performance computing software stack ROCm have been patched to enable embedded devices and shown to support machine learning software like TensorFlow and execution of CUDA code in a radiation tolerant/hardened silicon.

The experiments show that the theoretical compute throughput is reached in benchmarks but real applications using TensorFlow can be further optimized.

It has been shown that deep learning training can efficiently be performed in orbit and that neural networks tuned for Earth observation applications can be used for near real-time onboard information processing and onboard training.

The heterogeneous architecture is tested by expanded the AMD SOC with an Intel Movidius Myriad X neural accelerator which can significantly increase the AI processing speeds but at lower bit resolution.

Bibliography

[1] G. Richardson, K. Schmitt, M. Covert, and C. Rogers, "Small satellite trends 2009-2013," 2015.

[2] T. Segert, "Why did Google dump Skybox?" [Online]. Available: https://www.linkedin.com/pulse/why-did-google-dump-skybox-tom-segert/ (Accessed Nov 14, 2019)

[3] M. T. Hicks and C. Niederstrasser, "Small sat at 30: trends, patterns, and discoveries," 2016.

[4] P.-P. Mathieu, S. Loekken, and et al., "Towards a european ai4eo research and innovation agenda," in *ESA Phi-lab workshop proceedings*. ESA, 28 September 2018, pp. 1–16.

[5] ESA-TECT, *GSTP Element 1 "Develop" compendium 2019*. ESA, 2019. [Online]. Available: http://emits.sso.esa.int/emits-doc/ESTEC/News/GSTPAICompedium2019.pdf

[6] J. R. Behrens and B. Lal, "Exploring trends in the global small satellite ecosystem," *New Space*, vol. 7, no. 3, pp. 126–136, 2019.

[7] F. Bruhn, K. Brunberg, J. Hines, L. Asplund, and M. Norgren, "Introducing radiation tolerant heterogeneous computers for small satellites," in *2015 IEEE Aerospace Conference*. IEEE, 2015, pp. 1–10.

[8] AMD, "AMD V1000 Series SOC." [Online]. Available: https://www.amd.com/en/products/embedded-ryzen-v1000-series (Accessed Nov 3, 2019)

[9] R. Wright, M. Nunes, P. Lucey, L. Flynn, T. George, S. Gunapala, D. Ting, A. Soibel, C. Ferrari-Wong, A. Flom *et al.*, "Hyti: thermal hyperspectral imaging from a cubesat platform," in *IGARSS 2019-2019 IEEE International Geoscience and Remote Sensing Symposium*. IEEE, 2019, pp. 4982–4985.

[10] A. D. George and C. M. Wilson, "Onboard processing with hybrid and reconfigurable computing on small satellites," *Proceedings of the IEEE*, vol. 106, no. 3, pp. 458–470, 2018.

[11] C. Adams, A. Spain, J. Parker, M. Hevert, J. Roach, and D. Cotten, "Towards an integrated gpu accelerated soc as a flight computer for small satellites," in *2019 IEEE Aerospace Conference*. IEEE, 2019, pp. 1–7.

[12] L. Kosmidis, J. Lachaize, J. Abella, O. Notebaert, F. J. Cazorla, and D. Steenari, "Gpu4s: Embedded gpus in space," in *2019 22nd Euromicro Conference on Digital System Design (DSD)*. IEEE, 2019, pp. 399–405.

[13] G. A. Salazar and G. F. Steele, "Commercial off-the-shelf (cots) graphics processing board (gpb) radiation test evaluation report," *National Aeronautics and Space Administration*, 2013.

[14] N. Tsog, M. Behnam, M. Sjödin, and F. Bruhn, "Intelligent data processing using in-orbit advanced algorithms on heterogeneous system architecture," in *2018 IEEE Aerospace Conference*. IEEE, 2018, pp. 1–8.

[15] Microchip, "Microchip PolarFire FPGA." [Online]. Available: https://www.microsemi.com/product-directory/fpgas/3854-polarfire-fpgas (Accessed Nov 4, 2019)

[16] HSAFoundation, "HSA Foundation." [Online]. Available: https://www.hsafoundation.com/ (Accessed May 14, 2020)

[17] AMD, "ROCm Platform." [Online]. Available: https://rocmdocs.amd.com/en/latest (Accessed May 14, 2020)

[18] Samtec, "Fine Pitch Self Mating Connectors." [Online]. Available: https://suddendocs.samtec.com/catalog_english/lshm_dv.pdf (Accessed May 14, 2020)

[19] Pumpkin, "CubeSat Kit PCB Specification." [Online]. Available: http://www.cubesatkit.com/docs/CSK_PCB_Spec-A5 (Accessed May 14, 2020)

[20] N. F. Haddad, R. D. Brown, R. Ferguson, A. T. Kelly, R. K. Lawrence, D. M. Pirkl, and J. C. Rodgers, "Second generation (200mhz) rad750 microprocessor radiation evaluation," in *2011 12th European Conference on Radiation and Its Effects on Components and Systems*. IEEE, 2011, pp. 877–880.

[21] BAE Systems, "RAD5545 data sheet." [Online]. Available: https://www.baesystems.com/en/download-en/20190327203103/1434571328901.pdf (Accessed May 29, 2020)

[22] F. Sturesson, J. Gaisler, R. Ginosar, and T. Liran, "Radiation characterization of a dual core leon3-ft processor," in *2011 12th European Conference on Radiation and Its Effects on Components and Systems*. IEEE, 2011, pp. 938–944.

[23] J. J. Schaefer, I. A. Troxel, M. Gruber, C. Conger, J. Schaf, and K. Narveson, "Heavy ion test results for the moog broad reach bre440 processor," in *2019 IEEE Radiation Effects Data Workshop*. IEEE, 2019, pp. 1–6.

[24] I. Troxel, "Radiation tolerant technology: enabling new mission capabilities," in *Proc. Hardened Electronics and Radiation Technology (HEART) Conference, Albuquerque, NM, March 19–22, 2013*. Hardened Electronics and Radiation Technology (HEART) Conference, 2013.

[25] Office of the Chief Technologist, "2015 nasa technology roadmaps," July 2015.

[26] M. B. Goforth, "Nasa avionics architectures for exploration (aae) and fault tolerant computing," 2014.

[27] C. Moore, "Technology development for nasa's asteroid redirect mission," in *65th International Astronautical Congress, IAC-14-D2*, 2014, pp. 8–A5.

[28] D. R. Thompson, A. Altinok, B. Bornstein, S. A. Chien, J. Doubleday, J. Bellardo, and K. L. Wagstaff, "Onboard machine learning classification of images by a cubesat in earth orbit," *AI Matters*, vol. 1, no. 4, pp. 38–40, 2015.

[29] Bhat, Krishnaraj, "clpeak." [Online]. Available: https://github.com/krrishnarraj/clpeak (Accessed Nov 1, 2019)

[30] Konstantinidis, Elias, "mixbench." [Online]. Available: https://github.com/ekondis/mixbench (Accessed Nov 3, 2020)

[31] Bruhnspace, "ROCm APU." [Online]. Available: https://bruhnspace.com/rocm-apu (Accessed Nov 2, 2019)

[32] AMD, "AMD Ryzen™ Embedded V1000 Series." [Online]. Available: https://www.amd.com/en/products/embedded-ryzen-v1000-series (Accessed May 30, 2020)

[33] CPU-World, "AMD Ryzen™ Embedded V1000 Series." [Online]. Available: https://www.cpu-world.com/CPUs/Bulldozer/AMD-A10-Series%2520A10-8700P.html (Accessed May 30, 2020)

[34] OpenAI, "AI and Efficiency." [Online]. Available: https://openai.com/blog/ai-and-efficiency/ (Accessed May 26, 2020)

Chapter 9

Paper D
Simulation and Analysis of In-Orbit Applications under Radiation Effects on COTS Platforms

Nandinbaatar Tsog, Saad Mubeen, Moris Behnam, Mikael Sjödin, Fredrik Bruhn
In the Proceedings of the 42nd International IEEE Aerospace Conference, AeroConf 2021

Abstract

Radiation effects research is crucial as it defines risk to both human bodies and spacecraft. Employing radiation-hardened products is one way to mitigate radiation effects on in-orbit systems. However, radiation effects prohibit most of the state-of-the-art commercial off-the-shelf (COTS) technologies from use in space. Furthermore, radiation effects on software components are less studied compared to hardware components. In this work, we introduce a simulation tool that simulates and performs post-simulation analysis of the impact of radiation effects on schedulability of the software task sets that execute on COTS system-on-chip (SoC) platforms within in-orbit systems. In order to provide a meaningful verification environment, single-event effects (SEEs) are introduced as aleatory disturbances characterized by probability distribution of occurrence using their predefined models. The tool supports interoperability with several other tools as it uses the extensible markup language (XML) model files for input and output, i.e., for importing input task sets and radiation effects and exporting the simulation and analysis results. The proposed tool is extensively by running simulations using a use case of an in-orbit onboard monitoring system.

9.1 Introduction

Radiation effects increase the complexity of space explorations. The radiation challenge is crucial to consider risks on both biological and mechanical systems, including equipment used in orbit such as onboard computers. As radiation effects are cumulative on the one hand, although the dose of space radiation is mostly low, its risk increases by the total time traveled in space [1, 2]. This characteristic is described by total dose of radiation, i.e., total ionizing dose (TID). Developing shielding materials or radiation-hardened products in order to mitigate radiation effects in orbit components could worsen the other limitations such as size, weight, and power (SWaP), cost, and development time. On the other hand, particles such as electrons cause electrostatic discharge, single-event effects (SEEs). Therefore, radiation effects can hinder the usage of commercial off-the-shelf (COTS) technologies that have been successful in the systems used the earth, such as COTS system on chip (SoC), including the use of integrated graphics processing units (GPUs), which improve the quality of onboard data processing [3].

Technology advancements of COTS SoC accelerators bring the possibility of intelligent onboard data processing instead of transmitting all massive raw data to ground stations via narrow downlink. Examples of onboard data processing include image processing and smart decisions based on artificial intelligence (AI), to mention a few. However, system developers need to tackle radiation risks to the systems that use COTS SoCs. A task set is said to be schedulable if all tasks complete their executions before the corresponding deadlines. Schedualbility of the task set is its property that determines if the task set is schedulable or not.

The radiation environment of deep space and on the earth's surface or in low earth orbit (LEO) are different. The radiation in LEO even varies as the reason for solar activity fluctuations [4]. The study of radiation effects on the human body and materials of components used in-orbit systems, including hardware [5, 6], is well-known and on-going. However, the study of how the radiation effects impact at the software application level is a spotless research area due to their complex and broad characteristics covering various radiation types, different types of hardware, and die revision changes through each family of hardware [7].

9.1.1 A. Contributions

This paper aims at investigating and demonstrating the impact of radiation effects on the schedulability of task sets that run on COTS SoC platforms con-

sisting of heterogeneous processing units. In this regard, the paper introduces a simulation tool, namely Mälardalen-Unibap Simulation Tool (MUST). The tool supports several types of probability distributions and models to describe the radiation effects in the simulation. Furthermore, the simulation tool is able to add processing units such as central processing unit (CPU), graphics processing unit (GPU), field-programmable gate array (FPGA) as using their settings. Our aim in this paper is to identify how the probability distributions of radiation affect the timing schedulability of the task sets using the simulation tool. Note that we consider the challenges arising due to radiation effects at the software (program) level, particularly at the granularity of operating system task and task sets. Hence, the proposed tool, MUST, can be useful for simulating the schedulability of task set under aleatory disturbances of radiations to devices when developers need to start considering unknown working environment such as space.

9.1.2 B. Organization

The rest of the paper is organized as follows. In Section 9.2, the system architecture of MUST and other background information are provided. The layout and implementation of the tool is discussed in Section 9.3. Experimental evaluation and a discussion of the tool is followed in Section 9.4. Related work and related tools are introduced in Section 9.5. Section 9.6 concludes the paper and discusses future work.

9.2 MUST: System architecture

9.2.1 A. System Model

The system model considered in Mälardalen-Unibap Simulation Tool (MUST) consists of a system S. The system S comprises of radiation effect χ, m numbers of devices $\{P_m\}$ employed in onboard computer platforms including heterogeneous processing units (such as CPU and GPU), and a task set Γ. A task set means a set of programs/applications such as threads in Linux. The system is represented by the following tuple:

$$S = \langle \chi, \{P_m\}, \Gamma \rangle \tag{9.1}$$

We consider the fixed priority preemptive scheduling policy for CPU scheduler and non-preemptive fixed priority scheduling policy for GPU scheduler [8].

9.2.2 B. Task Model

Each task $\tau_i \in \Gamma$ is executed periodically and described with its worst case execution time C_i, its activation period T_i, and its relative deadline D_i (the deadline considered from the beginning of its activation period), i.e.,

$$\tau_i = \langle C_i, T_i, D_i \rangle \tag{9.2}$$

In order to simulate the task set Γ and check its schedulability, all tasks will be executed for the time interval that is equal to the hyperperiod of all tasks, which is calculated as the least common multiple of periods of all tasks, i.e.,

$$HP(\Gamma) = LCM(T_i) \tag{9.3}$$

for all $\tau_i \in \Gamma$. In other words, a task τ_i could be executed several times during the hyperperiod $HP(\Gamma)$. Therefore, we consider jobs of task τ_i and j^{th} job of task τ_i is denoted as $\tau_{i,j}$. Jobs are the released execution instances of a periodic task in each period. Every task consists of sequential, parallel, sequential segments. Sequential segments highlighted with blue color can be executed only sequential manner using CPU, while parallel segments highlighted with red color can be executed either on CPU sequential or on GPU parallel manner as depicted in Figure 9.1. C_i is expressed with an idea of alternative executions for parallel segments [9] on heterogeneous processing units $\{P_m\}$.

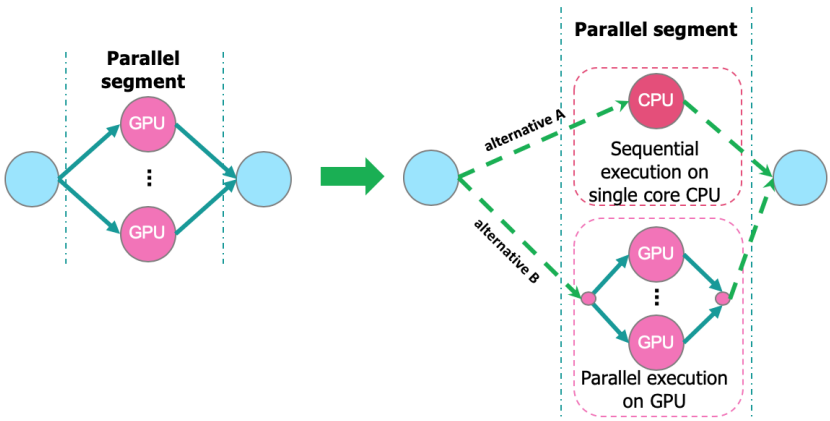


Figure 9.1: Alternative executions of parallel segment

As illustrated in Figure 9.1, this idea means that any parallel segment of

a task can be executed on different processing units for different jobs, i.e., the execution of a parallel segment of a particular task is not always allocated to one particular processing unit. For example, while a parallel segment of job $\tau_{i,j}$ executes on CPU, the same parallel segment of job $\tau_{i,k}$ may execute on GPU in order to avoid using one particular processing unit. Because, the intensive use of one particular processing unit can consequence a bottle neck.

9.2.3 C. Radiation Effect Model

The occurrence of a radiation effect generally follows the fault burst model [10, 11] in real-time systems as illustrated in Figure 9.2. The fault burst model describes the occurrence of multiple single radiation effects in radiation effect interval. This means that their distributions and total amount of radiation effects in the radiation effect interval can differ in each time, however, the burst of radiation effects can be bounded by the radiation effect interval.

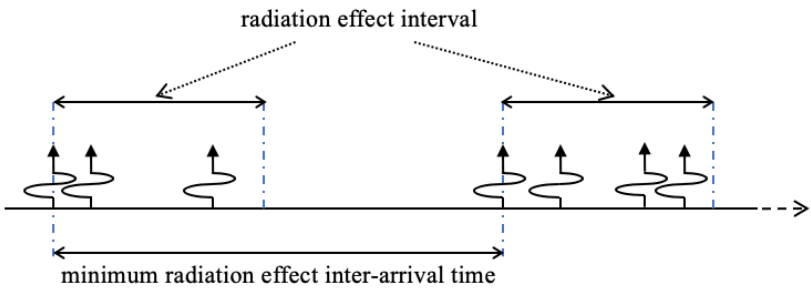


Figure 9.2: The fault burst model expressing radiation effects

Based on this model, we propose the following extended model of radiation effects employing probability distributions to radiations of environments. We consider that the system S performs under continuous radiation effects. However, the strength of radiation effect is distributed with the given probability distribution. Moreover, each device in the system S has different level of tolerances against radiations. We define it as a radiation tolerance of a device P_l (where $l \leq m$) and denote it as σ_l. Hence, the system can be executed normally under the following condition:

$$\chi(t) \leq \sigma_l \tag{9.4}$$

where t is the current clock tick and l is an index of the busy device P_l as executing a job $\tau_{i,j}$. A job $\tau_{i,j}$ needs to be re-executed if it does not satisfy the

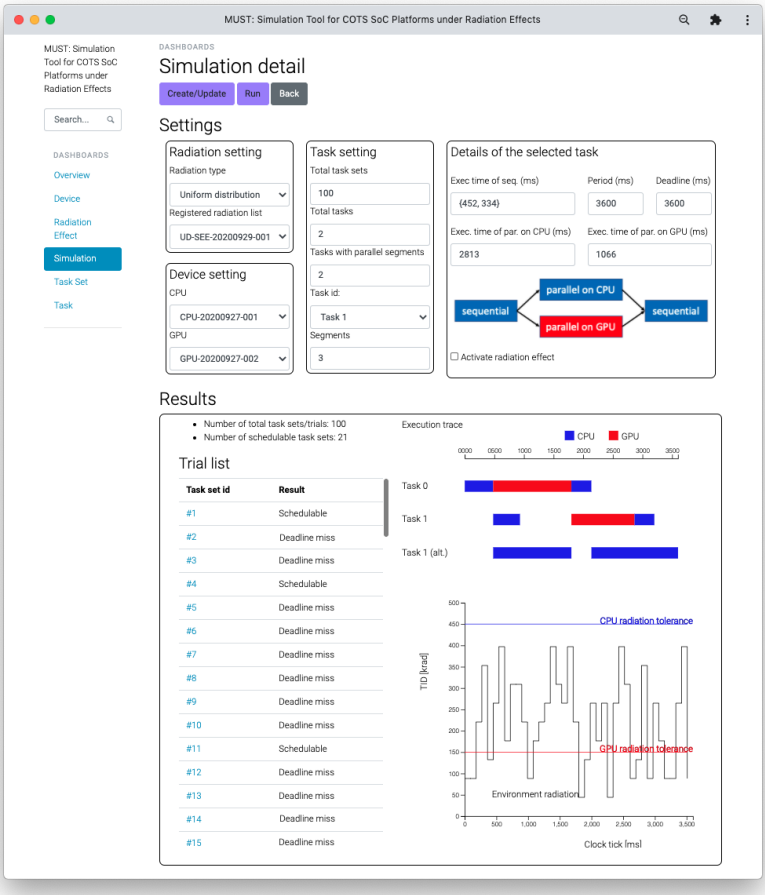
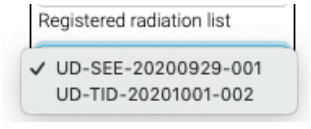
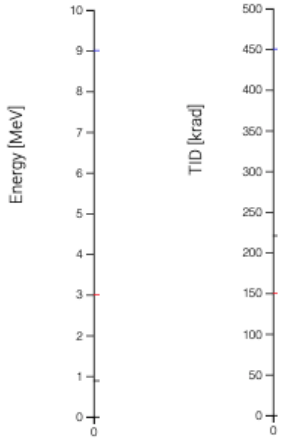


Figure 9.3: Layout of the MUST simulation tool under radiation effects

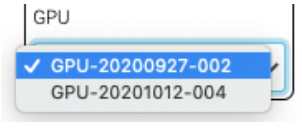


(a) Option of registered radiations



(b) Results for selected radiation types

(c) Option of registered CPUs



(d) Option of registered GPUs

Figure 9.4: Option of settings

condition described in Equation 9.4.

In this paper, we consider the following four well-known probability distributions for radiation effect χ: i) uniform distribution, ii) normal distribution, iii) triangular distribution, and iv) exponential distribution.

9.3 MUST: Design and implementation

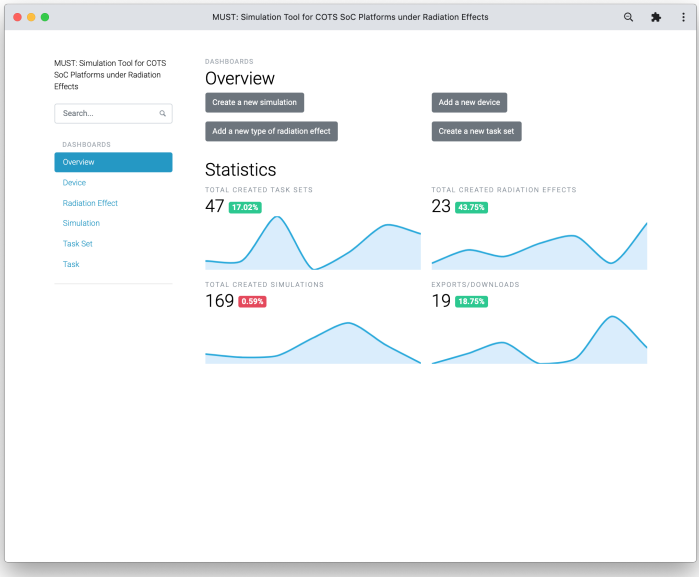
This section briefly discusses the inputs and outputs of the tool, design of the user interface of the tool, simulation mechanism, and implementation and distribution of the tool.

9.3.1 A. Input & Output

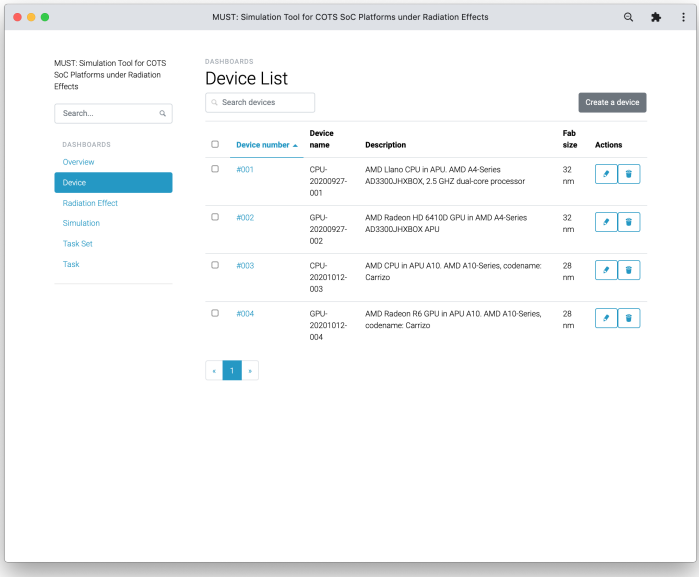
This tool performs simulations of the systems based on the input provided by the user. The input consists of number and type of devices, task sets, number of tasks and their properties in each task set, and radiation effects according to the system, task and radiation effect models discussed in Section 3. The overview of the simulation page of this simulation tool is illustrated in Figure 9.3. The necessary setting parameters are located at the upper part of the simulation page of the tool.

As depicted in Figures 9.3, 9.4(a), 9.4(c), and 9.4(d), both radiation and device settings are easily selected from the list of registered radiation types and devices, respectively. On the pages of device and radiation effect, a user can register their related information such as the radiation tolerance for devices, and probabilistic distribution and radiation type (SEE or TID) for radiations. The tool allows the users to register new devices and radiation effects. This allows the users to utilize the tool for the devices with known properties as well as for the prospective devices that the users expect to have. Regarding creating a new radiation effect, any probabilistic distributions can be used along with the existing radiation models provided in the tool. For example, the tool has the default radiation effects based on AP-8 and AE-8 models [12].

As the main output of the tool, a number of schedulable task sets (i.e., the task sets in which all tasks completed their executions before the corresponding deadlines) is reported based on the simulation results. Furthermore, the detailed results of each simulation trial are illustrated in the result part of the tool. This includes the execution trace of each simulation and radiation graph with respect to the clock tick used in the execution trace. As shown in Figure 9.4(b), the radiation graph can be illustrated with either ”TID [krad]” or ”Energy [MeV]” on the vertical axis.

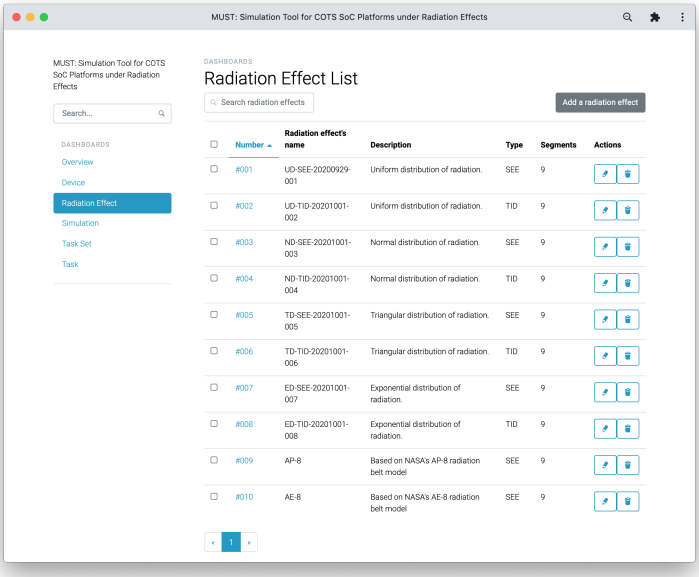


(a) Layout of overview page

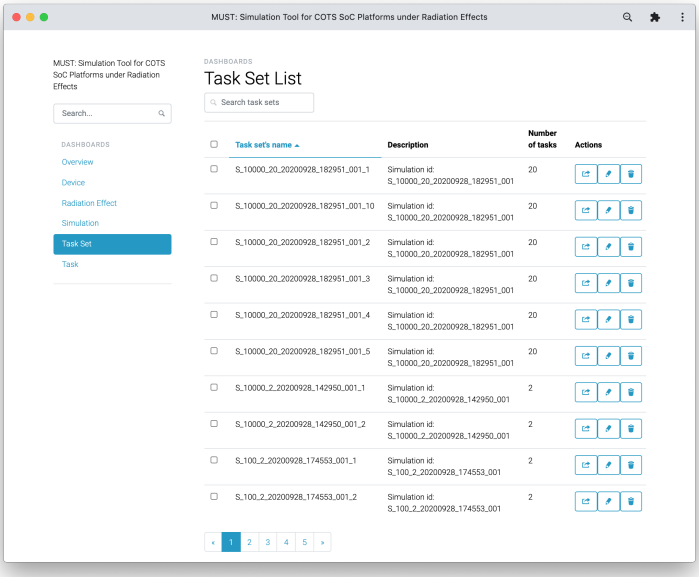


(b) Layout of device list page

Figure 9.5: Layout of various pages in the MUST tool



(a) Layout of radiation effect list



(b) Layout of task set list

Figure 9.6: Layout of various pages in the MUST tool

9.3.2 B. Design

The layout of the main pages are depicted in Figures 9.5 and 9.6. The tool consists of 3 parts, the input/setting part, the operation part, and the output/monitoring/archive part. As we discussed in the previous subsection, the device and radiation effect settings belong to the input part as illustrated in Figures 9.5(b) and 9.6(a), respectively. Based on the data received from the input part, the operation part performs simulations and produces analysis results as outputs. Hence, as depicted in Figure 9.3, the simulation page belongs to the operation part. The information generated through simulations is stored in the task lists (see Figure 9.6(b)). The created task set and radiation effects can be exported as an extensible markup language (XML) model file, which can be input to any other tool conforming to the XML format.

Further, as illustrated in Figure 9.4(a), the overview page shows the basic statistics of the tool such as the total number of the performed simulations and the created radiation effects.

9.3.3 C. Simulation Mechanism

Each task created in the tool has its own priority that depends on the setting. The different priority assignment policies such as rate monotonic (RM), deadline monotonic (DM) and earliest deadline first (EDF) can be used. In this paper, we consider only EDF priority assignment policy.

1) Generating a task set

First of all, the tool creates a task set with the assigned number of tasks. Each task is assigned an execution time, period, and deadline.

2) Assigning priorities

Since, we consider EDF, the priorities of tasks are dynamic. Thus, a job of the task with the earliest deadline gets the highest priority. In the case if two or more jobs of different tasks have the same deadlines then the priorities are assigned according to the ID numbers of the corresponding tasks, i.e., higher the ID of the task higher the priority of the corresponding job in case multiple jobs have the same deadlines. In order to perform a simulation, the allocation of parallel segments of each task to the appropriate processing unit should be handled.

3) Simulation

The simulation process continues until the clock tick reaches the time equal to the hyperperiod of the task set, $HP(\Gamma)$. At every clock tick, the simulator checks the priorities of tasks and selects the task that should be executed at this clock tick. The job of a task that has completed its execution is not considered until the release time of the next job that occurs periodically. Before executing a task, the simulator generates the environment radiation based on the type of radiation effect setting (TID or SEE). As the environment radiation is smaller than the radiation tolerance of the allocated processing unit (device) in the case of SEE, the simulator executes the task with 1 clock tick. In the case of TID, if the total exposure including current TID is still smaller than the radiation tolerance of the allocated device, the simulator executes the task with 1 clock tick as well. Otherwise, the simulator resets all the execution until this moment. This means that the task set starts from the beginning of the next clock tick, however, the current clock tick will continue. Then the simulator checks the deadline of the job. If the job misses its deadline, the simulator counts the deadline miss and ends this simulation trial.

9.3.4 D. Implementation

The user interface of the tool is developed considering the web browser based solution using the MERN stack[1] in order to be less platform dependent. The MERN stack is a combination of four web technologies, MongoDB[2], Express JS[3], React JS[4] and Node JS[5]. The user interface is based on [13], where the detailed development guide of the MERN stack can be found.

The back-end of the simulator is implemented in Python programming language[6] using MongoClient, datetime, logging, random, time, and count libraries. Hence, the simulator is less dependent on underlying platforms. As using MongoDB as NoSQL, the data structure can be extended and customized easily without destroying data in the current experiment. This means that the tool can be easily adapted for new devices and radiation models.

The tool is published and distributed to GitLab repository page[7] freely. The tool can be extended with different types of radiation models, devices and

[1] https://www.mongodb.com/mern-stack
[2] https://www.mongodb.com/
[3] https://expressjs.com/
[4] https://reactjs.org/
[5] https://nodejs.org/
[6] https://www.python.org/
[7] https://gitlab.com/nabarja/must_aeroconf2021

new execution models, for example, the adaption of CRÈME96 [14], SPENVIS [15], TASTE [16], and Radeon™ GPU Profiler[8]. All changes regarding these extensions can be tracked on this page.

9.4 MUST: Use case

This section evaluates the MUST tool using a monitoring use case of an in-orbit system and discusses how the schedulability of task sets can be improved.

9.4.1 A. Use Case Description

The evaluation of the MUST tool is considered to apply a use case of monitoring system. This use case is inspired by a smallsat computer system [17], which presents a pre-operational task-set and logging software. Satellites consist of several peripherals and it is significant to monitor their abnormal activity. As depicted in Figure 9.7, an onboard computer (OBC) handles a monitoring system with two tasks, namely τ_0 and τ_1. Task τ_0 detects and collects status of peripherals to the storage employed in the OBC. Task τ_1 analyzes the stored logs and makes an appropriate decision such as sending report to the ground station, rebooting the system, and restarting a particular peripheral, and so on.

The periods of τ_0 and τ_1 are 900 ms and 1200 ms, respectively. We consider implicit deadlines for the tasks. This means that the deadline of a task is considered to be equal to its period. Thus, the hyperperiod of the task set is 3600 ms. Each task has a parallel segment which is allocated to CPU. The execution times of the sequential, parallel, sequential segments of τ_0 and τ_1 are $\{20$ ms, 20 ms, 30 ms$\}$ and $\{40$ ms, 40 ms, 50 ms$\}$ respectively. The scheduling policy is EDF. Hence, the utilization of the task set is $U = 70/900 + 130/1200 = 0.186$, which satisfies the schedulability condition of EDF: $U \leq 1$.

The radiation tolerances of the devices, CPU and GPU, are given as 10 MeV. The environment radiation is created using the uniform distribution with the ranges' [1 MeV; 10.5 Mev]. This means that the environment radiation exceeds only 5% probability of the devices' radiation tolerance.

9.4.2 B. Evaluation and Discussion

Table 9.1 reports the simulation results of the experiment. There are 4 and 3 jobs of tasks τ_0 and τ_1 in the hyperperiod of 3600 ms. The table shows that

[8] https://gpuopen.com/rgp/

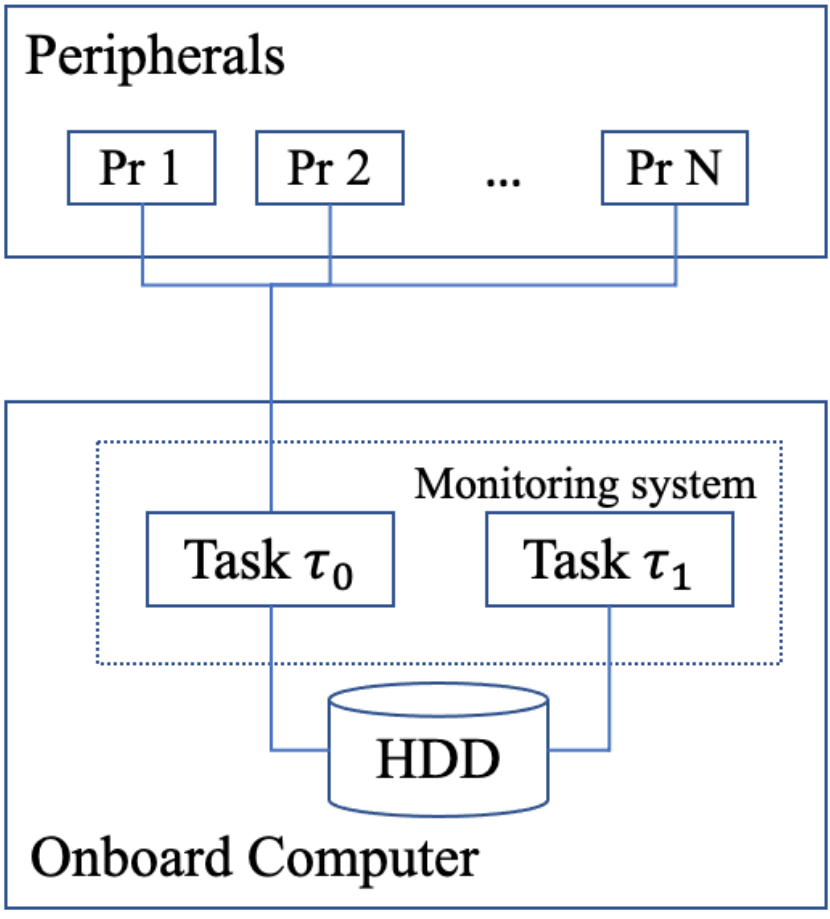


Figure 9.7: Use case: Monitoring system

each of two jobs of τ_0 within the hyperperiod complete their executions before and after the corresponding deadlines. In the case of τ_1, only one job completes its execution before the deadline, while two of its jobs miss their deadlines in the hyperperiod. Thus, the task set is not schedulable under the given radiation effect.

Let us focus on the job $\tau_{0,1}$. During 483 ms, the first and second segments of the job restarted 26 and 5 times, respectively. Furthermore, in this simulation, the first segment of any job needs to restart even if the second or the third segments experience the radiation effect while they are executing.

This experiment reveals that the smaller the size of the segment the lesser it is affected by the radiation effect. Hence, an important take-away from this

Table 9.1: **Experiment results**

Job $\tau_{i,j}$	**Execution time**	**Response time**	**Result**
$\tau_{0,1}$	70 ms	483 ms	pass
$\tau_{0,2}$	70 ms	>864 ms	misses deadline
$\tau_{0,3}$	70 ms	>268 ms	misses deadline
$\tau_{0,4}$	70 ms	169 ms	pass
$\tau_{1,1}$	130 ms	451 ms	pass
$\tau_{1,2}$	130 ms	600 ms	misses deadline
$\tau_{1,3}$	130 ms	729 ms	misses deadline

experiment is that the schedulability of the systems under radiation effects can be improved if the the execution times of the jobs are partitioned into smaller chunks so that the partial execution results can be more frequently saved. Developing such a technique to determine optimized chunks of the executions is left for the future work.

9.5 Related work and tools

Space missions are limited to bring the technological advances in COTS platforms due to the radiation effects. There are many works that focus on measuring the behaviour of COTS platforms under radiation effects [7], [18], [5]. These works consider the effect of radiation regarding total ionizing dose (TID) and single-event effects (SEEs) on in-orbit hardware and materials used in the spacecraft. Miller et al. [6] and Troxel [7] consider the radiation effect on commercial DRAMs. The exposed particle can damage hardware, which can end up with data loss as well. Moreover, the authors mention the changes of chip revision within each family can be another concern of radiation effects. Therefore, the current state of the art focuses on how radiation effects can affect materials of hardware that, in turn damages the stored data. There is a lack of research on investigation of radiation effects on the execution behavior of applications that are stored in the hardware.

Besides performing radiation testing, the space missions predict the radiation effects using several existing tools. CRÈME96 is a state-of-the-art prediction tool for SEEs based on the Cosmic Ray on Micro-Electronics code that provides better description of the environment with ionizing radiations and improved calculations of single-event upsets (SEUs) [14]. CRÈME96 provides

the prediction models that predict how cosmic ray affects microelectronics. The European Space Agency (ESA) provides space environment information system (SPENVIS) [15]. SPENVIS provides models of the hazardous space environment including cosmic rays, radiation belts, solar energetic particles, among others. OMERE[9] is a freeware radiation software dedicated to radiation effects to electronic devices in space environment. This tool is developed with the support of the National Centre for Space Studies (CNES) based on the industrial requirements from several organizations and companies. OMERE computes particle fluxes as the space environment, and dose, displacement damage, SEEs and solar cell degradation as radiation effects on electronic devices. We plan to add models from both CRÈME96, SPENVIS, and OMERE in future release of our tool.

Under the ESA initiative, TASTE is developed as a development tool-chain that targets heterogeneous, real-time, and embedded systems [16]. TASTE supports model-based development and provides early verification and testing of generated applications. We consider that bringing a possibility to import the TASTE generated applications as tasks to our tool can broaden the usability of it. Further, Radeon™ GPU Profiler provides the detailed execution trace of tasks on GPU computing. We plan to include it in the future release of our tool as this will help to consider the reality of task segments.

9.6 Conclusions

This paper introduced the architecture, design, implementation and simulation mechanisms of a new simulation tool for the task sets running on heterogeneous processing units that are subject to radiation effects. Furthermore, the tool performs post-simulation analysis to check the schedulability of the task set. The occurrence of radiation effects in this work is described with common probability distributions. As one of the outputs, the tool provides the rate of deadline misses among simulated task sets. The tool is designed to support interoperability with other tools that use the XML format for inter-tool communication. That is, the task sets and radiation models can be exchanged with the XML model files.

The preliminary experiment using the tool shows that a technique splits a task into small segments and guarantee to save their executed results from radiation effects can improve schedulability of task sets. As future work, the tool can be extended to interoperate with the existing tools such as the cosmic ray effects on micro-electronics CRÈME 96, ESA's space environment infor-

[9] https://www.trad.fr/en/space/omere-software/

mation system SPENVIS, and Radeon™ GPU Profiler. Further, although the tool includes the simplified NASA radiation belt models, AP8 and AE8, we continue to improve this simplified model in the tool.

Bibliography

[1] L. Walsh, U. Schneider, A. Fogtman, C. Kausch, S. McKenna-Lawlor, L. Narici, J. Ngo-Anh, G. Reitz, L. Sabatier, G. Santin *et al.*, "Research plans in europe for radiation health hazard assessment in exploratory space missions," *Life sciences in space research*, vol. 21, pp. 73–82, 2019.

[2] J. Rask, W. Vercoutere, B. Navarro, and A. Krause, "Space faring: The radiation challenge," *Nasa, Module*, vol. 3, no. 8, p. 9, 2008.

[3] L. L. Bello, R. Mariani, S. Mubeen, and S. Saponara, "Recent advances and trends in on-board embedded and networked automotive systems," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 2, pp. 1038–1051, 2018.

[4] L. M. Martines S, "Analysis of leo radiation environment and its effects on spacecraft's critical electronic devices," 2011.

[5] R. Kingsbury, F. Schmidt, W. Blackwell, I. Osarentin, R. Legge, K. Cahoy, and D. Sklair, "Tid tolerance of popular cubesat components," in *2013 IEEE Radiation Effects Data Workshop (REDW)*. IEEE, 2013, pp. 1–4.

[6] C. Miller, R. Owen, M. Rose, P. M. Rutt, J. Schaefer, and I. A. Troxel, "Trends in radiation susceptibility of commercial drams for space systems," in *2009 IEEE Aerospace conference*. IEEE, 2009, pp. 1–12.

[7] I. Troxel, "Memory technology for space," *Military and Aerospace Programmable Logic Devices (MAPLD)*, 2009.

[8] L. Sha, T. Abdelzaher, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, A. K. Mok *et al.*, "Real time scheduling theory: A historical perspective," *Real-time systems*, vol. 28, no. 2, pp. 101–155, 2004.

[9] N. Tsog, M. Becker, F. Bruhn, M. Behnam, and M. Sjödin, "Static allocation of parallel tasks to improve schedulability in cpu-gpu heterogeneous real-time systems," in *IECON 2019-45th Annual Conference of the IEEE Industrial Electronics Society*, vol. 1. IEEE, 2019, pp. 4516–4522.

[10] F. Many and D. Doose, "Fault tolerance evaluation and schedulability analysis," in *Proceedings of the 2011 ACM Symposium on Applied Computing*, 2011, pp. 729–734.

[11] A. Burns, R. Davis, and S. Punnekkat, "Feasibility analysis of fault-tolerant real-time task sets," in *Proceedings of the Eighth Euromicro Workshop on Real-Time Systems*. IEEE, 1996, pp. 29–33.

[12] C. E. Jordan, "NASA radiation belt models AP-8 and AE-8," RADEX INC BEDFORD MA, Tech. Rep., 1989.

[13] C. Bonneau, "Internship report," *Mälardalen University, Sweden*, 2020.

[14] A. J. Tylka, J. H. Adams, P. R. Boberg, B. Brownstein, W. F. Dietrich, E. O. Flueckiger, E. L. Petersen, M. A. Shea, D. F. Smart, and E. C. Smith, "CREME96: A revision of the cosmic ray effects on microelectronics code," *IEEE Transactions on Nuclear Science*, vol. 44, no. 6, pp. 2150–2160, 1997.

[15] D. Heynderickx, B. Quaghebeur, E. Speelman, and E. Daly, "ESA's Space Environment Information System (SPENVIS)-A WWW interface to models of the space environment and its effects," in *38th Aerospace Sciences Meeting and Exhibit*, 2000, p. 371.

[16] M. Perrotin, E. Conquet, J. Delange, A. Schiele, and T. Tsiodras, "Taste: a real-time software engineering tool-chain overview, status, and future," in *International SDL Forum*. Springer, 2011, pp. 26–37.

[17] C. R. Julien, B. J. LaMeres, and R. J. Weber, "An fpga-based radiation tolerant smallsat computer system," in *2017 IEEE Aerospace Conference*. IEEE, 2017, pp. 1–13.

[18] D. Sinclair and J. Dyer, "Radiation effects and cots parts in smallsats," 2013.

Chapter 10

Paper E
Static Allocation of Parallel Tasks to Improve Schedulability in CPU-GPU Heterogeneous Real-Time System

Nandinbaatar Tsog, Matthias Becker, Fredrik Bruhn, Moris Behnam, Mikael Sjödin
In the Proceedings of the 45th Annual Conference of the IEEE Industrial Electronics Society, IECON 2019

Abstract

Autonomous driving is one of the main challenges of modern cars. Computer visions and intelligent on-board decision making are crucial in autonomous driving and require heterogeneous processors with high computing capability under low power consumption constraints. The progress of parallel computing using heterogeneous processing units is further supported by software frameworks like OpenCL, OpenMP, CUDA, and C++AMP. These frameworks allow the allocation of parallel computation on different compute resources. This, however, creates a difficulty in allocating the right computation segments to the right processing units in such a way that the complete system meets all its timing requirements. In this paper, we consider pre-runtime static allocations of parallel tasks to perform their execution either sequentially on CPU or in parallel using a GPU. This allows for improving any unbalanced use of GPU accelerators in a heterogeneous environment. By performing several heuristic algorithms, we show that the overuse of accelerators results in a bottle-neck of the entire system execution. The experimental results show that our allocation schemes that target a balanced use of GPU improves the system schedulability up to 90%.

10.1 Introduction

Modern cars face several cases to be solved, such as environmentally friendly vehicles, connected vehicles, and self-driving cars. For example, while identifying obstacles using computer vision applications or making more smart decisions by on-board AI (artificial intelligence) applications, modern cars should be energy efficient. In order to cope with these challenges, thus, efficient energy consumption and intelligent on-board processing are crucial. With the increasing demand on processing capability with low power-consumption, the trend of processing units in real-time systems has started to shift from single core to multi- and many-core CPUs as well as heterogeneous processing units [1]. For example, a CPU is preferred for sequential computation while a GPU shows its advantages in parallel numerical computation.

Academic work on sequential and parallel computation is broadly conducted in the real-time and high performance computing communities. In heterogeneous computing[2, 3], a task is often considered as a sequence of multiple segments. A task segment can represent either sequential execution or parallel execution, where the same function is applied on different parts of the data as shown in Figure 10.1. A parallel segment can be executed by different processing units (not only GPU), however, in this paper we only consider GPU and CPU resources. Furthermore, the allocation of parallel segments of the program to compute resources is typically done at design time. However, allocating all parallel segments to the same resource might over-restrict the systems. In high performance computing and supercomputer community, distributed computing is considered in order to decrease the overuse of the same resource [3].

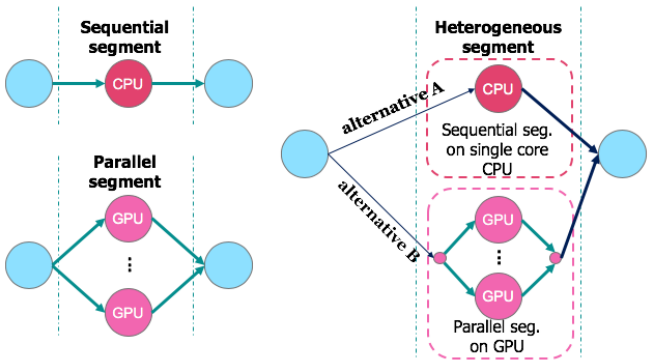


Figure 10.1: The execution alternatives of a parallel segment of tasks

In this work, we consider an extension of the fork-join task model [4], by adopting the alternative executions of parallel segments since this extended

model has been introduced in the real-time community just recently [5]. A parallel segment can either be mapped to a GPU for parallel execution (alternative B) or to a CPU for sequential execution of the same code segment (alternative A), see Figure 10.1. This then allows to take the characteristics of the execution on the different processing units into account during the system design phases where mapping decisions are taken.

Traditionally, parallel computations are often allocated to the GPU resources as it provides better performance compared to sequential execution on the CPU. However, the overuse of GPUs may end up in a bottle-neck situation [6]. Therefore, a study of how we can balance the allocation of parallel tasks to different processing units is warranted and important.

10.1.1 Contributions

In this work, we study how static allocation of parallel tasks can improve schedulability of task sets in GPU accelerated real-time systems. This allows to eliminate the bottle-neck caused by overuse of GPUs. Our main contributions in this paper are:

- We show that the overusing of GPU might bring a negative impact to the schedulability of real-time systems.
- We tackle this problem by offloading GPU computation to CPU. In other words, we conduct pre-runtime static allocations of the parallel segments either on CPU or GPU.
- We adapted the following heuristic approaches using synthetic workloads: Non-Greedy Resource Allocation Heuristic Approach (NHA), Speedup Classifier based Heuristic Approach (SHA), and Min-Min Approach (MMA). We show that the algorithms based on these approaches improve the schedulability of task sets compared to their default task mapping on the GPUs using Baseline Task Set (BTS) approach, see Section 10.5.2).
- Our synthetic experiments show up to 90% of improvement of the schedulability of task sets depending on the platform size compared to the default task mapping on the GPUs (BTS).

10.1.2 Organization

In the rest of this paper, we motivate this paper in Section 10.2. Section 10.3 presents a description of our system model, followed by detailed explanation of

the task model. Heuristic task allocation approaches are introduced in Section 10.4. In Section 10.5, we describe experiments and their setups and report our experimental evaluation. We discuss the related work in Section 10.6 and lastly, conclusions are presented in Section VII.

10.2 Motivation

Due to the following two reasons, we strongly consider the importance of balanced use of CPU and GPU in real-time systems with the partitioned fixed priority preemptive scheduling for CPU and the fixed priority non-preemptive scheduling for GPU.

- As the GPU is a single shared resource under the fixed priority non-preemptive scheduler, a higher priority task could be blocked by lower priority tasks. The blocking is well known in real-time community and it appears in many ways such as priority inversion problem.
- Interference from higher priority tasks extends the response time of a lower priority task that may result in the deadline miss.

As shown in Figure 10.2, assume a scenario for the first case above by considering the scheduling of three tasks, τ_h, τ_m and τ_l, with high, middle and low priorities, respectively. The system has 2 CPU cores and a single GPU, and follows rate-monotonic priority assignment on CPU and non-preemptive on GPU. We assume that task τ_h has 3 segments with sequential, parallel, sequential order. The worst case execution times (WCETs) of each segment are 2, 1, and 3 time units. As we focus on task τ_h, we do not consider all the segments of tasks τ_m and τ_l. Only necessary segments of these tasks are given as follows. The first segment of task τ_m is sequential with 3 time units of WCET. The first two segments of τ_l are sequential (WCET: 1 time unit) and parallel (WCET: 11 time units).

Then, task τ_l releases at 0 for 1 time unit on CPU core 2 and executes then with 11 time units on GPU. Task τ_m starts at 1 for 3 time units on CPU core 1 and tries to run on GPU. However, this execution will be blocked by task τ_l. Later, task τ_h releases at 4 on CPU core 1 for 2 time units. At 6, task τ_h tries to run on GPU, however, GPU is still occupied with task τ_l. Task τ_h starts running on GPU at 12 for 1 time unit and runs back on CPU core 1 from 13 for 3 time units. Therefore, task τ_h completes at 16 and the response time is 12.

Now we consider the alternative executions of parallel segments shown in Figure 10.3. In addition to the previous case, we assume that the WCET of the

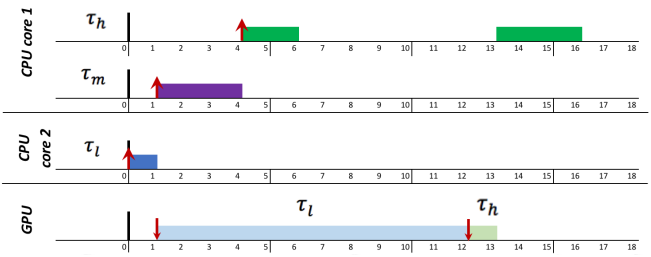


Figure 10.2: The execution of parallel segment on GPU

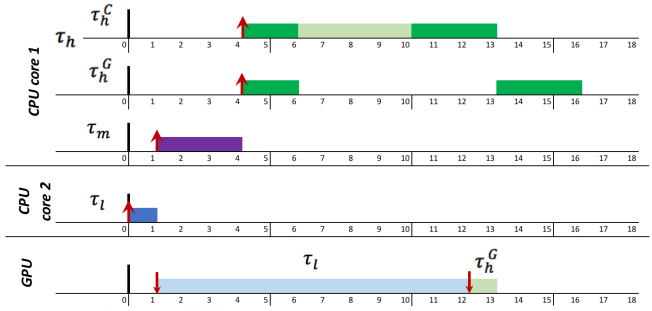


Figure 10.3: The execution of parallel segment either on GPU or CPU

parallel segment of task τ_h on CPU is 4 time units, which is 4 times longer than the WCET on GPU. In order to identify both alternatives, we use the notation of τ_h^C and τ_h^G, which are the executions of parallel segment on CPU and GPU, respectively. In Figure 10.3, we can see that the previous case is described as τ_h^G. Then, let us consider the case of τ_h^C. Until 6, the executions of both τ_h^G and τ_h^C are the same. As we know GPU is busy with τ_l at 6, τ_h^C can be executed on CPU. This execution finishes at 10, and the third segment of task τ_h is executed on CPU continuously. In this case, task τ_h finishes at 13 and its response time is 9. This means that the execution on CPU (τ_h^C) has shorter response time than the execution on GPU (τ_h^G).

10.3 System and task model

10.3.1 System model

We consider a system S, that consists of a task set Γ with n tasks. A hardware platform has m identical CPUs $\{P_m^{CPU}\}$ and a single GPU device P^{GPU}.

$$S =< \Gamma, \{P_m^{CPU}\}, P^{GPU} > \tag{10.1}$$

For CPU scheduler, the partitioned fixed priority preemptive scheduling technique is assumed. GPU allows to execute tasks with non-preemptive fixed priority scheduling.

10.3.2 Task Model

A task $\tau_i \epsilon \Gamma$ is represented by the fork-join task model, where parallel workload is modelled by fork/join segments. τ_i can be characterized by the tuple $\{S_i, D_i, T_i\}$, where D_i is the relative deadline of the task and T_i is the period of the task. S_i is composed of a finite sequence of l different execution segments $\{S_{i,1}, S_{i,2}, \ldots, S_{i,l}\}$, where $l \in \mathbb{N}$ (see Figure 10.4). Each of the segments $S_{i,j} \in S_i$ represents a number of k sub-tasks $\tau^1_{i,j}, \ldots, \tau^k_{i,j}$, where $k \in \mathbb{N}$, and has an assigned Worst-Case Execution Time (WCET) for each segment, denoted by $C_{i,j}$. We define a sequential segment for $S_{i,j}$ where k=1 (having a single sub-task), and a parallel segment for $S_{i,j}$ where k >1 (having multiple sub-tasks). We further require that the first and last segment of the task have a segment with only one sub-task. During execution, the sub-tasks of a segment $S_{i,j}$ can only be released once all sub-tasks of prior segments $S_{i,j-1}$, completed their execution. As this model defines the application, at this stage, no allocation to the different compute resources (CPU or GPU) is included and the WCET estimates are generally not known. We assume the allocation of the sequential segments on CPU is predefined. We are aware that changing this allocation might have a big impact on the performance and combining this with our solutions might provide better result. However, we leave this part as future work.

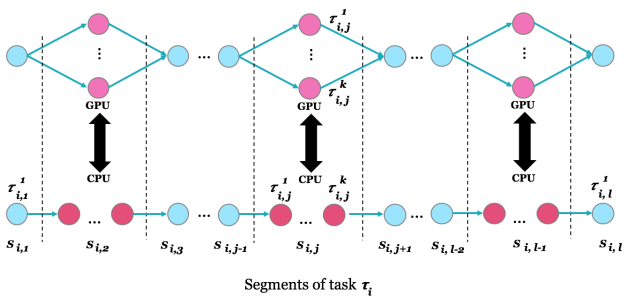


Figure 10.4: Execution segments of task τ_i and their allocations on the different resources

As the main focus of this work is the allocation of computation (parallel segments) to the different hardware types, the model is further extended. A parameter $h_{i,j}$ is added for each segment $S_{i,j} \in S_i$ of a task τ_i. $h_{i,j}$ encodes

the allocation decision, i.e. $h_{i,j} = \texttt{CPU}$ or $h_{i,j} = \texttt{GPU}$, where $\texttt{CPU}$ and $\texttt{GPU}$ are different constants.

The execution time $C_{i,j}$ is then dependent on the selected compute resource, and is represented as:

$$C_{i,j} = \begin{cases} C_{i,j}^{\texttt{CPU}} \cdot k & \text{if } h_{i,j} = \texttt{CPU} \\ C_{i,j}^{\texttt{GPU}} & \text{if } h_{i,j} = \texttt{GPU} \end{cases} \tag{10.2}$$

This represents the parallel execution of a segment, when assigned to the GPU (denoted by $C_{i,j}^{\texttt{GPU}}$) and the sequential execution of all k sub-tasks on the CPU(denoted by $C_{i,j}^{\texttt{CPU}}$). It also needs to be noted that typically execution on the GPU requires miscellaneous operations (such as copying of data). To model this, the execution time on the GPU can further be divided into $C_{i,j}^{\texttt{GPU}} = G_{i,j}^{m} + G_{i,j}^{C}$. Where $G_{i,j}^{m}$ represents miscellaneous computation and $G_{i,j}^{C}$ represents actual computation on the GPU.

To ease the later presentation of our approach we define $C_i^{\texttt{CPU}}$ (or simply C_i) and $C_i^{\texttt{GPU}}$ (or simply G_i) as the total execution time that the task τ_i requires from CPU and GPU respectively:

$$C_i^{\texttt{CPU}} = \sum_{\forall \tau_{i,j} | h_{i,j} = \texttt{CPU}} C_{i,j} \tag{10.3}$$

$$C_i^{\texttt{GPU}} = \sum_{\forall \tau_{i,j} | h_{i,j} = \texttt{GPU}} C_{i,j} \tag{10.4}$$

To give a estimate if a task τ_i is CPU-heavy or GPU-heavy, a metric is used as conversion ratio between the two options. This conversion ratio is denoted by μ_i and gives the ratio of required execution on CPU to GPU for all *parallel segments* (i.e. segments that have more than 1 sub-task, $k > 1$):

$$\mu_i = \frac{\sum\limits_{1 \leq j \leq l,\ k>1,\ h_{i,j}=\texttt{CPU}} C_{i,j}}{\sum\limits_{1 \leq j \leq l,\ k>1,\ h_{i,j}=\texttt{GPU}} C_{i,j}} \tag{10.5}$$

Smaller values of μ_i indicate that the benefit of execution on the GPU is smaller compared to larger values of μ_i.

As all segments $\tau_{i,j}$, where $k = 1$, represent purely sequential segments of execution, they must be executed on the CPU and $h_{i,j} = \texttt{CPU}$. For all other segments $h_{i,j}$ depends on the concrete allocation.

10.4 Heuristic Task Allocation Approaches

The response time of a task can be improved by selectively assigning parallel segments to CPU or GPU as highlighted in Section 10.2. The allocation of parallel segments is considered as one dimensional bin packing problem since it is defined as a problem of finding an optimal allocation of the parallel segments to the suitable processing resources. However, finding an optimal solution ends up with time complexity, which may be described with a year-unit. Thus, improved exhaustive algorithms and heuristic algorithms should be taken into account in order to decrease the search space and to find some reasonable results, respectively. Due to the limited space, in this section, we only skim an overview of heuristic approaches to allocate the tasks into their preferable processors instead of concerning the optimal allocation. The aim of these approaches is to improve the schedulability of a given task set. These approaches take a task set as an input and returns the schedulability of the task set. We assume that all the parallel segments of tasks are allocated to GPU by default.

10.4.1 Non-Greedy Resource Allocation Heuristic Approach (NHA)

NHA is a simple allocation approach, which is intended to reveal the importance of the balanced use of CPU and GPU. This approach performs the following steps.

- Step 1. Check the schedulability of the input task set by using a schedulability analysis test. This approach returns *schedulable* and stops if the task set is *schedulable*.
- Step 2. Identify a task that misses its deadline.
- Step 3. Return *not schedulable* if the task has been allocated on CPU already. Otherwise, allocate the task to CPU and goto Step 1 again.

10.4.2 Speedup Classifier based Heuristic Approach (SHA)

This approach is inspired by [3]. As illustrated in Figure 10.5, the idea of SHA is based on a queue of GPU-using tasks which is sorted by a speedup classifier. SHA follows the steps below.

- Step 1. Add tasks to a queue.
- Step 2. Sort them in order of speedup classifier as the task with the largest classifier is first and the task with the smallest classifier is last.

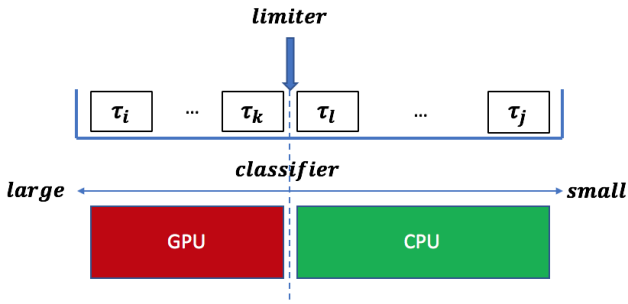


Figure 10.5: A sorted task queue based on speedup classifier

- Step 3. Allocate the parallel segments of tasks on the left side of limiter on GPU and the parallel segments of tasks on the right side of limiter on CPU. In order to find the suitable limiter, we perform the following different cases that 0%, 20%, 40%, 60%, 80% and 100% of the tasks are allocated on the left side of the limiter.

- Step 4. To check and returns the schedulability test of the task set.

In [3], the speedup classifiers are calculated by using a support vector machine (SVM). However, using machine learning algorithms may require longer allocation time, which will be a disadvantage in real-time systems. Therefore, instead of the speedup classifiers based on SVM, we consider the conversion ratio of parallel segment(μ_i), priority, and utilization of tasks. We call the algorithms used these classifiers as SHA-μ, SHA-prio, and SHA-util, respectively.

10.4.3 Min-Min Approach (MMA)

Braun et al. [7] report the Min-min approach shows the second best results among eleven static heuristics for mapping tasks onto heterogeneous distributed computing systems. The best heuristic is GA (Genetic Algorithms). However, similar to the SVM based speedup classifiers, we have adapted GA as its allocation time can be longer. In this paper, we consider the Min-min fashioned approach which is described in the steps below.

- Step 1. Initially, there is no task allocated to any processors.

- Step 2. All the CPU-using tasks are allocated to each processor which is allocated to when tasks are generated. All the GPU-using tasks are placed in a queue in order of priority.

- Step 3. Pick the highest priority task from the queue and calculates the two response times of this task using CPU or GPU for the parallel segment, respectively. If there is no more task in the queue and all the response-times are no greater than the deadlines, we say the task set is schedulable and the solution is provided.

- Step 4. Allocate the parallel segments of the task either on CPU or GPU according to the lowest response time of these cases.

- Step 5. Stop the algorithm if the systems is not schedulable when the task is allocated to the suitable processor(s) in Step 4. Otherwise, remove the task from the queue and go to Step 3.

10.5 Synthetic Experiments

In order to evaluate a wide range of application parameters with our proposed approach, synthetic experiments are performed.

Table 10.1: Initial configuration of task set generation

Parameters	**Values**
Number of CPU cores (N_p)	4, 8
Number of tasks (n)	$[2N_p, 6N_p]$
Task utilization (U_i)	[0.1, 0.2]
Task period and deadline ($T_i = D_i$)	[30, 500]ms
Percentage of GPU-using tasks	[10, 30]%
Ratio of GPU segment len. to normal WCET (G_i/C_i)	[10, 30]%
Number of parallel segments per task (η_i)	0 or 1-3
Ratio of misc. operations in $G_{i,j}$ ($G^m_{i,j}/C^{GPU}_{i,j}$)	[10, 20]%
GPU server overhead (ϵ)	$50\mu s$
Conversion ratio of parallel segments (μ_i)	[3-10]

10.5.1 Task set generation

The mechanism of task generation is based on the UUniFast algorithm, which is proposed by Bini and Buttazzo [8]. Necessary configuration values are described in Table 10.1. First, in order to generate both alternatives (the execution either on CPU or GPU), we generate the WCET of parallel segments, which is generated by using the ratio of GPU segment length to sequential segment's

WCET. Then, we create the WCET of the sequential execution of the parallel segment by using a conversion ratio of parallel segments, μ_i, which is in default a random number between 3 and 10 brought from the experimental studies from [9, 10].

10.5.2 Comparative algorithms

In order to evaluate our idea of the removal of bottlenecks of CPU-GPU, we consider 6 comparative algorithms (*BTS (Baseline Task Set)*, *NHA*, *SHA-*μ, *SHA-prio*, *SHA-util*, and *MMA*) using the proposed heuristic approaches in Section 10.4. *Baseline Task Set (BTS)* is a task set that all the parallel segments of the tasks are allocated on GPU. We choose BTS as a baseline for comparison investigation. The other algorithms are based on the proposed approaches.

The input to the approaches is a set of tasks that must fulfill the input requirements of the task model (Section 10.3.2) such as segments, deadlines and periods. As we defined in 10.3.2, the sequential segment of tasks is preemptible and self-suspending [6, 11]. Further, a parallel segment of tasks is stored in the priority-based GPU scheduler queue. The current executing parallel segment on the GPU is non-preemptive. In other words, a parallel segment can only be delayed by the higher priority parallel segments when this segment is on the GPU scheduler queue. Any schedulability analyzing test [11] for the given execution model can be used together with the proposed heuristic algorithms. We assume that the schedulability analysis returns the highest priority task which misses its deadline among tasks in the task set. Otherwise, the algorithm finishes its schedulability analysis and classifies the task set as schedulable.

10.5.3 Experiment setup

In the experiments, we used 10,000 randomly generated task sets in general. The experiments are based on the parameters as they shown in Table 10.1 except for the parameters that are varied in the respective experiment. To decide upon the schedulability of the task sets under the mapping that is created by our proposed approaches, the schedulability analysis for server-based approaches of [11, 12] are used.

10.5.4 Result

In this section, we describe the following 3 groups of experiments. *Experiment A* focuses on the understanding of the balanced use of CPU-GPU and *Experiment B* is a comparison study between 5 comparative algorithms under

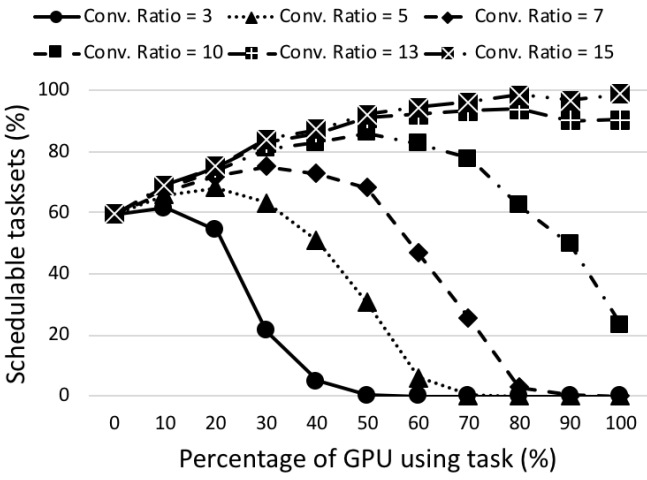


Figure 10.6: Schedulable task sets w.r.t. the fixed conversion ratio (4 CPU cores, 24 tasks)

different experiment setups. *Experiment C* is focusing on the experiment time for the heuristic algorithms. The experiments are targeting a system that has either 4 or 8 CPU cores and 1 GPU. Due to the limited space, we show only the results with 24 tasks on 4 CPU cores.

Experiment A In this experiment, we focus on the change of schedulable task sets as the task set changes from 0% of GPU-using tasks (i.e., 100% of CPU-using tasks) to 100% of GPU-using tasks (i.e., 0% of CPU-using tasks). The peak value of the change describes the balanced use of CPU-GPU. For example, in Figure 10.6, the curve for $\mu_i = 5$ gets the peak value of around 70% of schedulable tasksets at 20% of GPU-using tasks and 80% of CPU-using tasks. Furthermore, the schedulability of task sets with the different $\mu_i = 5$ and $\mu_i = 15$ results in about 0% and 97%, respectively. In other words, in case of $\mu_i = 5$, the execution time on GPU is 3 times smaller than the case of $\mu_i = 15$ where the total execution times are the same for both cases. Hence, this result could be explained that the tasks with the shorter execution time on GPU can block and interfere the other tasks less compared to the tasks with the longer execution time on GPU. This shows the importance of the balanced use of CPU-GPU. We note that this experiment uses NHA algorithm in order to understand and optimize task sets.

Experiment B Figure 10.7 illustrates the percentage of schedulable task sets with respect to the percentage of GPU using task with the default settings. We see that MMA is the best performing heuristics, since the percentage of schedulable task set is 100% in all the cases. NHA is the second best heuristics, and SHA-prio and SHA-util follow. In this case, SHA-μ did not perform

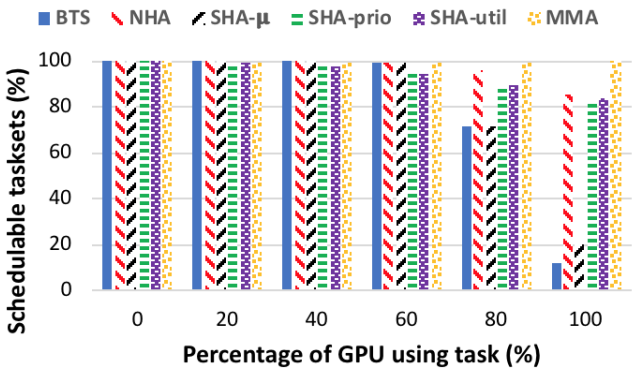


Figure 10.7: Schedulable task sets w.r.t GPU using task with the default settings

well compared to the other heuristics. Here, in 100% of GPU using task, we confirm that NHA and MMA perform about 70% and 90% better than BTS, respectively.

Figure 10.8 shows the results when the range of μ_i has been extended from [3-10] to [3-100]. This change, obviously, worsens the execution of parallel segment on the CPU as it may take 100 times longer on CPU than GPU. In this case, we see only NHA improves BTS. Further, the algorithms SHA-prio, SHA-util and MMA could not succeed for the allocation. Because, these algorithms allocate tasks with longer execution times (about 100 times) on CPU compared to the tasks in BTS. Moreover, all these algorithms would not consider the results of BTS whether they are schedulable or not. Hence, the results of these algorithms are lower than BTS (see Figure 10.8). The schedulability of task set could be improved when these algorithms include the knowledge of whether BTS is schedulable or not although the experiment time gets longer. For example, the schedulability result of BTS is the same in both Figures 10.7 and 10.8. However, these algorithms result worse than BTS in Figure 10.8, and better in Figure 10.7 On the other hand, NHA improves the results all the time compared to BTS as NHA is based on the results of BTS.

Experiment C Table 10.2 shows the experiment time of the heuristic algorithms for a task set measured 10,000 times. The experiment time consists of the times of the task generation, task allocation and schedulability analysis for a task. Max, median, mean and min values of 99.9th percentile of the experiment time are shown in Table 10.2.

We can see that the median value of NHA (40us) is the best among all the heuristic algorithms. SHA-μ (55us), SHA-prio (51us), and SHA-util (51us) are in the same level although SHA-μ takes a bit longer time compared to

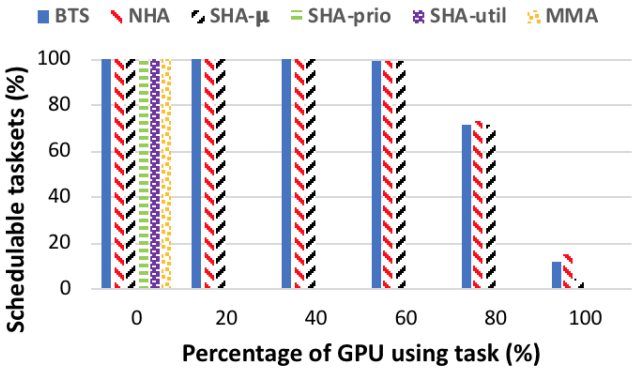


Figure 10.8: The extended range of μ_i (between 3-100)

Table 10.2: The experiment time for heuristics algorithms for a task set

No.	Heuristic Algorithm	*Experiment time [us]*			
		max	median	mean	min
1	BST	71.0	25.5	27.0	19.0
2	NHA	121.0	40.0	43.9	22.0
3	SHA-μ	134.0	55.0	58.2	34.0
4	SHA-prio	116.0	51.0	53.6	42.0
5	SHA-util	103.0	51.0	53.0	43.0
6	MMA	930.0	419.0	433.8	249.0

other 2 algorithms. MMA requires the longest experiment time among all the algorithms while MMA gives the best schedulability results in most of the scenarios. In conclusion of the experiment C, we could say that NHA is the best choice from the time-wise.

10.6 Related work

Heterogeneous computing is well studied in high performance community, especially, in supercomputers as an extension of the distributed computing [13, 14, 15, 3]. However, these techniques are not investigated targeting the real-time embedded systems. Furthermore, the techniques are mostly focused on the distributed heterogeneous systems, not on the heterogeneous systems included in the system on chips although their architecture may be similar to each other.

There exist several approaches considering the use of GPU in real-time embedded systems. Kato et al. introduced TimeGraph [16], RGEM [17] and Gdev [18] along with zero-copy I/O processing for low-latency GPU com-

puting [19]. In addition, self-suspending task techniques [20, 6] are broadly studied in real-time systems, and they are applicable to GPU accelerated real-time systems. Most of these works consider compensating the limitation of early existing GPU hardware and device drivers such as a zero-copy technique for accelerators' memory and splitting tasks into smaller chunks for allowing preemption. However, these limitations will be solved by coming new technologies such as unified memory, zero-copy and preemption technologies in CUDA[21] and Heterogeneous System Architecture (HSA)[22, 9, 10]. Furthermore, the works of Elliott et al. [23, 24] and Kim et al. [11, 12] consider worst-case timing behavior in GPU accelerated real-time systems. These works could be used as the timing analysis tool.

The bin-packing problem is one of the most important optimization problems. In our work, the problem is to find an optimal solution for the allocation of tasks to different processing units. The study of the bin-packing problem has been done widely in parallel processing [25, 26, 27, 28] and real-time systems [29]. However, due to its NP-hard nature, it is common to introduce heuristic approaches to figure out the system in an affordable time. Moreover, there are many works regarding the resource mapping in heterogeneous platforms such as [30, 31, 7, 13]. In this paper, we adopt the heuristic approaches from [7, 3] in order to understand how the behavior of the alternative execution (either on CPU or GPU) of parallel segment affects the schedulability of task sets.

10.7 Conclusions

In this paper, we target the mapping of parallel segments to either the GPU or CPU. While GPU resources can boost the performance, heavy usage can result in a bottleneck for the system. This affects on one hand tasks that want to access the shared GPU resource, as they can be blocked by other GPU segments, and on the other hand tasks that receive additional interference due to blocking by the GPU-handler task that is executed on the CPU. Thus, always executing parallel code segments on the GPU potentially degrades the system performance. We demonstrate that selective mapping of parallel segments to either CPU or GPU can improve the system performance. Heuristic algorithms are described to select the respective compute resource for such tasks. Synthetic evaluations reveal that our proposed heuristics are able to improve the schedulability of task sets up to 90% compared to task sets where no mapping decisions are taken.

Future work will focus on runtime assignment of parallel segments to CPU or GPU. Furthermore, we hope to expand our investigation to the use of multiple GPUs or any other accelerators.

Bibliography

[1] L. L. Bello, R. Mariani, S. Mubeen, and S. Saponara, "Recent advances and trends in on-board embedded and networked automotive systems," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 2, pp. 1038–1051, 2019.

[2] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *2009 IEEE international symposium on workload characterization (IISWC)*. Ieee, 2009, pp. 44–54.

[3] Y. Wen, Z. Wang, and M. F. O'boyle, "Smart multi-task scheduling for opencl programs on cpu/gpu heterogeneous platforms," in *2014 21st International conference on high performance computing (HiPC)*. IEEE, 2014, pp. 1–10.

[4] C. Maia, M. Bertogna, L. Nogueira, and L. M. Pinho, "Response-time analysis of synchronous parallel tasks in multiprocessor systems," in *Proceedings of the 22Nd International Conference on Real-Time Networks and Systems*, 2014, pp. 3–12.

[5] S. Baruah, "Resource-efficient execution of conditional parallel real-time tasks," in *European Conference on Parallel Processing*. Springer, 2018, pp. 218–231.

[6] K. Bletsas, N. Audsley, W.-H. Huang, J.-J. Chen, and G. Nelissen, "Errata for three papers (2004-05) on fixed-priority scheduling with self-suspensions," CISTER-Research Centre in Realtime and Embedded Computing Systems, Tech. Rep., 2015.

[7] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen *et al.*, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *Journal of Parallel and Distributed computing*, vol. 61, no. 6, pp. 810–837, 2001.

[8] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, no. 1-2, pp. 129–154, 2005.

[9] N. Tsog, M. Behnam, M. Sjödin, and F. Bruhn, "Intelligent data processing using in-orbit advanced algorithms on heterogeneous system architecture," in *2018 IEEE Aerospace Conference*. IEEE, 2018, pp. 1–8.

[10] N. Tsog, M. Sjödin, and F. Bruhn, "Advancing on-board big data processing using heterogeneous system architecture," in *ESA/CNES 4S Symposium 4S 2018, 28 May 2018, Sorrento, Italy*, 2018.

[11] H. Kim, P. Patel, S. Wang, and R. R. Rajkumar, "A server-based approach for predictable gpu access control," in *2017 IEEE 23rd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2017, pp. 1–10.

[12] ——, "A server-based approach for predictable gpu access with improved analysis," *Journal of Systems Architecture*, vol. 88, pp. 97–109, 2018.

[13] D. Grewe and M. F. O'Boyle, "A static task partitioning approach for heterogeneous systems using opencl," in *International Conference on Compiler Construction*. Springer, 2011, pp. 286–305.

[14] P. Czarnul and P. Rościszewski, "Optimization of execution time under power consumption constraints in a heterogeneous parallel system with gpus and cpus," in *International Conference on Distributed Computing and Networking*. Springer, 2014, pp. 66–80.

[15] H. Zhou and C. Liu, "Task mapping in heterogeneous embedded systems for fast completion time," in *2014 International Conference on Embedded Software (EMSOFT)*. IEEE, 2014, pp. 1–10.

[16] S. Kato, K. Lakshmanan, R. Rajkumar, and Y. Ishikawa, "Timegraph: Gpu scheduling for real-time multi-tasking environments," in *2011 USENIX Annual Technical Conference (USENIX ATC 11)*, 2011, pp. 17–30.

[17] S. Kato, K. Lakshmanan, A. Kumar, M. Kelkar, Y. Ishikawa, and R. Rajkumar, "Rgem: A responsive gpgpu execution model for runtime engines," in *2011 IEEE 32nd Real-Time Systems Symposium*. IEEE, 2011, pp. 57–66.

[18] S. Kato, M. McThrow, C. Maltzahn, and S. Brandt, "Gdev: First-class gpu resource management in the operating system," in *2012 USENIX Annual Technical Conference (USENIX ATC 12)*, 2012, pp. 401–412.

[19] S. Kato, J. Aumiller, and S. Brandt, "Zero-copy i/o processing for low-latency gpu computing," in *Proceedings of the ACM/IEEE 4th International Conference on Cyber-Physical Systems*, 2013, pp. 170–178.

[20] J.-J. Chen, G. Nelissen, W.-H. Huang, M. Yang, B. Brandenburg, K. Bletsas, C. Liu, P. Richard, F. Ridouard, N. Audsley *et al.*, "Many suspensions, many problems: a review of self-suspending tasks in real-time systems," *Real-Time Systems*, vol. 55, no. 1, pp. 144–207, 2019.

[21] M. Harris, ""Unified Memory for CUDA Beginners." June 19, 2017." available: https://devblogs.nvidia.com/unified-memory-cuda-beginners/ [Oct 16, 2018].

[22] HSA Foundation, ""Heterogeneous System Architecture.","available: http://www.hsafoundation.com/ [Oct 16, 2018].

[23] G. A. Elliott, B. C. Ward, and J. H. Anderson, "Gpusync: A framework for real-time gpu management," in *2013 IEEE 34th Real-Time Systems Symposium*. IEEE, 2013, pp. 33–44.

[24] G. A. Elliott and J. H. Anderson, "Globally scheduled real-time multiprocessor systems with gpus," *Real-Time Systems*, vol. 48, no. 1, pp. 34–74, 2012.

[25] J. O. Berkey, "Massively parallel computing applied to the one-dimensional bin packing problem," in *Proceedings 2nd Symposium on the Frontiers of Massively Parallel Computation*. IEEE Computer Society, 1988, pp. 317–318.

[26] E. G. Coffman, M. R. Garey, and D. S. Johnson, "Approximation algorithms for bin-packing—an updated survey," in *Algorithm design for computer system design*. Springer, 1984, pp. 49–106.

[27] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham, "Worst-case performance bounds for simple one-dimensional packing algorithms," *SIAM Journal on computing*, vol. 3, no. 4, pp. 299–325, 1974.

[28] J. D. Ullman, "Np-complete scheduling problems," *Journal of Computer and System sciences*, vol. 10, no. 3, pp. 384–393, 1975.

[29] D. De Niz and R. Rajkumar, "Partitioning bin-packing algorithms for distributed real-time systems," *International Journal of Embedded Systems*, vol. 2, no. 3-4, pp. 196–208, 2006.

[30] A. Schranzhofer, J.-J. Chen, and L. Thiele, "Dynamic power-aware mapping of applications onto heterogeneous mpsoc platforms," *IEEE Transactions on Industrial Informatics*, vol. 6, no. 4, pp. 692–707, 2010.

[31] A. H. Alhusaini, V. K. Prasanna, and C. S. Raghavendra, "A framework for mapping with resource co-allocation in heterogeneous computing systems," in *Heterogeneous Computing Workshop, 2000.(HCW 2000) Proceedings. 9th*. IEEE, 2000, pp. 273–286.

Chapter 11

Paper F
Offloading Accelerator-intensive Workloads in CPU-GPU Heterogeneous Processors

Nandinbaatar Tsog, Saad Mubeen, Fredrik Bruhn, Moris Behnam, Mikael Sjödin
In the Proceedings of the 26th International Conference on Emerging Technologies and Factory Automation, ETFA 2021

Abstract

Autonomous vehicular systems require computer vision and intelligent on-board decision making functionalities that include a mix of sequential and parallel workloads. The execution times of the workloads and power consumption in these functionalities can be lowered by utilizing the accelerators (e.g., GPU) instead of running the workloads entirely on the host processing units (CPU). However, allocating all the parallelizable workload to accelerators can create a computation bottleneck in the accelerators that, in turn, can have an adverse effect on schedulability of the systems. This paper presents a novel framework that can allocate the accelerate-intensive workloads to the accelerators as well as to the non-accelerated host processing units. Within the context of this framework, the paper introduces five offloading techniques to mitigate the accelerator-intensive workloads by utilizing excess capacity of non-accelerated processing units under dynamic scheduling in CPU-GPU heterogeneous processors. The proposed techniques are evaluated using simulation experiments. The evaluation results indicate that one of the proposed techniques can achieve up to 16% improvement in schedulability of the task sets compared to the traditional non-offloading technique.

11.1 Introduction

Employing accelerators in modern embedded platforms increases the diversification of embedded system applications. Examples of the accelerators include graphics processing units (GPUs), field-programmable gate arrays (FPGAs) and digital signal processors (DSPs), which are often used in parallel programming applications like computer vision. The accelerators often perform better than general-purpose processors (i.e., central processing units (CPUs)) with respect to latency and power consumption. For example, in autonomous vehicles, computer vision and intelligent decision making often require the use of heterogeneous processors (e.g., CPU and GPU) [1, 2]. However, the accelerators act as shared resources within heterogeneous processors [3, 4], which brings challenges of synchronization and blocking.

While the processing trend has been shifting from single-core to multi- and many-core processors as well as to heterogeneous processors, the development of single-core processors is also progressing, e.g., AMD Ryzen 5000 series with Zen3 architecture CPUs[1] and 11-th Generation Intel Core i9 processors[2]. In other words, the host processing units, CPUs, in heterogeneous processors are becoming more and more capable of assisting the accelerators in computing accelerator-intensive workloads. This paper focuses on techniques to offload accelerator-intensive workloads from GPUs to non-accelerated host processing units in the systems with heterogeneous processors. We assume that accelerator-intensive workloads contain a segment, a parallel segment, which can be parallelizable on the accelerators. Furthermore, in this paper, we consider that the applications that run on the heterogeneous processors are constrained by real-time requirements.

The main contributions in this paper are as follows:

- We propose a new framework to allocate accelerator-intensive workloads in CPU-GPU heterogeneous processors. The proposed framework utilizes the alternative executions of parallel segments of the workloads [4, 5] and the server-based scheduling [6, 7, 8].
- Based on the proposed framework, we introduce five techniques for mitigating the accelerator-intensive workloads by lowering the overuse of the accelerators. The proposed techniques use the resource-reservation mechanism by means of servers to offload the execution of parallel segments of the workload to non-accelerated host processing units.

[1] https://www.amd.com/en/processors/ryzen

[2] https://www.intel.com/content/www/us/en/products/details/processors/core/i9.html

- We perform a comparative evaluation of the proposed offloading techniques with respect to the baseline technique that always executes the parallel segments on the accelerators. The evaluation is performed on the basis of schedulability of the task sets and the time to perform the offloading.

The rest of the paper is organized as follows. Section 11.2 discusses the related work. Section 11.3 presents the proposed system model, followed by the proposed workload allocation framework in Section 11.4. The offloading techniques for mitigating the accelerator-intensive workloads are presented in Section 11.5. Section 11.6 presents the experimental evaluation. Finally, Section 11.7 discusses the conclusion and future work.

11.2 Related Work

Historically, the adoption of heterogeneous processors is intimately bound to the development of supercomputers, especially, in the area of distributed heterogeneous supercomputing [9]. The execution times of the workloads can vary a lot depending on what type of processing units they are executed on. To this end, there are several existing works that focus on how to allocate applications to the appropriate processing units in order to achieve the best-case execution time, i.e., the shortest execution time [5, 10, 11]. In contrast, the work presented in this paper focuses on offloading the accelerator-intensive workloads, constrained by real-time requirements, e.g., deadlines on the response times of the workloads, to the available non-accelerated host processing units.

The heterogeneous processors considered in this paper consist of mainly two parts: (i) a host processing unit, CPU, and (ii) accelerator(s) that include GPUs and FPGAs, among others. There exist several research trends on how to tackle heterogeneous processors in real-time systems. One of the research trends is to explore the properties of accelerators in heterogeneous processors since a host processing unit is a well-studied single-core CPU. The existing works in this regard include TimeGraph [12], Gdev [13], the black-box method [14], to mention a few.

Another line of existing works targets resource management in the systems that use heterogeneous processing units. There are several works [15, 16, 17] that focus on splitting a task on accelerators for improving the schedulability. Moreover, TimeGraph [12], GPUSync [18], and the works by Kim et al. [19] and Biondi et al. [20] consider schedulability analysis of the systems that use heterogeneous processors. These works focus on accelerators, which obviously offer better (shorter) execution times of the compute-intensive workloads compared to the executions on the host processing units. On the other hand,

the work in this paper aims at mitigating the accelerator-intensive workload by efficiently offloading it to the non-accelerated host possessing units.

There exist several works that support server-based scheduling on single- and multi-core CPU(s) such as the constant bandwidth server (CBS) [6], total bandwidth server (TBS) [7], polling server (PS), sporadic server (SS) and deferrable server (DS) [8]. The work presented in this paper uses the DS. Some of the existing works also address the challenge of using the server-based scheduling in accelerators. For instance, the works in [21, 22] show that the server-based scheduling on accelerator(s) can improve the schedulability of the systems that use heterogeneous processors. In comparison to these works, the work presented in this paper uses the server-based scheduling in the host processing units instead of accelerators. The rationale behind this decision is that the proposed framework offloads the accelerator-intensive workloads to host processing units for efficiently utilizing their excess resources to assist the accelerators.

The idea of using alternative executions of parallel segments of real-time workloads is discussed in a few works [23, 4]. Baruah [23] applies conditional branching by using the if-then-else construct for two or more alternative executions of a workload. Moreover, a scheduling approach is introduced based on the conditional DAG model for reserving the necessary amount of computing resources. Tsog et al. [4] discuss a static allocation of real-time tasks using alternative execution of parallel segments of the tasks. Both works construct the fundamental of alternative executions of segments under real-time constraints. However, dynamic allocation of tasks using the alternative executions of parallel segments is missing from the state of the art. Provisioning of such an allocation is the main focus of the work presented in this paper.

11.3 System Model

We consider compute-intensive tasks that heavily require the use of accelerators such as GPUs. A periodic task, τ_i, is characterized by the tuple $\{S_i, D_i, T_i\}$, where S_i represents the set of finite sequence of execution segments of the task, D_i identifies the relative deadline of the task, and T_i represents the task's period. Furthermore, S_i consists of l sequential and parallel segments, $\{S_{i,1}, \ldots, S_{i,l}\}$, where $l \in \mathbb{N}$, i.e., it follows the traditional fork-join task model [24]. Regarding a parallel segment ($S_{i,j}, 1 < j < l$), we consider the model of alternative execution of parallel segments according to the work in [4]. Fig. 11.1 illustrates these execution segments as well as alternative executions of parallel segments. Traditionally, a sequential segment is executed on CPU as it can

only be executed in sequential manner. In contrast, a parallel segment can be executed either in a sequential/parallel manner.

In most cases, executing a parallel segment in parallel manner improves its execution time compared to executing it in a sequential manner. Hence, the developers tend to allocate parallel segments to GPU (or on CPU with multi-threading techniques) to execute them in parallel manner as shown in Fig. 11.1(a). This may not be efficient in all cases, especially when a parallel segment can be executed in a sequential manner if the GPU is busy serving other parallel segments. Fig. 11.1(b) illustrates the two alternatives to execute the parallel segments. In this paper, we consider that a parallel segment is executed in parallel manner only on GPU and in sequential manner only on CPU. This means that an instance of a parallel segment can be executed on GPU in parallel manner, while another instance of the same parallel segment can be executed on CPU in sequential manner.

The execution time $C_{i,j}$ of any segment $S_{i,j}$ of task τ_i is described by (11.1), where $h_{i,j}$ expresses the allocation decision of the segment $S_{i,j}$ to the processing units, and $C_{i,j}^{\texttt{CPU}}$ and $C_{i,j}^{\texttt{GPU}}$ are the execution times of the segment $S_{i,j}$ on CPU and GPU, respectively. In other words, $h_{i,j} = \texttt{CPU}$ and $h_{i,j} = \texttt{GPU}$ mean that the segment is allocated to `CPU` and `GPU`, respectively.

$$C_{i,j} = \begin{cases} C_{i,j}^{\texttt{CPU}} & \text{, if } h_{i,j} = \texttt{CPU} \\ C_{i,j}^{\texttt{GPU}} & \text{, if } h_{i,j} = \texttt{GPU} \end{cases} \tag{11.1}$$

We consider that the execution time C_i of task τ_i is the summation of the total execution times of its sequential segments on CPU and parallel segments on GPU, i.e.,

$$C_i = \sum_{\forall \tau_{i,j} | h_{i,j} = \texttt{CPU}} C_{i,j} + \sum_{\forall \tau_{i,j} | h_{i,j} = \texttt{GPU}} C_{i,j} \tag{11.2}$$

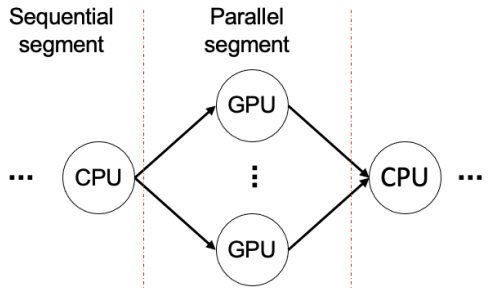
We define $C_i^{\texttt{CPU}}$ and $C_i^{\texttt{GPU}}$ as the total execution times of task τ_i on CPU and GPU respectively.

$$C_i^{\texttt{CPU}} = \sum_{\forall \tau_{i,j} | h_{i,j} = \texttt{CPU}} C_{i,j} \tag{11.3}$$

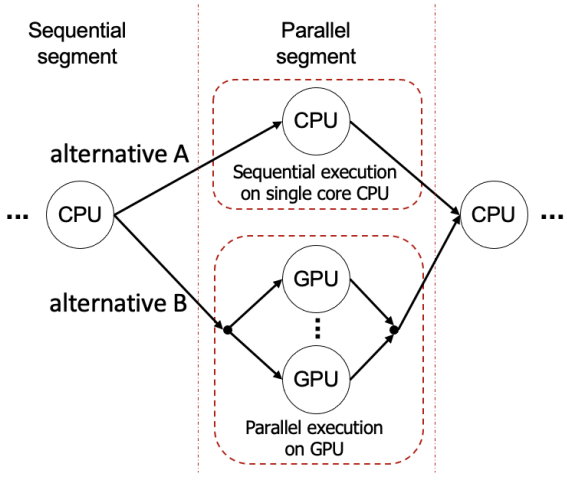
$$C_i^{\texttt{GPU}} = \sum_{\forall \tau_{i,j} | h_{i,j} = \texttt{GPU}} C_{i,j} \tag{11.4}$$

Hence, the execution time C_i can be represented as:

$$C_i = C_i^{\texttt{CPU}} + C_i^{\texttt{GPU}} \tag{11.5}$$



(a) Sequential and parallel segments



(b) Applying alternative executions to parallel segment

Figure 11.1: Difference between parallel segments applied with and without alternative executions.

Intuitively, the utilization of task τ_i is defined as:

$$U_i = (C_i^{\texttt{CPU}} + C_i^{\texttt{GPU}})/T_i \tag{11.6}$$

Note that the value of C_i depends upon how its segments are allocated. To distinguish if a task τ_i is CPU-heavy or GPU-heavy, we consider a metric, μ_i, which is defined as conversion ratio between $C_i^{\texttt{CPU}}$ and $C_i^{\texttt{GPU}}$, i.e,

$$\mu_i = C_i^{\texttt{CPU}}/C_i^{\texttt{GPU}} \tag{11.7}$$

The objective of this paper is to lower the utilization of highly-utilized accelerators and reduce the response times of middle and low priority tasks. The response time of task τ_i is expressed by the parameter R_i.

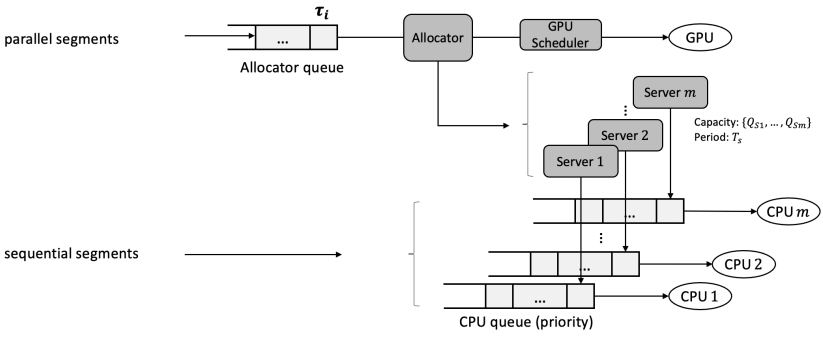


Figure 11.2: Proposed workload allocation framework based on server-based global scheduling of heterogeneous processors.

11.4 Proposed Workload Allocation Framework

The proposed allocation framework, illustrated in Fig. 11.2, considers the allocation of parallel and sequential segments of tasks differently. A given sequential segment is allocated to a CPU, while different sequential segments might be allocated to different CPUs, and these allocations are fixed. This means, a sequential segment that is ready to execute will be allocated to the appropriate CPU queue. The CPU queues follow the priority-based preemptive scheduling policy. The priorities are assigned according to the rate-monotonic algorithm. However, any other scheduling policy can be used in the CPU queues using the proposed allocation framework, e.g., earliest deadline first.

We introduce an allocator to efficiently manage the execution of parallel

segments. All parallel segments that are ready to execute are placed in the allocator queue as shown in Fig. 11.2. The allocator, in turn, decides which parallel segment to allocate to the GPU or CPU depending upon the availability of these compute units. As the first step, we consider the priority-based arbitration in the allocator queue such that the highest priority parallel segment is selected for allocation to the GPU or CPU. Other arbitration policies can also be applied such as the first-come-first-served policy. However, incorporation of the other policies within the proposed framework is left for the future work.

We assume that only one parallel segment executes on the GPU at a time and it is non-preemptive. This assumption is in line with the assumption of running one task on GPU at a time considered in the previous works [18, 19]. Furthermore, the policies for scheduling of tasks in the GPU proposed by Elliot et al. [18] and Kim et al. [19] can be used. The main difference between these policies is whether to keep the selected ready parallel segment in busy waiting or self suspension if the compute resources are busy. In this paper, we consider the self-suspension policy [19].

The proposed framework relies on the server-based dynamic scheduling to ensure that CPU resources are reserved to execute the parallel segments. For this purpose, we adopt the deferrable server (DS) with synchronization to m multi-core CPUs [8]. Other servers such as the Constant Bandwidth Server, Total Bandwidth Server, and Sporadic Server are also applicable. The DS, considered in this paper, is a set of m synchronized deferrable servers (SDS), which is characterized by an $(m+1)$-tuple $\{T_s, Q_{s_1}, Q_{s_2}, ..., Q_{s_m}\}$, where T_s is the common replenishment period of the SDSs and Q_{s_i} is the maximum capacity/budget of the i-th DS. We assume that each server has the highest priority in its respective CPU queue. Parallel segments are allocated to a single CPU server at a time. However, this restriction does not limit the execution of a parallel segment on different CPU servers. Thus, it is possible to dynamically allocate a given parallel segment to different servers throughout its execution.

11.5 Offloading Techniques

Traditionally, all parallel segments are allocated to accelerators in order to exploit the high-performance computing potential of the accelerators. However, this can have an adverse effect on the response times of the middle- and lower-priority tasks allocated to the accelerators when priority-based arbitration is used in the allocator queue. To address this, we consider to offload accelerator-intensive workloads, especially the middle and low priority parallel segments, to non-accelerated host processing units. However, the execution time of the parallel segments on non-accelerated devices is mostly longer than the execu-

tion time on the accelerator. That is, μ_i (see Equation 11.7) can reach up to 20 [25, 26] or even more as it is related to the type of CPU and GPU. In this paper, we consider the value of μ_i to be between 1.5 and 10. Note that due to the offloading strategy, the execution of parallel segments on non-accelerated processing units using servers can block the sequential segments of higher priority tasks.

There are two main concerns regarding the offloading techniques: (i) how to lower the overhead of allocation time while dynamically allocating the parallel segments to the compute units using the offloading techniques? and (ii) how to mitigate the impact of the allocation techniques on higher priority parallel segments? That is, how to reduce the time between the instant when the task is ready and the instant when it starts executing in the GPU? We introduce five offloading heuristic techniques for dynamic allocation of accelerator-intensive tasks in Sections 11.5.2- 11.5.6 in order to explore the potential impacts using these techniques. Optimal algorithms for allocating tasks to heterogeneous processing are NP-hard problems [27]. The main difference among these techniques is how they address and balance the above mentioned concerns. Note that we do not consider the genetic algorithms based techniques as their computation time is very high due to which they are not preferable for dynamic allocations [28, 4].

11.5.1 Baseline: Default Allocation Technique (DAT)

The DAT allocates all parallel segments to the GPU. Hence, this technique does not offload parallel segments to the non-accelerated host processing units. We consider the DAT as the baseline for comparative evaluations (discussed in the next section).

11.5.2 Naive Offloading Technique (NOT)

The NOT has no intelligent decision mechanism for offloading the parallel segments. This technique is intended to reveal the difference of computing performance between the host processing unit and accelerator devices. In short, this technique allocates parallel segments to the devices in the order of their available computing capacity. This technique can be expressed in the following steps.

- Step 1. Compute a list of available devices based on their computing capacity.

- Step 2. Allocate the parallel segment with the highest priority in the allocator queue to the device with the highest computing capacity in the list of available devices.
- Step 3. Repeat Step 2 until there are no available devices or no tasks in the allocator queue.

11.5.3 Min-min Fashioned Offloading Technique (MOT)

The min-min bin-packing approach is a well-known approach by packing rule and packing results [28]. The MOT offloading technique for parallel segments to non-accelerated devices is based on the min-min bin-packing approach. This technique is described by the following steps.

- Step 1. When there are available accelerators, allocate the highest priority parallel segments from the allocator queue to the accelerators.
- Step 2. If the allocator queue is empty then either all the allocator segments have been allocated to the accelerators and/or there is no ready parallel segment. If the allocator queue is not empty and there are no more accelerators available then go to next step.
- Step 3. Select the parallel segment with the highest priority in the allocator queue.
- Step 4. Calculate the summation of the current waiting time of the selected segment in the allocator queue and its execution time on the accelerator (GPU).
- Step 5. Calculate the difference between the summation (calculated in Step 4) and the execution time of the parallel segment on the CPU. If the difference is negative, allocate the segment to the CPU. Otherwise, allocate the segment to the GPU.
- Step 6. Repeat Step 1 if the allocator is not empty.

11.5.4 Speedup Classifier Based Technique (SCT)

The offloading technique is based on the speedup classifier bin-packing algorithm, which is studied in several works within the context of heterogeneous processors [5, 4]. As illustrated in Fig. 11.3, the different versions of this technique can exist based on selection of the classifier. In this paper, we select μ_i as the classifier. The SCT follows the following steps.

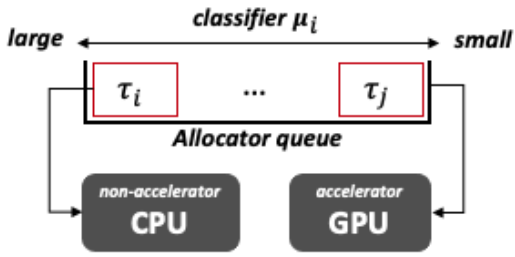


Figure 11.3: A sorted task queue based on speedup classifier.

- Step 1. Sort the parallel segments in the allocator queue according to the speedup classifier, which is μ_i in this case. This means, the segments with smaller μ_i will be stored on right side (close to the accelerator) and the segments with larger μ_i will be at placed at the left side (close to the non-accelerated device).
- Step 2. Allocate the parallel segments with the smallest μ_i to an accelerator with the higher computing capacity when there are available accelerator(s). Repeat this step until there is no available accelerator.
- Step 3. If there are parallel segments in the allocator queue and non-accelerated devices (CPUs) are available, allocate the parallel segment with the highest μ_i to the available CPU. Repeat this step until all CPUs are unavailable.

A disadvantage of this technique is that the parallel task segments that have the smallest and largest values of the μ_i classifier are prioritized compared to those with average values of the μ_i classifier. This means that a higher priority parallel segment with the average value of the μ_i classifier can be blocked by a lower priority parallel segment with a higher or lower value of the μ_i classifier.

11.5.5 Synchronized Servers Technique (SST)

This technique uses several servers to offload parallel segments to the non-accelerated devices. This technique can be described by the following steps.

- Step 1. Allocate the highest priority parallel segments to the available accelerators.
- Step 2. Repeat Step 1 until no more available accelerators.
- Step 3. If no parallel segment is currently using any server capacity, pick the parallel segment with the highest priority in the allocator queue. Otherwise, jump to Step 6.

- Step 4. Calculate the summation of the current waiting time of the selected segment in the allocator queue and its execution time on the accelerator (GPU).

- Step 5. Calculate the difference between the summation (calculated in Step 4) and the execution time of the parallel segment on the CPU. If the difference is negative, allocate the segment to the CPU. Otherwise, allocate the segment to the GPU. Select a parallel segment with the second-highest priority in the allocator queue and repeat Step 4.

- Step 6. In step 3, if there is a parallel segment that is currently using a server budget, then check whether the current server has a left over capacity to serve new requests. If yes, continue to run the parallel segment on the same server. Otherwise, switch the parallel segment to the next available server.

- Step 7. Repeat Step 6 until no more servers are available or the parallel segment completes its execution. In the former case, the execution of the parallel segment should be postponed until the start of the next period of the server and then repeat Step 6.

11.5.6 Efficient Offloading Technique (EOT)

We adapt the previous techniques to propose an efficient offloading technique. The EOT consists of the following steps.

- Steps 1-2. The first two steps are the same as that of Steps 1-2 in the SST.

- Step 3. If no parallel segment is currently using any server capacity, pick the parallel segment of the lowest priority task in the allocator queue as the lowest priority task tends to miss its deadline. Otherwise, jump to Step 5.

- Step 4. Allocate the selected parallel segment to a server, which runs on next CPU to the CPU that handles the sequential segment of the same task. Allocate the selected parallel segment to a server with the index of $i+1$ when the index of the server that serves the sequential segment of the same task is i. It is worth to note that we consider the server index of 1 instead of $i+1$ when i equals to m. This avoids to block the higher-priority sequential segments of other tasks assigned to the same server of the sequential segments of the select parallel segment's task.

- Step 5. In step 3, if there is a parallel segment that is currently using a server budget, then check whether the current server has a left over capacity to

serve new requests. If yes, continue to run the parallel segment on the same server. Otherwise, switch the parallel segment to the next available server.

- Step 6. Repeat Step 5 until no more servers are available or the parallel segment completes its execution. In the former case, the execution of the parallel segment should be postponed until the start of the next period of the server and then repeat Step 5.

11.6 Experimental Evaluation

A number of synthetic experiments are performed to evaluate a wide range of application parameters using the proposed framework and offloading techniques.

11.6.1 Task Set Generation and Experimental Setup

Table 11.1 illustrates the configuration that is used to generate the task sets. The task generation technique is based on the UUniFast algorithm [29]. The UUniFast is used in two ways to generate a task. First, using the given system utilization U, the UUniFast generates n random task utilizations for n tasks. For example, U_i represents the utilization for the task τ_i. The simulator initializes the following basic parameters of a task:

- period T_i,
- number of parallel segments
- ratio of the length of parallel and sequential segments, and
- conversion ratio of parallel segments μ_i.

The UUniFast generates the random length of parallel and sequential segments based on each task utilization and the total execution times of parallel and sequential segments. The total execution times of parallel and sequential segments are derived from the task utilization, the period and the ratio of the length of parallel and sequential segments. In order to generate an alternative of parallel segments (i.e., the sequential execution of parallel segments), the conversion ratio of parallel segments μ_i is used. The value of μ_i is randomly selected between 1.5 and 10, which is in line with the existing experimental studies [25, 30, 31].

In each experiment, the synthetic experiment simulator is run until the schedulability of the task set converges to the given condition. The simulator is executed minimum 100 times with 3,000,000 cycles to get the variance

Table 11.1: Initial configuration of task set generation.

Parameters	**Values**
Number of CPU cores (N_p)	4, 8
Number of GPUs	1
Number of tasks (n)	$[N_p, 10N_p]$
System utilization (U)	0.5-2
Task period and deadline ($T_i = D_i$)	[30, 500]ms
Ratio of parallel to sequential segments length (C_i^{GPU}/C_i^{CPU})	[0.1, 3]
Number of parallel segments per task (η_i)	1-3
Conversion ratio of parallel segments (μ_i)	[1.5-10]
Server utilization (Q_{s_i}/T_s)	[20-40]%
The common period of the SDSs (T_s)	[30, 500]ms

of schedulability of the task sets. Based on the variance value, the simulator continues to execute until 200 times in fast-converging cases and 500 times in slow-converging cases.

11.6.2 Offloading Techniques

We perform comparative evaluation of five offloading techniques (NOT, MOT, SCT, SST and EOT), presented in Section 11.5, with respect to the DAT technique (Section 11.5). The DAT acts as the baseline technique as it does not support offloading the parallel segments to non-accelerated processing units. Note that the NOT, MOT, SCT are focused on executing parallel segments on a CPU server, while the SST and EOT consider to execute parallel segments on multiple synchronized CPU servers. The input to the offloading techniques is a set of tasks that must fulfill the input requirements of the task model (Section 11.3) such as segments' deadlines. The execution on CPU is preemptible and self-suspending, while the execution on GPU is non-preemptive. The execution traces in the experiments are evaluated until the hyperperiod (least common multiple of all periods) of the task set, which takes 3,000,000 cycles.

11.6.3 Evaluation Results

This subsection presents the evaluation results of the following three groups of experiments.

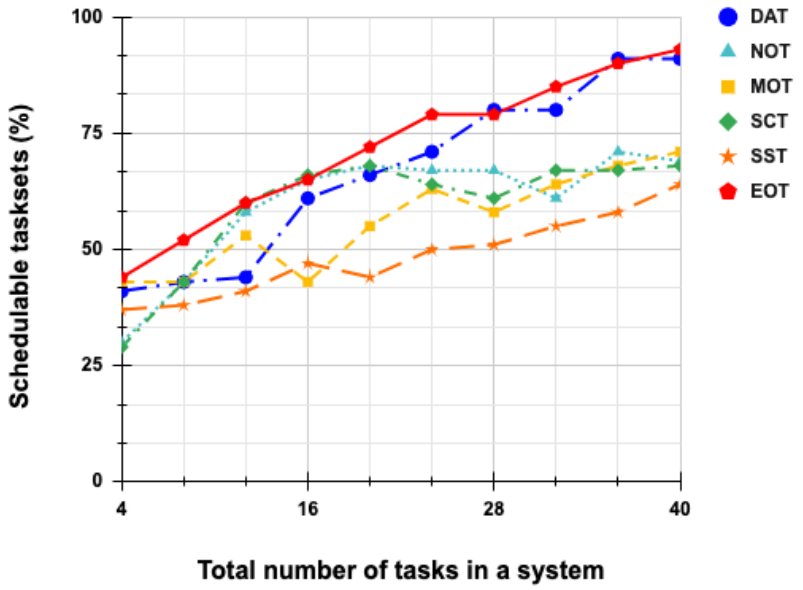


Figure 11.4: Schedulable task sets w.r.t. the total number of tasks in the system with 4 CPU cores and 1 GPU.

- *Experiment A* focuses on the comparative evaluation of the offloading techniques.
- *Experiment B* performs a detailed comparative evaluation of the EOT and the baseline technique (DAT) using different experimental setups.
- *Experiment C* focuses on evaluating the time to perform the offloading using various offloading techniques.

The experiments simulate 4 or 8 CPU cores and 1 or 2 GPUs. The implementation of the experiments can be extended to more than 8 CPU cores and more than 2 GPUs. However, such extensions are left for future work.

Experiment A

Fig. 11.4 illustrates the performance of the five offloading techniques in terms of the percentage of schedulable task sets with respect to the total number of tasks in the system (with utilization equals to 1). The NOT, MOT, SCT and SST show similar trend of schedulable task sets compared to the DAT and EOT, although there are some big differences between these two sets of techniques.

This confirms that these four techniques (NOT, MOT, SCT and SST) focus on only part of properties for mitigating accelerator-intensive loads. Among these techniques, the SCT shows better results. The reason is that the SCT allocates a parallel segment with highest conversion ratio μ_i to CPU and the lowest conversion ratio to GPU. So, the SCT reorders the priority of the tasks and allocates them to the appropriate processing units. However, this can considerably change the order of the execution.

The DAT and EOT show the best results among the lot. More specifically, the EOT shows slightly better results than the DAT. It can be concluded that the use of accelerators is one of the best choices with respect to the schedulability of the task sets. However, the results of the EOT indicate that the improvement can be achieved by better utilizing all computing resources in the heterogeneous processors. Since the DAT and EOT are the best performing techniques, we focus only on them while performing a detailed comparative evaluation in the next simulation experiment.

Experiment B

In this simulation experiment, we consider how the EOT handles the schedulability of the task sets. Fig. 11.5 illustrates the percentage of schedulable task sets with respect to the system utilization under the EOT and DAT techniques. The results indicate that the EOT performs better than the DAT, although the relative improvement is small. In the case of system utilization $U = 1$, the EOT with both 12 and 24 tasks shows the maximum improvements of 16% and 8% under dynamic scheduling.

Fig. 11.6 describes how the conversion ratio of parallel segment to sequential segment (i.e., μ_i) manipulates the schedulability of task sets. The horizontal axis expresses the maximum value that μ can take. We see that the EOT performs better than the DAT until the value of μ_i reaches 10. This means, our proposed framework can perfectly handle the accelerator-intensive workloads even if the accelerators compute 9 times faster than the host device.

Fig. 11.7 depicts how the variation in the ratio of the length of parallel and sequential segments, C_i^{GPU}/C_i^{CPU}, influences the schedulability of the task sets under the EOT and DAT techniques. Both techniques show a similar trend in the results; however, the EOT performs slightly better than the DAT. This confirms again that the use of accelerators for parallel segments is the best choice generally. There exists a slightly small improvement window between the essential use of accelerators and the total use of heterogeneous processors. When the length of parallel segments is equal to 75% of the entire execution time of the tasks, we observed up to 16% improvement in the schedulability of

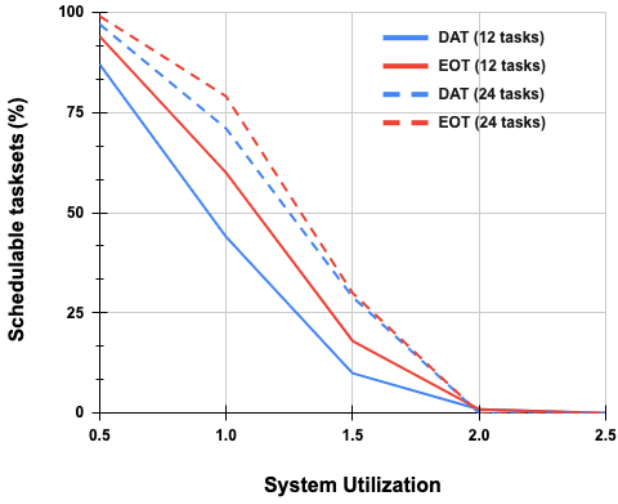


Figure 11.5: Percentage of schedulable task sets with respect to the system utilization under the EOT and DAT techniques.

the task sets with respect to the DAT.

Experiment C

In this experiment, we extract the execution traces of the simulation of schedulable task sets. Table 11.2 shows the mean time to perform the offloading. The DAT shows the best mean time to perform offloading (22.33s), while the SCT gives the worst results (37.19s). The MOT and NOT more or less take the same time to perform the offloading. The reason is that these techniques select a parallel segment to allocate to the CPU, while the SCT needs to create a list, ordered by the μ_i, before selecting to allocate a parallel segment to the CPU. The SST shows the best results among the NOT, MOT, SCT and SST. In the SST, only one task can use the servers, which explains why the SST shows the best results among the four techniques.

Although the DAT performs the best in terms of the time to perform the offloading (22.33s), the EOT also performs nearly best (23.52s). This is because the EOT leverages the advantages of the NOT, MOT, SCT, and SST techniques.

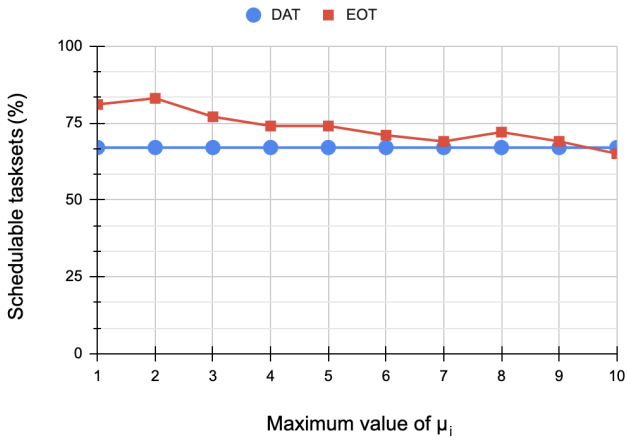


Figure 11.6: Percentage of schedulable task sets with respect to variations in maximum value of μ_i under the EOT and DAT techniques.

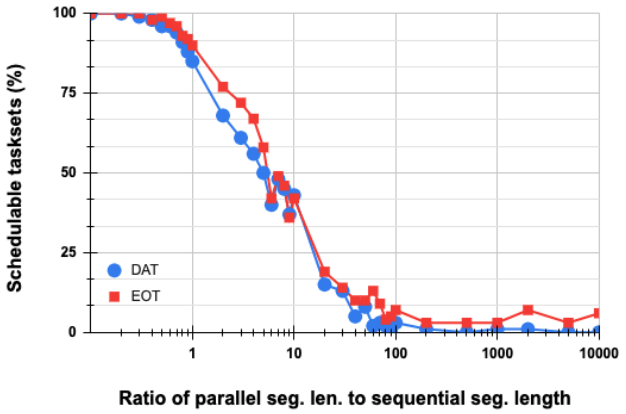


Figure 11.7: Percentage of schedulable task sets with respect to the ratio of the length of parallel and sequential segments μ_i under the EOT and DAT techniques.

Table 11.2: Mean time to perform the offloading under various offloading techniques.

No.	Offloading Techniques	Experiment mean time
1	DAT	22.33s
2	NOT	34.49s
3	MOT	33.15s
4	SCT	37.19s
5	SST	28.75s
6	EOT	23.52s

11.7 Conclusion

This paper presented a novel and efficient framework to allocate parallel segments of the accelerator-intensive workloads to non-accelerated processing units in the CPU-GPU heterogeneous processors under dynamic scheduling. Within the context of this framework, the paper proposed five offloading heuristic techniques for mitigating the accelerator-intensive workloads. The use of accelerators for parallel segments of the tasks is crucial and is an excellent choice in most cases. In this regard, the paper showed that a considerable improvement can be achieved if all processing units in the heterogeneous processors are better utilized, i.e., by efficiently offloading some of the parallel segments to non-accelerated compute units. The evaluation results indicate that one of the proposed techniques, namely the efficient offloading technique, can achieve up to 16% improvement in schedulability of the task sets under dynamic scheduling compared to the non-offloading technique. It is worth to note that we have not optimized the SDS in this paper. In other words, the optimization of the SDS can improve the proposed framework and we consider it as future work. Another area of future work will focus on improvement of the proposed framework and offloading techniques using the pipelining technique. Furthermore, to expand our investigation to the use of multiple GPUs and/or different capacity of processing units entails another line of future work.

Acknowledgements

The work presented in this paper was supported by the the Swedish Knowledge Foundation via the DPAC and HERO projects, and the Swedish Governmental Agency for Innovation Systems (VINNOVA) via the DESTINE, PROVIDENT and INTERCONNECT projects.

Bibliography

[1] J. Huang, "NVIDIA CEO Keynote," *GPU Technology Conference*, Oct. 2017.

[2] L. L. Bello, R. Mariani, S. Mubeen, and S. Saponara, "Recent advances and trends in on-board embedded and networked automotive systems," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 2, pp. 1038–1051, 2018.

[3] C. Margiolas and M. F. O'Boyle, "Portable and transparent software managed scheduling on accelerators for fair resource sharing," in *Proceedings of the 2016 International Symposium on Code Generation and Optimization*, 2016, pp. 82–93.

[4] N. Tsog, M. Becker, F. Bruhn, M. Behnam, and M. Sjödin, "Static allocation of parallel tasks to improve schedulability in cpu-gpu heterogeneous real-time systems," in *IECON 2019-45th Annual Conference of the IEEE Industrial Electronics Society*, vol. 1. IEEE, 2019, pp. 4516–4522.

[5] Y. Wen, Z. Wang, and M. F. O'boyle, "Smart multi-task scheduling for opencl programs on cpu/gpu heterogeneous platforms," in *2014 21st International conference on high performance computing (HiPC)*. IEEE, 2014, pp. 1–10.

[6] L. Abeni and G. Buttazzo, "Integrating multimedia applications in hard real-time systems," in *Proceedings 19th IEEE Real-Time Systems Symposium (Cat. No. 98CB36279)*. IEEE, 1998, pp. 4–13.

[7] M. Spuri and G. C. Buttazzo, "Efficient Aperiodic Service Under Earliest Deadline Scheduling," in *RTSS*, 1994, pp. 2–11.

[8] H. Zhu, S. Goddard, and M. B. Dwyer, "Response time analysis of hierarchical scheduling: The synchronized deferrable servers approach," in *32nd Real-Time Systems Symposium*. IEEE, 2011, pp. 239–248.

[9] R. F. Freund and D. S. Conwell, "Superconcurrency: A form of distributed heterogeneous supercomputing," NAVAL OCEAN SYSTEMS CENTER SAN DIEGO CA, Tech. Rep., 1991.

[10] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *2009 IEEE international symposium on workload characterization (IISWC)*. Ieee, 2009, pp. 44–54.

[11] P. Czarnul and P. Rościszewski, "Optimization of execution time under power consumption constraints in a heterogeneous parallel system with gpus and cpus," in *International Conference on Distributed Computing and Networking*. Springer, 2014, pp. 66–80.

[12] S. Kato, K. Lakshmanan, R. Rajkumar, and Y. Ishikawa, "Timegraph: Gpu scheduling for real-time multi-tasking environments," in *2011 USENIX Annual Technical Conference (USENIX ATC 11)*, 2011, pp. 17–30.

[13] S. Kato, M. McThrow, C. Maltzahn, and S. Brandt, "Gdev: First-class gpu resource management in the operating system," in *2012 USENIX Annual Technical Conference (USENIX ATC 12)*, 2012, pp. 401–412.

[14] N. Otterness, M. Yang, S. Rust, E. Park, J. H. Anderson, F. D. Smith, A. Berg, and S. Wang, "An evaluation of the NVIDIA TX1 for supporting real-time computer-vision workloads," in *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2017, pp. 353–364.

[15] C. Basaran and K.-D. Kang, "Supporting preemptive task executions and memory copies in GPGPUs," in *2012 24th Euromicro Conference on Real-Time Systems*. IEEE, 2012, pp. 287–296.

[16] S. Kato, K. Lakshmanan, A. Kumar, M. Kelkar, Y. Ishikawa, and R. Rajkumar, "Rgem: A responsive gpgpu execution model for runtime engines," in *2011 IEEE 32nd Real-Time Systems Symposium*. IEEE, 2011, pp. 57–66.

[17] E. Rossi, M. Damschen, L. Bauer, G. Buttazzo, and J. Henkel, "Preemption of the partial reconfiguration process to enable real-time computing with FPGAs," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 11, no. 2, pp. 1–24, 2018.

[18] G. A. Elliott, B. C. Ward, and J. H. Anderson, "Gpusync: A framework for real-time gpu management," in *2013 IEEE 34th Real-Time Systems Symposium*. IEEE, 2013, pp. 33–44.

[19] H. Kim, P. Patel, S. Wang, and R. R. Rajkumar, "A server-based approach for predictable gpu access control," in *2017 IEEE 23rd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2017, pp. 1–10.

[20] A. Biondi, A. Balsini, M. Pagani, E. Rossi, M. Marinoni, and G. Buttazzo, "A framework for supporting real-time applications on dynamic

reconfigurable FPGAs," in *2016 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2016, pp. 1–12.

[21] S. Kato, K. Lakshmanan, Y. Ishikawa, and R. Rajkumar, "Resource sharing in GPU-accelerated windowing systems," in *2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE, 2011, pp. 191–200.

[22] Y.-S. Chen, H. C. Liao, and T.-H. Tsai, "Online real-time task scheduling in heterogeneous multicore system-on-a-chip," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 1, pp. 118–130, 2012.

[23] S. Baruah, "Resource-efficient execution of conditional parallel real-time tasks," in *European Conference on Parallel Processing*. Springer, 2018, pp. 218–231.

[24] C. Maia, M. Bertogna, L. Nogueira, and L. M. Pinho, "Response-time analysis of synchronous parallel tasks in multiprocessor systems," in *Proceedings of the 22Nd International Conference on Real-Time Networks and Systems*, 2014, pp. 3–12.

[25] F. C. Bruhn, N. Tsog, F. Kunkel, O. Flordal, and I. Troxel, "Enabling Radiation Tolerant Heterogeneous GPU-based Onboard Data Processing in Space," *CEAS Space Journal*, vol. 12, no. 4, pp. 551–564, 2020.

[26] N. Tsog and M. Larsson, "Time Predictability of GPU Kernel on an HSA Compliant Platform," 2016.

[27] S. K. Baruah, "Task Partitioning Upon Heterogeneous Multiprocessor Platforms," in *IEEE real-time and embedded technology and applications symposium*. Citeseer, 2004, pp. 536–543.

[28] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen *et al.*, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *Journal of Parallel and Distributed computing*, vol. 61, no. 6, pp. 810–837, 2001.

[29] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, no. 1-2, pp. 129–154, 2005.

[30] N. Tsog, M. Behnam, M. Sjödin, and F. Bruhn, "Intelligent data processing using in-orbit advanced algorithms on heterogeneous system architecture," in *2018 IEEE Aerospace Conference*. IEEE, 2018, pp. 1–8.

[31] N. Tsog, M. Sjödin, and F. Bruhn, "Advancing on-board big data processing using heterogeneous system architecture," in *ESA/CNES 4S Symposium 4S 2018, 28 May 2018, Sorrento, Italy*, 2018.