Using Docker in Process Level Isolation for Heterogeneous Computing on GPU Accelerated On-Board Data Processing Systems

Nandinbaatar Tsog[1], Mikael Sjödin[1], Fredrik Bruhn[1,2]

[1]Mälardalen University
Box 883, 721 23, Västerås, Sweden
Mail: {nandinbaatar.tsog, mikael.sjodin, fredrik.bruhn}@mdh.se

[2] Unibap AB (publ)
Svartbäcksgatan 5, 753 20 Uppsala, Sweden
Mail: f@unibap.com

Abstract: The technological advancements make the intelligent on-board data processing possible on a small scale of satellites and deep-space exploration spacecraft such as CubeSats. However, the operation of satellites may fall into critical conditions when the on-board data processing interferes strongly to the basic operation functionalities of satellites. In order to avoid these issues, there exist techniques such as isolation, partitioning, and virtualization. In this paper, we present an experimental study of isolation of on-board payload data processing from the basic operations of satellites using Docker. Docker is a leading technology in process level isolation as well as continuous integration and continuous deployment (CI/CD) method. This study continues with the prior study on heterogeneous computing method, which improves the schedulability of the entire system up to 90%. Based on this heterogeneous computing method, the comparison study has been conducted between the non-isolated and isolated environments.

1. INTRODUCTION

The role of intelligent on-board data processing for in-situ information value extraction is significant in space systems such as earth and atmospheric observation satellites, CubeSat constellations, and deep-space exploration spacecraft. For example, the next generation earth and atmospheric observation satellites require the sensors with high sensitivity and resolution, while, the sensors generate more data than the cross-links or downlink can handle. However, the operation of satellites may fall into critical conditions when many on-board operations and processes, including the basic operation functionalities of satellites, interfere with each other strongly. Thus, in this paper, we present an experimental study of isolation of on-board payload data processing from the basic operation functionalities of satellites in a common platform using Docker[1].

In this study, we consider heterogeneous computing for on-board data processing which is explored by using radiation tolerant on-board processing platforms[2][3]. These platforms take advantage of both commercial off-the-shelf products (COTS) and a new computer architecture, Heterogeneous System Architecture (HSA)[4], which reduces the data transfer bottle-neck by enabling coherent virtual shared memory between different processing units (e.g. CPU, GPU FPGA). Moreover, HSA simplifies the programming process of the heterogeneous computing software. The platforms inherit the advantages of the platform which is commercialized by Unibap AB and selected by NASA for high-performance on-board processing for the "HyTI" thermal hyperspectral mission[5].

Space systems require to satisfy different types of limitations such as SWaP (size, weight, and power) and radiation hardness. As being as real-time systems, timing constraints are

required as well. Further, space systems require to adopt continuous integration and continuous development/deployment (CI/CD) and automated testing in order to improve their development process and quality of the products. Thus, in this paper, we consider using Docker for isolation as well as CI/CD.

Contribution: This study continues prior work[6][3] of a heterogeneous computing method, which improves the schedulability of the entire system up to 90%. This heterogeneous computing method is well known in high performance computing[7] and supercomputers. However, to the best of our knowledge, there is a lack of prior research studies of this method on real-time embedded systems. Therefore, in this paper, we conduct an experimental study of this method in Docker isolated environments.

Organization: The rest of this paper is organized as follows. Section 2 and 3 introduce related work and necessary background information. Section 4 presents our system model and system architecture. Section 5 introduces case study and experiments and reports experimental evaluation. Lastly, Section 6 concludes this paper.

2. RELATED WORK

FPGA accelerated onboard computer is one of the main representers of heterogeneous processors used in satellites, as FPGAs are strong against in the radiation-hardened environments. For example, FPGAs are considered for on-board processing in an advanced imaging system[8] and real-time cloud detection[9] since FPGAs are good for image and video processing. On the other hand, adopting GPUs in the context of space was not appreciated, since the concern of GPUs about the radiation-hardened environments was unacceptable. Recently, adopting GPU onto the on-board computer is increasing[10, 2, 3]. Furthermore, Kosmidis et al.[11] presents benchmarking results of GPU accelerated platforms for on-board data processing.

In this paper, we focus on systems using both CPU and GPU in the context of space. As we see in the survey[12], the study of CPU-GPU systems as heterogeneous processors and heterogeneous computing is very active. Especially, the impact of GPU in supercomputers is significant as the most of supercomputers adopt GPU. However, greedy use of specific processing unit may worsen the entire system and not all the applications are suitable for parallelism[7]. Therefore, there are methods using the nature of OpenCL[13, 7] that makes heterogeneous computing easier. Because, in OpenCL, it is possible to prepare the different kernels of the same execution part on the different devices. In this paper, we perform experiments with tasks which can be executed whether on CPU or GPU.

Although there are fewer studies on GPU in real-time systems compared to high-performance computing, there exist several works which tackle with real-time properties of GPU accelerated systems. Shinpei et al. presented TimeGraph[14] and RGEM[15] along with zero-copy I/O processing for low-latency GPU computing[16]. In addition, the works[17, 18] consider worst-case timing scenarios in GPU accelerated real-time systems. Most of these works consider solving the limitation of early existing GPU hardware and device drivers such as a zero-copy technique for accelerator memories and splitting tasks into smaller chunks in order to perform preemption. However, these limitations will be solved by the latest new technologies such as unified memory, zero-copy and preemption technologies in CUDA[19] and Heterogeneous System Architecture (HSA)[4, 3].

3. BACKGROUND

3.1 Real-time system

A real-time system is a system that responds to external events within a finite and required time. In other words, both accuracy and timeliness of the response of the system are a crucial factor for the system. Thus, in real-time systems, we focus on worst-case scenarios rather than best- and average-case scenarios, while high performance computing focuses on them. Real-time systems can be divided into a hard, firm and soft real-time systems according to their timing constraints. Hard real-time systems must pass entire timing constraints. Any small deadline miss can result in failure which leads to a fatality and/or big cost damage. For example, an airbag system in cars is a hard real-time system, and any deadline miss can end up a loss of human life. On the other hand, soft real-time system can accept one or more deadline misses although it affects to its quality of service. Systems such as music player and car window opening control system are soft real-time systems since any deadline misses of these systems would not end up with catastrophic results. A firm real-time system is between hard and soft real-time systems. In satellites, a system including basic functionalities can be considered as hard real-time system, while payloads can be considered as soft real-time systems.

3.2 Docker

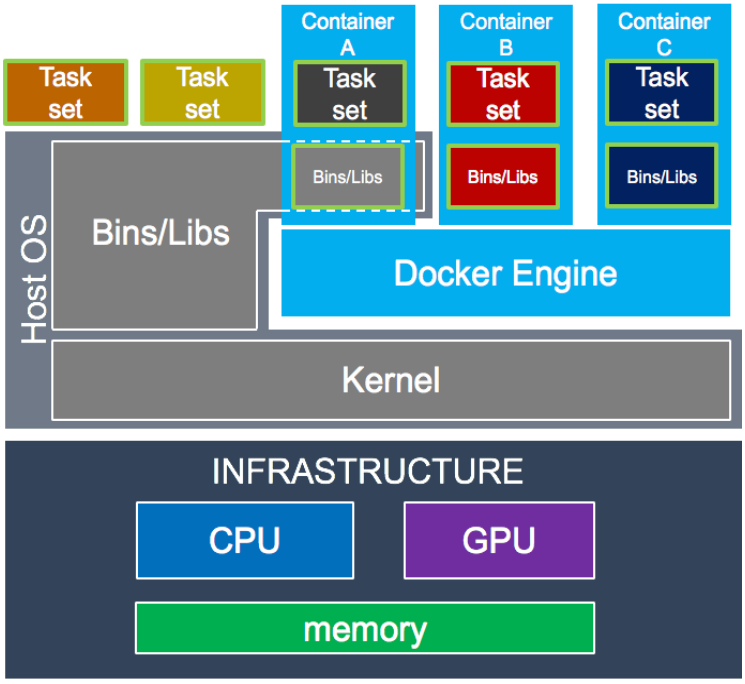

Figure 1. System architecture

To achieve the independent conduct of both on-board data processing and basic operations of satellites as real-time systems, we use Docker [1] for the process level isolation. Docker is a container and runs on top of the kernel of the host operating system (OS). In this paper, we consider Linux as the host OS. Docker uses the basic Linux features, namespaces and cgroups, in order to bring isolation for systems[1] (see Figure 1). Contrary to virtual machines, Docker containers share the same host OS and it starts running as an application in the host OS while virtual machines should first boot entire OS

installed in the virtual machines. This feature enables to keep the entire system smaller and to perform the quick startup of isolated containers.

Furthermore, containers may include different Linux distributions as long as their kernels match with the kernel included in the host OS. Hence, it is possible that the containers can use libraries and binaries of the host (*container A* in Figure 1) or other distributions (*containers B and C* in Figure 1). By using namespaces feature, a container runs under a different root from the root of the host OS, i.e., it helps to isolate container tasks from tasks running on the host OS. On the other hand, cgroups[20] helps to limit the resource usage of containers as it works as constant bandwidth server. In addition to limiting CPU and memory usages, a container can restrict system calls, e.g. terminating network access for security reason and so on. A container does not need to consume the resources when it is idle while virtual machine still consumes the resources for its OS.

4. SYSTEM MODEL AND ARCHITECTURE

4.1 System model

We consider a system *S* (Eq. 1), that deals with a task set Γ, that consists of *n* tasks, i.e., $\Gamma = \{\tau_n\}$. Each task should be assigned to one of a benchmarking environment set *E* (Eq. 2). The benchmarking environment set *E* consists of a host environment, E^{Host} or E_0, and *m* pieces of container environments, $E_m^{Container}$ or $\{E_1, .., E_m\}$.

$$S =< \Gamma, E > \quad (1)$$

$$E = E^{Host} \cup \{E_m^{Container}\} = \{E_0, \dots, E_m\} \quad (2)$$

We further define that each benchmarking environment should be able to manage *k* pieces of CPU cores, $\{P_k^{CPU}\}$, and memory limit, M_i.

$$E_i =< \{P_k^{CPU}\}, M_i >, i = \{0, \dots, m\} \quad (3)$$

4.2 System architecture

The system architecture is shown in Figure 1. This system employs a HSA compliant accelerated processing unit (APU) maintained in a system-on-chip (SoC). The APU (A10-8700P) adopts Radeon™ R6 GPU, which consists of 6 compute units, and 2 CPUs (each CPU has 2 cores). 2 cores of each CPU share one 128KB L1 cache and all four cores share 1MB L2 cache. Total processing capability of the system reaches up to 614 GFLOPS. There is no memory copy between memories of CPU and GPU since the APU is HSA compliant and they share 8GB DDR3 memory. Above this platform, Ubuntu 16.04 distribution is running as a host operating system (OS) including Linux kernel 4.15 and ROCm 1.9 driver. ROCm is an open source driver for heterogeneous computing. On top of the host OS, we run Docker 1.13.1.

5. EVALUATION

5.1 Case study

Two type of applications, basic operation functionalities of satellite and workload, are conducted to represent on-board data processing platforms of satellites. We deal with machine learning and computer vision applications as a workload on different containers in order to study interference between host-and-container and container-and-container. A simple matrix calculation along with the basic processes of the host OS are considered as the basic operation functionalities of satellites.

As shown in Figure 1, there are 3 types of containers; the container A uses binaries/libraries of the host OS and the containers B and C are based on different binaries/libraries. The allocations of the applications to the containers are shown as follows; matrix multiplication, TensorFlow[21] and Harris Corner Detector are on the container A, B and C, respectively.

TensorFlow is an open source machine learning library created by Google. In this paper, we use TensorFlow 1.12 together with ROCm 1.9 driver. Harris Corner Detector is a computer vision algorithm, which is provided by open-source computer vision library OpenCV[22]. All the containers are able to perform the applications on both CPU and GPU.

5.2 Experiments

The following 3 experiments are considered to identify the process level isolation by Docker. Due to space limitations, we only give simple explanations of the experiments.

- ***Experiment A***. The aim of this experiment is to understand the basic mechanism of the Docker container and the interference factors to the host environment.
- ***Experiment B***. In this experiment, we deal with the interference between host-and-container applications. The comparison study of the execution time of the different applications on host and container environments is performed.
- ***Experiment C.*** The interference between different container environment is the key in this experiment.

5.3 Results

Due to the space limitation, we only present interesting results. In Table 1, we see the results of Experiments A and B. The mean and measurement based worst case response time (WCRT) of computing only AlexNet and Harris Edge Detector are {7.875s; 8.036s} and {1.649s; 1.87s}, respectively. Furthermore, the WCRTs of executing both AlexNet and Harris Edge Detector together in host environment are 8.104s and 1.897s. These results are similar to the WCRTs of executing AlexNet in host environment and Harris Edge Detector in Docker. This means that we do not confirm any interferences from Docker to the systems.

Execution time [s] (measured)	**AlexNet with TensorFlow**	**Harris Edge Detector**

	Mean	WCRT	Mean	WCRT
Stand Alone	7.875	8.036	1.649	1.87
Together in Host Environment	7.906	8.104	1.821	1.897
AlexNet in Host & Harris in Docker	7.929	8.113	1.837	1.893

Table 1. Summary of experimental results

A part of the results from experiment C is shown in Table 2. Table 2 expresses the execution times of AlexNet algorithm on different environments with different data set. The mean values of AlexNet running both in host environment (12.348s) and Docker containers (12.349s) are shorter than the mean value of AlexNet running stand-alone (12.355s), although their WCRTs (12.374s and 12.371s) is longer than the WCRT of stand-alone (12.366s). This explains that we confirm no interferences from Docker container to the systems. Moreover, we do not confirm any interference to the CPU-GPU communication.

Execution time [s] (measured)	**AlexNet with TensorFlow**	
	Mean	WCRT
Stand Alone	12.355	12.366
Together in host environments	12.348	12.374
Together with payloads using Docker	12.349	12.371

Table 2. CPU-GPU communication

Finally, although it is obvious, we can state that starting up systems on Docker is much faster compared to booting the system including its host OS. This strongly supports adopting Docker in CI/CD required systems.

6. CONCLUSION

Through the experiments, we confirm that Docker helps to perform CI/CD without decreasing the computing potential. The results show that the tasks allocated in the different environments with the independent resources (CPU cores and memory) would not interfere with each other. Moreover, Docker would not worsen the computation potential when the tasks, which allocated on the different containers, share the resources. Finally, we see that our heterogeneous computing method fits with Docker for the process level isolation.

7. ACKNOWLEDGMENTS

The work presented in this paper was partially supported by the Swedish Knowledge Foundation via the research profile DPAC. The authors 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. We would also like to express our sincere

gratitude to Dr. Moris Behnam for his great knowledge in real-time embedded systems.

8. REFERENCES

[1] Dirk Merkel, "Docker: lightweight Linux containers for consistent development and deployment", Linux Journal, Volume 2014 Issue 239, March 2014

[2] Fredrik Bruhn et al., "Introducing radiation tolerant heterogeneous computers for small satellites", 2015 IEEE Aerospace Conference, Big Sky, USA, 2015

[3] Nandinbaatar Tsog et al., "Intelligent Data Processing using In-Orbit Advanced Algorithms on Heterogeneous System Architecture", 2018 IEEE Aerospace Conference, Big Sky, USA, 2018

[4] HSA Foundation, "HSA Foundation - ARM, AMD, Imagination, MediaTek, Qualcomm, Samsung, TI". Available online at: http://www.hsafoundation.com (accessed 6 November 2018)

[5] Robert Wright, Thomas George et al., "Hyperspectral Thermal Imager (HyTI)", University of Hawaii and Saraniasat Inc. Available online at: https://esto.nasa.gov/files/solicitations/INVEST_17/ROSES2017_InVEST_A49_awards.html#george (accessed 3 August and 6 November 2018)

[6] Nandinbaatar Tsog et al., "Using Heterogeneous Computing on GPU Accelerated Systems to Advance On-Board Data Processing", The European Workshop on On-Board Data Processing 2019 (OBDP2019), ESTEC, Amsterdam, Netherlands, 2019

[7] Yuan Wen et al., "Smart multi-task scheduling for OpenCL programs on CPU/GPU heterogeneous platforms", 21st International Conference on High Performance Computing (HiPC), IEEE, Dona Paula, India, 2014

[8] Charles D. Norton et al., "An evaluation of the Xilinx Virtex-4 FPGA for on-board processing in an advanced imaging system", 2009 IEEE Aerospace Conference, Big Sky, USA, 2009

[9] John Williams et al., "FPGA-based cloud detection for real-time onboard remote sensing", 2002 IEEE International Conference on Field-Programmable Technology, Hong Kong, China, 2002

[10] R.L. Davidson and Christopher P. Bridges, "Adaptive multispectral GPU accelerated architecture for Earth Observation satellites", 2016 IEEE International Conference on Imaging Systems and Techniques (IST), Chania, Greece, 2016

[11] Leonidas Kosmidis et al., "Embedded GPU benchmarking for High-Performance On-board Data Processing", The European Workshop on On-Board Data Processing 2019 (OBDP2019), ESTEC, Amsterdam, Netherlands, 2019

[12] S Mittal and J.S. Vetter, "A Survey of CPU-GPU Heterogeneous Computing Techniques", ACM Computing Surveys (CSUR), Volume 47, 2015

[13] P Czarnul and P Rosciszewski, "Optimization of Execution Time under Power Consumption Constraints in a Heterogeneous Parallel System with GPUs and CPUs", Distributed Computing and Networking , ICDCN 2014, Coimbatore, India, 2014

[14] Shinpei Kato et al., "TimeGraph: GPU Scheduling for Real-time Multi-tasking Environments", USENIX Conference on USENIX Annual Technical Conference (USENIXATC), Portland, USA, 2011

[15] Shinpei Kato et al., "RGEM: A Responsive GPGPU Execution Model for Runtime Engines", 32nd IEEE Real-Time Systems Symposium (RTSS), Vienna, Austria, 2011

[16] Shinpei Kato et al., “Zero-copy I/O processing for low-latency GPU computing”, ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS), Philadelphia, USA, 2013

[17] Glenn Elliott and Jim Anderson, “Globally Scheduled Real-time Multiprocessor Systems with GPUs”, Real-Time Systems, Volume 48, 2012

[18] Hyoseung Kim et al., “A server-based approach for predictable GPU access control”, 23rd IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), Hsinchu, Taiwan, 2017

[19] Mark Harris, “Unified Memory for CUDA Beginners”, Available online at: https://devblogs.nvidia.com/unified-memory-cuda-beginners/ (accessed Oct 16, 2018)

[20] Rami Rosen, “Resource management: Linux kernel Namespaces and cgroups”, Available online at: https://sites.cs.ucsb.edu/~rich/class/old.cs290/papers/lxc-namespace.pdf (accessed 5 May 2019)

[21] TensorFlow. Available online at: https://www.tensorflow.org/ (accessed 31 Jan 2019)

[22] OpenCV. Available online at: https://opencv.org/ (accessed 31 Jan 2019)