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

By Nandinbaatar TSOG,[1)] Saad MUBEEN,[1)] Mikael SJÖDIN,[1)] and Fredrik BRUHN[1),2)]

[1)]*Mälardalen University, Västerås, Sweden*
[2)]*Unibap AB (Publ.), Uppsala, Sweden*

(Received July 12th, 2019)

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

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* with flight heritage and selected by NASA for high-performance on-board data processing for the HyTI thermal hyperspectral mission.[9)]

* https://unibap.com/

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[5,7,8] 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.

1.2. Organization

In the rest of this paper, we provide needed related work in Section 2. Section 3 presents detailed explanations about real-time systems, heterogeneous computing as well as advanced applications in satellite. A description of our system model is discussed in Section 4. Section 5 reports experimental evaluation. Lastly, we conclude in Section 6.

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)[5,10] 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).[3,20,21]

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-then-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,[32] Digital Video Broadcasting - Satellite - Second Generation (DVB-S2) transport stream[30] and real-time cloud detection.[31] 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.[2,33] In this paper, we conduct experimental observations of utilizing GPU in the context of space.

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.

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 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.

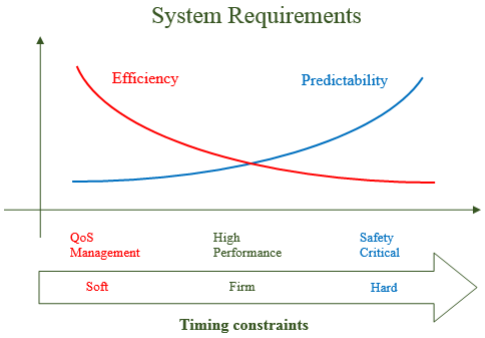

Fig. 1. A real-time system requirements.[†]

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.

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 (Nvidias 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.

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[‡], Torch[§], Caffe[¶], and Microsoft Cognitive Toolkit (CNTK)[‖]. Of course, one could also employ unsupervised learning approaches, such as a (deep) auto-encoder

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

[‡] https://www.tensorflow.org/

[§] http://torch.ch/

[¶] https://caffe.berkeleyvision.org/

[‖] https://github.com/Microsoft/CNTK

scheme. Having a trained model, one can execute it on the satellites 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 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).

4. System Model

We consider a task model which is described by Fork-Join task model. As shown in Fig. 2, the parallel segment (starts with a Fork and ends with a Join) of tasks could be executed in two manners, parallel and sequential.[8] 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.

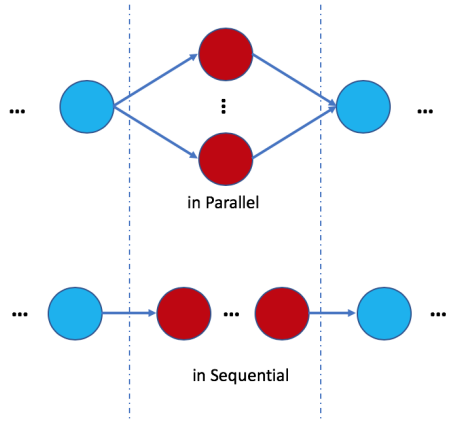

Fig. 2. Execution manner of parallel segment of Fork-Join task model.

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

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

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 1 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

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.

5.1. Algorithms

In this paper, we consider two on-board algorithms, namely Anisotropic Diffusion[40] and Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics (LULESH).[43] The different combinations of the algorithms are used in the different purposes of the experiments.

5.1.1. 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 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. 1, 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)]

5.1.2. 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[**††], which is ported to the different environments such as MPI, OpenMP, OpenCL and C++AMP.

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 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.

5.3. Experimental observations

Observation 1: Compiler vs Computing potential. First, we consider the relation between computing potential with respect to the different compiler versions. Test application 1 is compiled by three different versions (GCC5.4.0, GCC6.2.1[‡‡] and GCC 7.1.0[§§]) 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"[¶¶] 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.

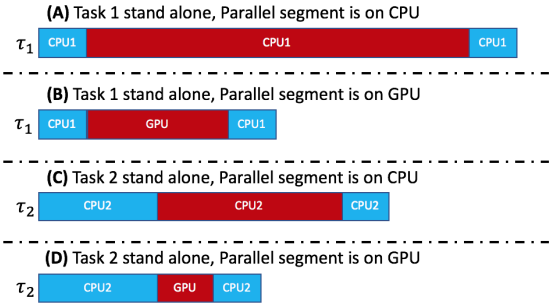

Fig. 3. Execution manners of a stand-alone parallel segment in terms of considering worst case execution time (WCET). Blue bars express sequential segments of tasks that should be executed sequentially only. Red bars are parallel segments of tasks. Parallel segments could be executed either in sequential 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).

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 3 and 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 3-(A), the parallel segment of task τ_1 is allocated to CPU1. On the other hand, in Fig3-(B), we can see that this parallel segment is allocated to

** https://github.com/LLNL/LULESH
†† https://github.com/AMDComputeLibraries/ComputeApps

‡‡ Untrusted PPA:*ppa:jonathonf/gcc-6.2*
§§ Untrusted PPA: *ppa:jonathonf/gcc-7.1*
¶¶ 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.

Table 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

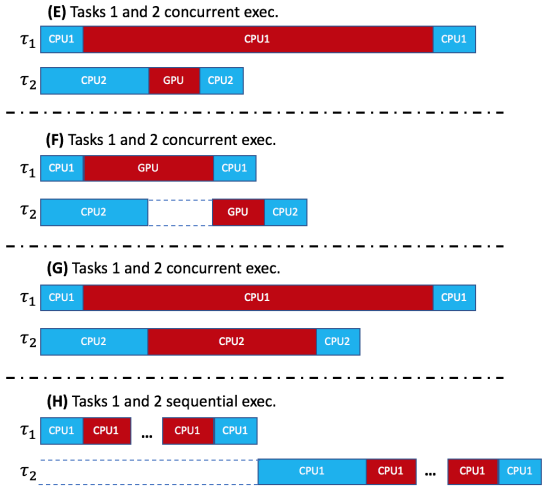

Fig. 4. Execution manners of concurrent parallel segments in terms of considering response time (RT). The combinations of the tasks, Task 1 (τ_1) and Task 2 (τ_2), described in Fig. 3 are considered. White space bordered with dot-lines expresses no execution of tasks, e.g., τ_2 in cases (F) and (H). 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).

GPU. Stand alone executions of the tasks are illustrated in Fig. 3. Allocations of concurrent executions of tasks τ_1 and τ_2 are shown in Fig. 4. Here, we consider that τ_1 and τ_2 implement LULESH and Anisotropic Diffusion algorithms, respectively.

5.4. Evaluation and results

Observation 1: The results of running the test application 1 are shown in Tables 2, 3, 4, 5, 6 and 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 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 specific 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.

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 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 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 Joules. The energy consumption of the preparation part of CPUCalc and

*** 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 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***	131.265	1	2.470	135.468	1	3.028

Table 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

Table 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 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 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

Table 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

OMPCalc decrease by half of HSACalc to approximately 13 and 7.5 Joules for non-optimised and optimised versions, respectively. From Figure 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 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

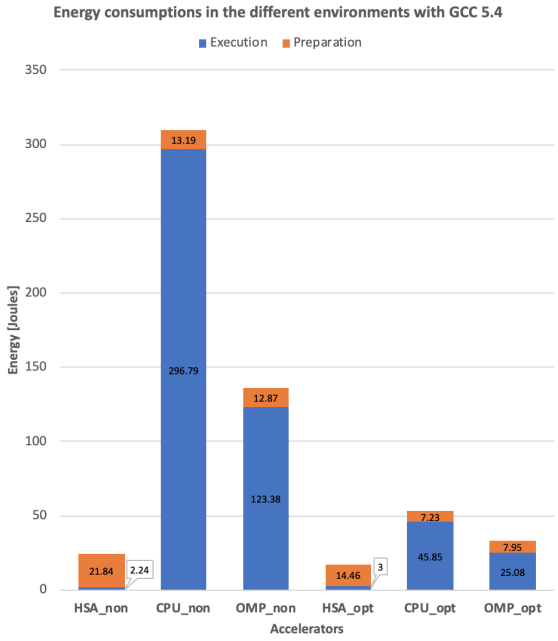

Fig. 5. Comparison of energy consumption between different programming manners with optimised and non-optimised codes.

Table 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 8 and 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)($E_{system} = 150.32Joules$) and (G)($E_{system} = 152.47Joules$) 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 ($R_1 = 15.78sec$ for τ_1 and $R_2 = 32.1sec$ for τ_2) and consumes most energy ($E_{system} = 382.41Joules$). 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.

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 3 and 5 would not be improved without his advice and valuable discussions.

AMD, Ryzen, Radeon and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

References

1) Master, T.: Consortium for Execution of Rendezvous and Servicing Operations (CONFERS), DARPA, https://www.darpa.mil/program/consortium-for-execution-of-rendezvous-and-servicing-operations, (accessed Apr 22, 2019).
2) Bruhn, F., Brunberg, K., Hines, J., Asplund, L., and Norgren., M.: Introducing Radiation Tolerant Heterogeneous Computers for Small Satellites, 2015 IEEE Aerospace Conference, pp. 1–10, IEEE, 2015.
3) Tsog, N., Behnam, M., Sjödin, M., and Bruhn, F.: Intelligent Data Processing Using In-orbit Advanced Algorithms on Heterogeneous System Architecture, 2018 IEEE Aerospace Conference, pp 1–8, IEEE, March 2018.
4) Mittal, S. and Vetter, J.S.: A Survey of CPU-GPU Heterogeneous Computing Techniques, *ACM Computing Surveys (CSUR)*, **47** (2015),

pp. 69:1–69:35.
5) Wen, Y., Wang, Z., and O'boyle, M. F. P: Smart Multi-task Scheduling for OpenCL Programs on CPU/GPU Heterogeneous Platforms, 21st International Conference on High Performance Computing (HiPC), pp. 1–10, IEEE, 2014.
6) HSAFoundation : HSA Foundation - ARM, AMD, Imagination, MediaTek, Qualcomm, Samsung, TI, http://www.hsafoundation.com/, (accessed Apr 22, 2019)
7) Tsog, N., Sjödin, M., and Bruhn, F.: Using Heterogeneous Computing on GPU Accelerated Systems to Advance On-Board Data Processing, European Workshop on On-Board Data Processing (OBDP2019), ESA, CNES and DLR, 2019.
8) Tsog, N., Becker, M., Bruhn, F., Behnam, Sjödin, M.: Static Allocation of Parallel Tasks to Improve Schedulability in CPU-GPU Heterogeneous Real-Time Systems, IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society, pp. 4516–4522, 2019.
9) Wright, R., George, T., et al.: Hyperspectral Thermal Imager (HyTI), NASA, https://esto.nasa.gov/files/solicitations/INVEST_17/ROSES2017_InVEST_A49_awards.html#george, (accessed August 6, 2018)
10) Czarnul, P. and Rosciszewski, P.: Optimization of Execution Time under Power Consumption Constraints in a Heterogeneous Parallel System with GPUs and CPUs, ICDCN 2014: Distributed Computing and Networking, pp 66–80, 2014.
11) Kato, S., Lakshmanan, K., Rajkumar, R., and Ishikawa, Y.: TimeGraph: GPU Scheduling for Real-time Multi-tasking Environments, *USENIX Conference on USENIX Annual Technical Conference (USENIXATC)*, (2011), pp 2–2.
12) Kato, S., Lakshmanan, K., Kumar, A., Kelkar, M., Ishikawa, Y., and Rajkumar, R.: RGEM: A Responsive GPGPU Execution Model for Runtime Engines, 32nd IEEE Real-Time Systems Symposium (RTSS), pp. 57–66, 2011.
13) Kato, S., McThrow, M., Maltzahn, C., and Brandt, S.: Gdev: First-class GPU Resource Management in the Operating System, USENIX Conference on Annual Technical Conference (USENIXATC), pp. 37–37, 2012.
14) Kato, S., Aumiller, J., and Brandt, S.: Zero-copy I/O Processing for Low-latency GPU Computing, ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS), pp. 170–178, 2013.
15) Elliott, G. A. and Anderson, J. H.: Globally Scheduled Real-time Multiprocessor Systems with GPUs,*Real-Time Systems*, **48** (2012), pp. 34–74.
16) Elliott, G. A., Ward, B.C., and Anderson, J.H.: GPUSync: A Framework for Real-Time GPU Management, 34th IEEE Real-Time Systems Symposium (RTSS), pp. 33–44, 2013.
17) Kim, H., Patel, P., Wang, S., and Rajkumar, R. R.: A Server-based Approach for Predictable GPU Access Control, 23rd IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), pp. 1–10, 2017.
18) Kim, H., Patel, P., Wang, S., and Rajkumar, R. R.: A Server-based Approach for Predictable GPU Access with Improved Analysis, *J. Systems Architecture*, **88** (2018), pp. 97–109.
19) Harris, M.: Unified Memory for CUDA Beginners, 2017, , https://devblogs.nvidia.com/unified-memory-cuda-beginners/, (accessed Oct 16, 2018).
20) HSA Foundation: Heterogeneous System Architecture, http://www.hsafoundation.com/, (accessed Oct 16, 2018).
21) Tsog, N., Sjödin, M., and Bruhn, F.: Advancing On-Board Big Data Processing Using Heterogeneous System Architecture, ESA/CNES 4S Symposium 4S, 2018.
22) Maia, C., Bertogna, M., Nogueira, L., and Pinho, L. M.: Response-Time Analysis of Synchronous Parallel Tasks in Multiprocessor Systems, 22nd International Conference on Real-Time Networks and Systems (RTNS), pp. 3:3–3:12, 2014.
23) Saifullah, A., Ferry, D., Li, J., Agrawal, K., Lu, C., and Gill, C. D.: Parallel Real-Time Scheduling of DAGs, *IEEE Trans. Parallel and Distributed Systems*, **25**, (2014), pp. 3242–3252.
24) Ullman, J. D.: NP-complete Scheduling Problems, *J. Comput. Syst. Sci.*, **10** (1975), pp. 384–393.
25) Melani, A., Bertogna, M., Bonifaci, V., Marchetti-Spaccamela, A., and Buttazzo, G. C.: Response-Time Analysis of Conditional DAG Tasks in Multiprocessor Systems, 27th Euromicro Conference on Real-Time Systems (ECRTS), pp. 211–221, 2015.
26) Sanjoy, B.: Resource-Efficient Execution of Conditional Parallel Real-Time Tasks, Euro-Par 2018: Parallel Processing, pp. 218–231, 2018.
27) Liu, Q. and Luk, W.: Heterogeneous Systems for Energy Efficient Scientific Computing. Reconfigurable Computing: Architectures, Tools and Applications, Springer, pp. 64-75, 2012.
28) Aydin, H., Melhem, R., Mossé, D., and Mejía-Alvarez, P.: Power-aware Scheduling for Periodic Real-time Tasks, *IEEE Trans. Computers*, **53** (2004), pp. 584–600.
29) Rofouei, M., Stathopoulos, T., Ryffel, S., Kaiser, W., and Sarrafzadeh, M.: Energy-aware High Performance Computing with Graphic Processing Units, Workshop on Power Aware Computing and System, 2008.
30) Varsha, R., Arora, R., Ram, T.V.S., and Patel, A.: Design and Implementation of DVB-S2 Transport Stream for Onboard Processing Satellite, 19th International Symposium on VLSI Design and Test, pp. 1–6, 2015.
31) Williams, J.A., Dawood, A.S., and Visser, S.J.: FPGA-based cloud detection for real-time onboard remote sensing, 2002 IEEE International Conference on Field-Programmable Technology, pp. 110–116, 2002.
32) Norton, C.D., Werne, T.A., Pingree, P.J., and Geier, S.: An Evaluation of the Xilinx Virtex-4 FPGA for On-board Processing in an Advanced Imaging System, 2009 IEEE Aerospace Conference, IEEE, pp. 1–9, 2009.
33) Davidson, R.L. and Bridges, C.P.: Adaptive Multispectral GPU Accelerated Architecture for Earth Observation Satellites, 2016 IEEE International Conference on Imaging Systems and Techniques (IST), pp. 117–122, 2016.
34) Buttazzo, G. C., *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, Springer, 2011.
35) OpenMP Architecture Review Board: OpenMP Application Programming Interface, Version 4.5, November 2015.
36) Khronos OpenCL Working Group: The OpenCL Specification, Version 1.2, Document Revision 19.
37) NVIDIA: NVIDIA CUDA C Programming Guide, Version 4.2, April 16, 2012.
38) NVIDIA: CUDA C PROGRAMMING GUIDE, Version PG-02829-001_v9.2, Design Guide, July 2018.
39) Microsoft: C++ AMP : Language and Programming Model, Version 1.0, August 2012.
40) Perona, P. and Malik, J.: Scale-space and Edge Detection Using Anisotropic Diffusion, *IEEE Trans. Pattern Analysis and Machine Intelligence* **12** (1990), pp. 629–639.
41) Schwarzkopf, A., Kalbe, T., Bajaj, C., Kuijper, A., and Goesele, M.: Volumetric Nonlinear Anisotropic Diffusion on GPUs, SSVM 2011, LNCS 6667, pp. 62-73, 2012.
42) Lopes, D.S.: A Set of Filters That Perform 1D, 2D and 3D Conventional Anisotropic Diffusion, 2007, http://se.mathworks.com/matlabcentral/fileexchange/14995-anisotropic-diffusion–perona—malik-, (accessed April 22, 2019).
43) Lawrence Livermore National Laboratory: LULESH, https://computation.llnl.gov/projects/co-design/lulesh, (accessed April 21, 2019).
44) Sedov, L. I.: Similarity and Dimensional Methods in Mechanics, (3d ed.; Moscow: State Publishing House), pp. 225–237, 1954.