The International Journal of Advanced Manufacturing Technology
https://doi.org/10.1007/s00170-023-11252-0

ORIGINAL ARTICLE

Efficient GPU Cloud architectures for outsourcing high-performance processing to the Cloud

Víctor Sánchez-Ribes[1] · Antonio Maciá-Lillo[1] · Higinio Mora[1] · Antonio Jimeno-Morenilla[1]

Received: 4 October 2022 / Accepted: 9 March 2023

Abstract
The world is becoming increasingly dependant in computing intensive applications. The appearance of new paradigms, such as Internet of Things (IoT), and advances in technologies such as Computer Vision (CV) and Artificial Intelligence (AI) are creating a demand for high-performance applications. In this regard, Graphics Processing Units (GPUs) have the ability to provide better performance by allowing a high degree of data parallelism. These devices are also beneficial in specialized fields of manufacturing industry such as CAD/CAM. For all these applications, there is a recent tendency to offload these computations to the Cloud, using a computing offloading Cloud architecture. However, the use of GPUs in the Cloud presents some inefficiencies, where GPU virtualization is still not fully resolved, as our research on what main Cloud providers currently offer in terms of GPU Cloud instances shows. To address these problems, this paper first makes a review of current GPU technologies and programming techniques that increase concurrency, to then propose a Cloud computing outsourcing architecture to make more efficient use of these devices in the Cloud.

Keywords GPU · Cloud computing · High-performance processing · Offloading computation

1 Introduction

Cloud computing is one of the technologies that are shaping today's world. This computation model has collaborated in the development of the information society, and is being used extensively in many areas, transforming and creating new business models, where the challenges of traditional systems are being overcome. In this regard, businesses must make use of this technology to stay competitive in a globalized market. Cloud systems provide extensive benefits to enterprises and users with respect to ubiquitous access to data, resource management and information processing.

✉ Antonio Maciá-Lillo
a.macia@ua.es

Víctor Sánchez-Ribes
vsr37@alu.ua.es

Higinio Mora
hmora@ua.es

Antonio Jimeno-Morenilla
jimeno@ua.es

[1] Department of Computer Science Technology and Computation, University of Alicante, Alicante, Spain

Distributed Cloud Architectures, such as "Edge" and other intermediate layers, are envisioned as a key technology that will model the future of Information and Communication Technologies (ICT) [1]. This infrastructure is being enriched by adding specific purpose computing devices, such as Graphic Processing Units (GPU). Their use is specially useful in external processing architectures where computing intensive tasks of specialized fields such as CAD/CAM [2, 3] or Artificial Intelligence (AI) [4] are being offloaded to the Cloud, where the use of GPUs provides better performance [5, 6], and allows commodity hardware to perform these operations. This brings superior parallel computing capabilities that tackle new computing intensive needs [7]. However, this is at the expense of new challenges, specially in the manufacturing industry [8], where small businesses are prone to design incorrect Cloud strategies, underutilizing the advantages that bring this paradigm.

Cloud providers offer GPUs as a part of the virtual infrastructure, usually using the *Pass-Through* method, where the entire GPU is being used exclusively by a single virtual instance. This strategy, although valid for big enterprises, may not be optimal for smaller ones, such as CAD/CAM clients of manufacturing industry, online video games and

Published online: 26 May 2023

AI [9, 10], where individual applications may underutilize the full potential provided by GPUs. Other method is the GPU virtualization (vGPU). This is based on a time multiplexing strategy to provide concurrent GPU access to multiple independent applications [11–13]. GPU virtualization via this method presents inefficiencies, as sequential execution of Kernels that do not make use of all GPU resources leaves those extra resources unused. However, new technologies such as Nvidia Multi-Instance GPU (MIG) [14], may improve this scenario in the near future.

The need to provide an efficient method for using these specialized processing architectures in the Cloud for small-medium workloads has opened up a research line focusing on optimizing the execution of specific code on processing devices. Following this trend, the aim of this paper is to explore new ways to take advantage of the specialized devices and improve the capabilities of GPUs by reviewing and comparing available parallel programming techniques, and propose optimal Cloud architecture configuration and development of applications to improve the degree of parallelization of specialized processing devices, serving as a basis for their increased exploitation in the Cloud for small and medium-sized enterprises.

This paper is organized as follows. Besides the introduction, we sketch the overview of the current state of research in computing outsourcing of intensive specialized computations to the Cloud, and current state of GPU usage in the Cloud. Then, a review of GPU technologies that improve performance of applications by increasing the concurrency and overall GPU usage is done. After this, a Cloud architecture that uses these techniques is proposed, as a way of using the GPU more efficiently. Next, experiments that test and show the benefits and characteristics of the techniques and the presented architecture are shown. And last, conclusions are drawn from the research done in this paper.

2 Current state of research

This section makes a review on the current state of the art of processing outsourcing architectures and technologies, and the current state of GPU usage and virtualization in the Cloud, in order to draw the border of knowledge on the most relevant issues concerned with the objectives of our proposal.

2.1 Processing outsourcing

The externalization process to offload part of the computing load to the Cloud allows to increase the capabilities of devices and other computing platforms at the Edge. In this area, mobile Cloud Computing paradigm was initially designed to extend the battery life of IoT things and mobile devices [15]. However, this trend has evolved as a way to give enhanced performance and access to high-performance computing resources [16–18].

Outsourcing the processing of computing workloads using a distributed Cloud architecture has some technical challenges, as the numerous offload possibilities to Edge and Cloud infrastructures introduce new variables that must be taken into account [16]. The Edge paradigm is based on the use of computation platforms in the same location as where the data is being gathered, giving increased processing power and data protection to the system [19]. In this line, a distributed computation model in the Cloud has the physical location of the servers running the platform as a part of its definition [20].

The evolution of the Cloud paradigm becomes more important with the recent introduction of 5G networks, which provide greater connectivity and data exchange capabilities [21], and the proliferation of new Internet of Things applications [22] such as smart lighting [23] or Internet of Vehicles (IoV) whose purpose is the offloading of processing on servers (Edge) for content decoding tasks [24] to the closest available servers in order to obtain shorter response times.

In this regard, managing the outsourcing process to the Cloud is one of the main challenges to be addressed in building new software of IoT, mobile devices and other distributed systems [25]. Usually, the most common case is the use of the CPU as the final execution platform for the outsourced workload.

Nowadays, there are an increasing number of proposals trying to apply this paradigm for offloading workloads with specific needs to specialized and massive parallel devices such as GPUs [26]. These proposals aim to optimize or accelerate the computation of highly mathematical primitives taking advantage of the performance of these specialized computing platforms [27, 28]

2.2 GPU usage in Cloud architectures

GPUs are a widely used tool in order to obtain better performance in intensive computation of specialized fields, as they provide high computing power for geometric and matrix computations, as their Single Instruction Multiple Threads (SIMT) architecture gives the best results with data parallelism.

Small and medium enterprises of the traditional manufacturing sector make use of CAD/CAM and Artificial Intelligence applications that use GPU hardware [29].

CAD/CAM applications that involve intensive computation tasks rely on GPUs for the development of efficient software tools that can upgrade their performances [29]. A few examples of CAD/CAM applications that make use

of this kind of hardware are Real-time co-design with 3D model visualization and rendering, computation algorithms for solid models, real-time engineering simulation and analysis, collision detection, and data exchange between different parties [29]. There are several works on these fields. In [30], an optimization method that uses GPUs to improve the toolpath computation and reduce the average machining time is presented. There are works that show the use of GPUs in cloth simulation [31], cloth collision detection [32], cloth animation [33, 34], and authoring and simulating of cloth or garment patterns [35]. In [36], a parallelized genetic algorithm that is implemented on the GPU finds the best model orientation that minimizes the building time and supporting area. In [37], a genetic algorithm that is based on CUDA achieves an optimum part deposition orientation.

In Artificial Intelligence (AI), algorithms are being sped up using GPUs [38, 39]. Moreover, AI applications make extensive use of the GPU to speed up the training process [40, 41]

As these applications started to use processing outsourcing architectures, there is an inherent need to include these specialized devices in Cloud architectures. To meet this demand, Cloud Providers started offering GPUs as a part of their available services. In general, a GPU can be offered in a Cloud service in two ways:

- Exclusive dedication of the GPU to the entire platform (*pass-through*)
- GPU virtualization

The *pass-through* method is valid for companies with large compute needs capable of leveraging this powerful hardware, but this method wastes GPU computing power when used with small applications that are not able to fully occupy the capabilities of a large GPU platform, but still benefit from them with better performance.

On the other hand, GPU virtualization aims to share a physical GPU concurrently between different virtual machine clients in a secure manner, where hardware resources can be split and provisioned to different virtual GPUs, depending on the needs of the specific applications. Traditionally, this is done sharing the utilization of the graphic card by spreading the time across several instances or processes through a round robin scheduling [11–13, 42]. Although sharing by time multiplexing helps to reduce GPU idle times, this can also be inefficient, as kernels may not utilize all GPU resources during its execution window. To accomplish a more efficient GPU virtualization, NVIDIA launched its MIG technology for the new Ampere architecture. This technology allows for hardware supported virtualization, where each virtual GPU instance can run concurrently in a secure manner, having separated allocated memory and computing resources, with assured Quality of Service (QoS) and fault isolation [14].

A study has been conducted on popular Cloud service providers to know which methods have available for a Cloud GPU instance. Table 1 shows the results. A total of ten service providers have been reviewed. Almost all of them offer GPUs in their Cloud servers via the *pass-through* method, which grants exclusive use of the GPU hardware to a single client. These servers are meant for applications with high computing needs. On the contrary, only two of them (Azure and Tencent Cloud) offer smaller and cheaper virtual GPUs to adjust for the needs of smaller applications, with very limited options and configurations. Azure only allows its virtual GPU instances to use the Windows operating system [43], and Tencent only has vGPU option available in its GN7 instances, with the option of having 1/2 or 1/4 of the resources of a NVIDIA Tesla T4 graphics card [44].

2.3 Findings

From the previous review, the following general findings are drawn:

- Hardware virtualization is of great importance for the right deployment and optimal utilization of powerful specific computing platforms such as GPUs in the Cloud.
- The use of GPUs in Cloud platforms is underused due to inefficiencies in the methods they are currently provided.
- Programming on graphics cards using CUDA can become inefficient due to serial execution and programming overheads.

The rest of this paper will cope with these problems by reviewing and proposing different technologies, programming models and architectures.

Table 1 Cloud providers support for GPU instances

PROVIDER	PASS-THROUGH	VGPU
AWS	yes	no
Azure	yes	yes
Google Cloud	yes	no
IBM Cloud	yes	no
Alibaba Cloud	yes	no
Oracle Cloud	yes	no
Tencent Cloud	yes	yes
OVHcloud	yes	no
Digital Ocean	no	no
Linode	yes	no

3 Technologies involved

As shown by the study on Cloud providers, GPU virtualization in the Cloud is, to this day, not developed enough. The virtualization of this computing platform is still not properly resolved, due to its sophisticated architecture and the nature of the applications that make use of it. As a result, Cloud systems have limitations in the shared use of this resource.

As it is not possible to provide virtual Cloud instances with a virtual GPU with the exact resources required by the concrete application, it is left to the Cloud client to optimize the deployments for the physical GPUs available at an application level. In this line, this section is going to analyze the common architecture used for processing outsourcing of GPU accelerated computations, and the main techniques that can be used to increase the GPU usage, allowing to obtain better performance at a lower cost.

3.1 Computing outsourcing architecture

The architecture shown in Fig. 1 represents a basic model of reference of a Cloud architecture for computing outsourcing that uses a GPU to accelerate intensive specific processing. This architecture is based on the infrastructure offered by main Cloud providers to increase the efficiency of intensive computing primitives by the parallelization of those operations using a GPU. This architecture operates by receiving computing primitives of clients such as IoT devices, computing those primitives, and returning the results to the client.

In this environment, total execution time of a petition (T) is defined by adding the communication time ($T_{\text{Communication}}$) to the processing time ($T_{\text{Processing}}$), as shown in the formula. δ_T is an error threshold to ensure a minimum QoS.

$$T = T_{\text{Communication}} + T_{\text{Processing}} + \delta_T \tag{1}$$

Although communication time is intrinsic to the time it takes to process a petition in a Cloud architecture, and has important aspects to maintain QoS such as transmission speed or distance between the client and the server, the scope of this paper is limited to improve the computation time. In the rest of the paper, techniques are presented with the purpose of reducing $T_{\text{Processing}}$ at an application level.

3.2 Techniques at the application level

The following technologies can be used to optimize GPU accelerated applications increasing GPU usage by increasing the parallelism. By default, kernel calls from within an application are executed sequentially [45]. CUDA Streams is a technique to increase concurrency of a process that launches multiple CUDA kernels [45, 46]. It is based on a set of queues (Streams) where kernel execution requests can be enqueued by the host in a non-blocking fashion [47]. Each queue runs the enqueued kernels sequentially, but kernels of different queues can run concurrently in the GPU device [46]. The programmer has to explicitly assign every kernel call to a desired Stream. Kernel calls that are not assigned to a Stream are allocated in the default stream, or Stream 0 [46], being this the reason why by default, kernels run in a serial manner.

Although all kernels can be scheduled in different Streams to be run in parallel, an important aspect of GPU kernel concurrency is that there are physical limits that affect how kernels can run concurrently. In [45] a theoretical model is presented to predict kernel concurrency behavior. The amount of threads that can be scheduled every running round depends on several factors, such as the number of multiprocessors available, the number of blocks and threads of the kernel, or the amount of shared memory [45].

On a process level, it is also possible to obtain concurrency, with NVIDIA MPS. MPS is a implementation of the CUDA API that uses a runtime client–server architecture to achieve GPU concurrency between multiple processes [48]. Although this technology was developed with the objective of optimizing GPU accelerated MPI applications, it is useful in a variety of systems, as it allows for an increase in performance when single processes do not make full use of the resources present in the device.

As told by NVIDIA [48], MPS gives the benefits of an increase in GPU usage between different CUDA processes, lower video memory usage, and a reduction of the amount of context changes.

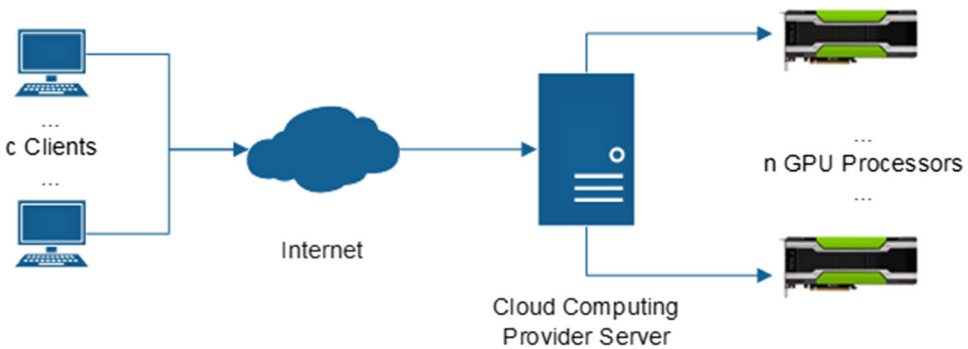

Fig. 1 Reference model of Cloud computing outsourcing architecture

NVIDIA graphics cards implement this technology since Kepler architecture was introduced. However, since the introduction of Volta architecture, new characteristics in the line of increasing security and isolation between concurrently running processes were added. Nowadays, MPS offers these characteristics in terms of security:

- Execution resource provisioning: Client contexts can be set to only use a portion of the available threads. This can be used as a classic QoS mechanism to limit available compute bandwidth.
- Memory protection: Applications are provided with a private video memory space, completely isolated from external processes.
- Error containment: When a fatal exception happens, it is reported to all processes that are running in the GPU in that moment, without any indication of which process has generated the exception. The GPU then is temporally blocked from accepting new processes until all processes that were running exit.

Due to these features, MPS cannot be used as way of virtualizing the GPU, as the error containment system does not provide enough isolation between processes to allow for a complete virtualization.

4 Proposed Cloud architecture

With the technologies explained before, an architecture for computing outsourcing to the Cloud is presented, that ensures a better utilization of the GPU resources. This has the potential of reducing costs, as well as increasing overall performance. This architecture is aimed at Cloud applications that do not fully utilize the GPU, but benefit from the use of this kind of hardware.

The architecture, shown in Fig. 2, consists of a group of docker containers that run GPU accelerated applications. These containers are grouped in a Cloud server instance that has a GPU assigned via the *passt-hrough* method. The host of the docker containers has a MPS server, and every container has a client to the MPS server. This connexion is possible by binding the files that represent the MPS UNIX socket (/tmp/nvidia-mps) from the host and the container. This is the main reason why the architecture uses containers instead of virtual machines, as UNIX sockets cannot be accessed natively by virtual machines.

With this architecture, different applications running inside the docker containers can efficiently share the GPU concurrently, increasing its usage.

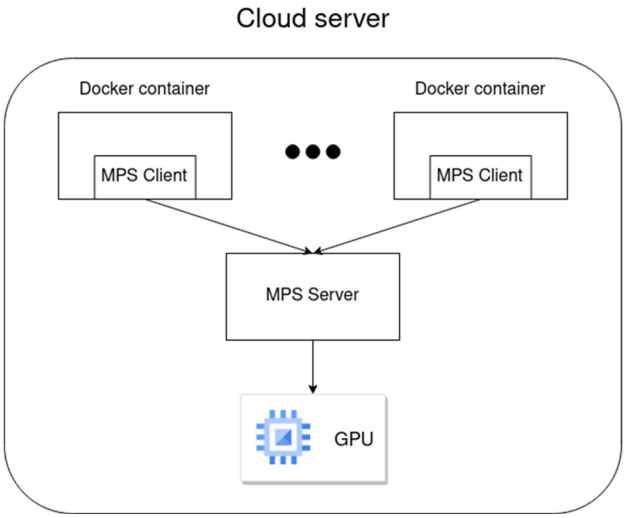

Fig. 2 Proposed Cloud architecture for computing outsourcing

In terms of security, it inherits the characteristics of the MPS technology. It is possible to establish Quality of Service restrictions, assigning specific amount of GPU computing resources and video memory to each container application. Modern GPUs (since Volta architecture) have fully isolated memory space, and provisioned GPU resources for the different processes when running a MPS server [48]. Data transfers between the GPU and the processes happen within the operating system environment, so they have the traditional process isolation. As every application is running inside a container, they have an isolated environment from the other applications. Apart from the specific characteristics described, this architecture has the same security and privacy challenges in terms of data generation, transfer, use, share, storage, archival and destruction as a traditional Cloud architecture [49, 50]. This means security issues associated with data transfers between the cloud servers and the clients, and data storage in the cloud have been extensively researched [51] and can be mitigated with techniques such as encryption [52] and data audit mechanisms such as verification signatures [53].

As memory is fully isolated between different MPS clients, data from a container application is secured from other processes, which brings some security assurance. However, as this technology has limited error containment, applications running in this architecture must be trusted. As any MPS process that provokes a fatal exception will propagate it to any other process running in the GPU at the same time, applications must take into account that exceptions caused by other external processes may be received. This is another reason why this architecture uses containers. The architecture is meant to be used by a single client with trusted

applications. Therefore, the increased security that virtual machines provide should not be needed, and containers provide better performance [51].

5 Experiments and results

In this section, the techniques to increase GPU usage explained before are going to be tested. For this experimentation, an effective parallelization framework is proposed by outsourcing the workload to GPU devices deployed on the Cloud servers, where in the first experiment, a classic externalization architecture will be used, and in the second the proposed architecture will be tested. In this environment, the client's request will be transferred to the server, which will execute the calculations and return the results to the client. As stated before, from the total time perceived by each client, only the results concerning the processing time ($T_{\text{Processing}}$) are relevant to this study.

The GPU deployed is the NVIDIA TITAN RTX. This device has been built according to the NVIDIA Turing architecture, and it has 24GB VRAM, 576 Tensor cores and 4608 CUDA cores.

For experimentation, the *gradient descent method* has been used over the *Rosenbrock function* [54]. This method calculates the local greatest or least value of a function when its variables are restricted to a given region. Usually, each variable represents a point in 2D or 3D. This method is commonly used in 3D calculations for optimizing CAD/CAM designs for industry and engineering [55, 56], and in Artificial Intelligence applications [57, 58]. The implementation computes the local minimum of every point of a vector (randomly generated) of a defined length. The kernel configuration consists on a thread per block, where there is a block for every point of the vector.

The metric used in the results shown is Floating Point Operations per Second (FLOPS). To get this metric, the amount of floating point operations that a kernel does has been calculated theoretically

$$FLOPS = \frac{27 * Iterations * Points}{time} \tag{2}$$

where the number of floating point operations is obtained by multiplying a constant number of operations by the number of iterations and the number of points in the vector. The total number of operations is then divided by the time of execution registered in the experiment.

In the first experiment, the objective is to test the capabilities of the Streams technique. The gradient descent method is run both in the normal way and using Streams. Several input data sizes have been used (50, 100, and 500) for computing increasing rounds of iterations to find the local minimum in the Rosenbrock function. The iterations per round can be consulted in Table 2. Ten independent kernels are launched by the main process. In the Streams version, every kernel is allocated in a different Stream. The results of this experiment can be seen in Fig. 3.

This experiment shows the benefits of the Streams technique. In this experiment, the concurrency limit is the 128 maximum parallel Streams that the NVIDIA TITAN RTX has. With this limit, the Stream version of the experiment has better performance than the default one. The GPU is being better utilized with Streams, which produces the increase in performance shown. An interesting aspect shown in the figure is that with default mode of execution, the performance increases until it reaches a stable point. On the other hand, with Streams, performance is stable from the beginning. This happens because with default execution, kernel scheduling and memory copies presents an overhead that has to be compensated with sufficient execution time (iterations). With Streams, these operations run concurrently, allowing for stable performance even with resource intensive kernels with low execution time.

Table 2 Number of iterations per round

Round	Iter.	Round	Iter.	Round	Iter.
1	500	21	10,500	41	20,500
2	1000	22	11,000	42	21,000
3	1500	23	11,500	43	21,500
4	2000	24	12,000	44	22,000
5	2500	25	12,500	45	22,500
6	3000	26	13,000	46	23,000
7	3500	27	13,500	47	23,500
8	4000	28	14,000	48	24,000
9	4500	29	14,500	49	24,500
10	5000	30	15,000	40	25,000
11	5500	31	15,500	51	25,500
12	6000	32	16,000	52	26,000
13	6500	33	16,500	53	26,500
14	7000	34	17,000	54	27,000
15	7500	35	17,500	55	27,500
16	8000	36	18,000	56	28,000
17	8500	37	18,500	57	28,500
18	9000	38	19,000	58	29,000
19	9500	39	19,500	59	29,500
20	10,000	40	20,000	60	30,000

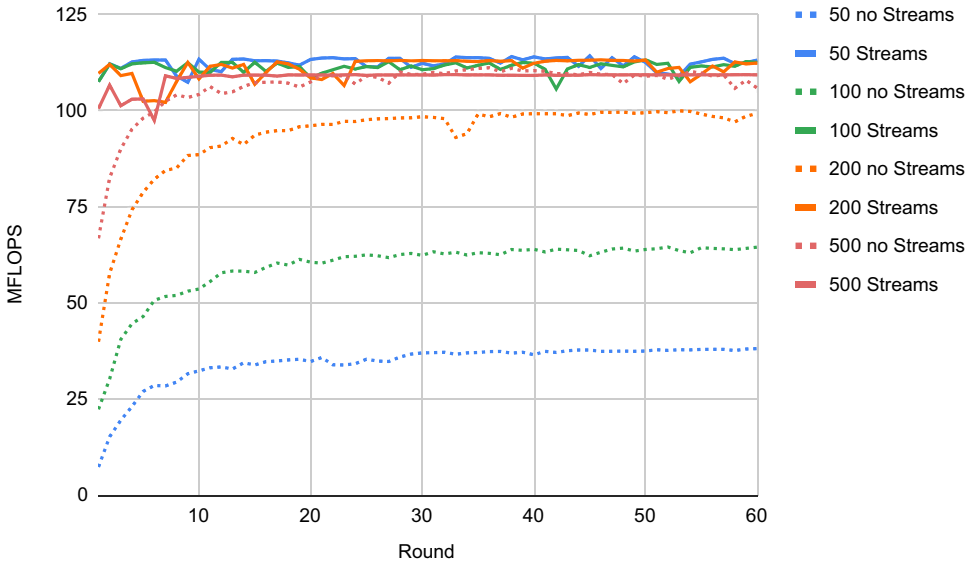

Fig. 3 Experiment 1: performance with different vector sizes with and without Streams

In the second experiment, the proposed Cloud architecture for computing outsourcing is evaluated. As in the previous experiment, the local minimum point is calculated on random vectors with different sizes for several rounds of iterations. This time, ten different applications launch a kernel call to compute the local minimum in the Rosenbrock function. With default mode, kernels are expected to run sequentially, as the applications will share the GPU usage by time multiplexing, but with MPS, the applications will connect to the MPS server via the MPS clients, so some level of concurrency is expected. The results of this experiment are shown in Fig. 4. The performance obtained is not directly comparable to the Streams experiment, and it is expected to be lower, as whole process times are being measured, and in the previous experiment, only kernel execution times have been measured.

The results in the second experiment show performance gains by the use of the proposed Cloud architecture. This difference increases with the number of iterations. This is expected, as measuring process time introduces the time of the process being created by the operating system. However, the experiment shows that some overhead is associated with setting up the MPS client, as in some cases, with low number of iterations, performance is lower with the MPS technique than with default CUDA operating mode.

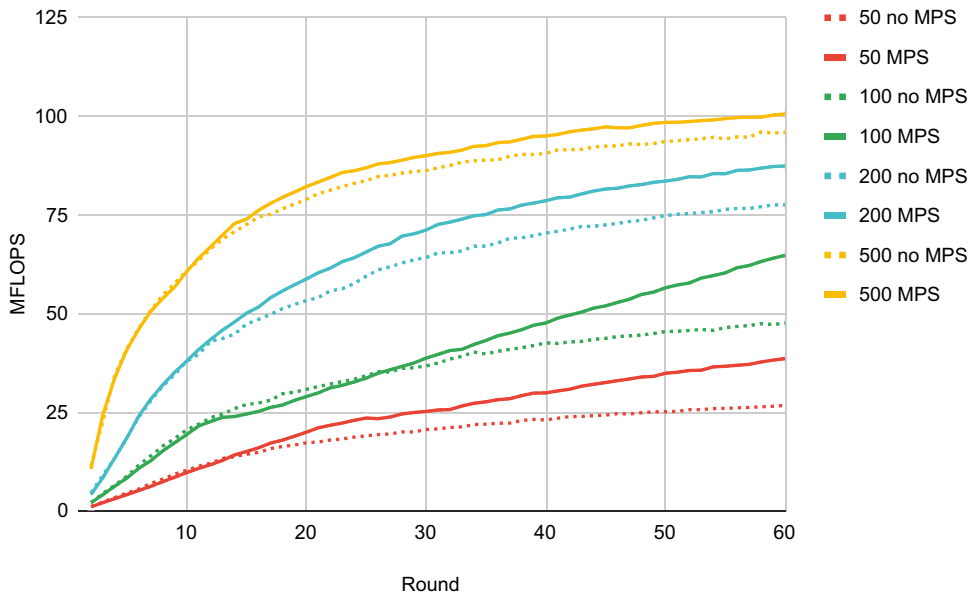

Fig. 4 Experiment 2: performance with different vector sizes using different processes with and without MPS

6 Conclusions

The new needs for the industry require the incorporation of technology that makes possible the transformation towards more agile production systems capable of giving a flexible response to market preferences, even more so in those traditional sectors subject to a changing consumer demands. In this line, computing outsourcing architectures of computing intensive computations to the Cloud, with architectures that include specialized devices such as GPUs, are a way to achieve greater performance for IoT, mobile devices and other Edge platforms.

This computing paradigm depends on finding efficient methods to virtualise GPU infrastructure and maximize the hardware utilization of these devices. However, at the moment, Cloud providers do not provide a way of provisioning a personalized amount of GPU resources adapted to specific applications, as they currently do not widely support GPU virtualization. To address these problems, GPU programming techniques have been researched, and a Cloud architecture for computing outsourcing has been proposed for applications that not fully utilize all the resources of the GPU.

The experiments have put to the test the proposed techniques and architecture. The results show that performance can be increased by incrementing overall GPU usage. In the case of Streams, it is a great technology to increase kernel parallelism from within an application level. Programmers may use this technique for an increase in performance whenever possible. The proposed Cloud architecture has also shown promising results. It allows for multiple GPU accelerated applications to share the use of the GPU concurrently, presenting a great opportunity to reduce costs and increase performance, as it makes a more efficient use of Cloud resources.

However, this architecture cannot be used in a general way, with totally independent applications. As the architecture does not present complete GPU virtualization, due to the limitations of the MPS technology, specially with the error containment system, this architecture is reserved to trusted and closely related applications. In the future, this research should be extended to cover remaining challenges around this paradigm and to propose an extended model with effectively virtualized GPUs in the Cloud. In this regard, NVIDIA MIG technology is a great candidate for providing complete and efficient GPU virtualization for Cloud systems.

Author Contributions All authors have contributed equally in the development of this work.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature. This work was supported by the Spanish Research Agency (AEI) under project HPC4Industry PID2020-120213RB-I00.

Declarations

Ethical approval Not applicable.

Consent to participate All authors have read and approved the final manuscript.

Consent for publication All authors agree to publish in The International Journal of Advanced Manufacturing Technology.

Conflict of interest The authors of this paper have no conflict of interest with regard to this publication.

References

1. Gartner (2020) Gartner Top Strategic Technology Trends for 2021 — gartner.com. https://www.gartner.com/smarterwithgartner/gartner-top-strategic-technology-trends-for-2021. Accessed 15 Sep 2022
2. Efstathiou C, Tapoglou N (2022) Simulation of spiral bevel gear manufacturing by face hobbing and prediction of the cutting forces using a novel cad-based model. Int J Adv Manuf Technol 1–25
3. Stavropoulos P, Tzimanis K, Souflas T, Bikas H (2022) Knowledge-based manufacturability assessment for optimization of additive manufacturing processes based on automated feature recognition from cad models. Int J Adv Manuf Technol 122(2):993–1007
4. Kounta CAKA, Arnaud L, Kamsu-Foguem B, Tangara F (2022) Review of ai-based methods for chatter detection in machining based on bibliometric analysis. Int J Adv Manuf Technol 1–26
5. Rico-Garcia H, Sanchez-Romero JL, Jimeno-Morenilla A, Migallon-Gomis H, Mora-Mora H, Rao RV (2019) Comparison of high performance parallel implementations of tlbo and jaya optimization methods on manycore gpu. IEEE Access 7:133822–133831
6. Khouzami N, Michel F, Incardona P, Castrillon J, Sbalzarini IF (2022) Model-based autotuning of discretization methods in numerical simulations of partial differential equations. J Comput Sci 57:101489. https://doi.org/10.1016/j.jocs.2021.101489
7. Sfiligoi I, Schultz D, Riedel B, Wuerthwein F, Barnet S, Brik V (2020) Demonstrating a pre-exascale, cost-effective multi-cloud environment for scientific computing: producing a fp32 exaflop hour worth of icecube simulation data in a single workday. Practice and Experience in Advanced Research Computing (PEARC '20). Association for Computing Machinery, New York, pp 85–90. https://doi.org/10.1145/3311790.3396625
8. Lloret-Climent M, Nescolarde-Selva JA, Mora-Mora H, Jimeno-Morenilla A, Alonso-Stenberg K (2019) Design of products

through the search for the attractor. IEEE Access 7:60221–60227. https://doi.org/10.1109/ACCESS.2019.2915678
9. Chen H, Lu M, Ma Z, Zhang X, Xu Y, Shen Q, Zhang W (2021) Learned resolution scaling powered gaming-as-a-service at scale. IEEE Trans Multimedia 23:584–596. https://doi.org/10.1109/TMM.2020.2985538
10. Han Y, Guo D, Cai W, Wang X, Leung VCM (2022) Virtual machine placement optimization in mobile cloud gaming through qoe-oriented resource competition. IEEE Trans Cloud Comput 10(3):2204–2218. https://doi.org/10.1109/TCC.2020.3002023
11. Peña AJ, Reaño C, Silla F, Mayo R, Quintana-Ortí ES, Duato J (2014) A complete and efficient cuda-sharing solution for hpc clusters. Parallel Comput 40(10):574–588. https://doi.org/10.1016/j.parco.2014.09.011
12. Giunta G, Montella R, Agrillo G, Coviello G (2010) A gpgpu transparent virtualization component for high performance computing clouds. In: D'Ambra P, Guarracino M, Talia D (eds) Euro-Par 2010 - Parallel Processing. Springer Berlin Heidelberg, Berlin, pp 379–391
13. NVIDIA (2022a) Virtual GPU Software User Guide :: NVIDIA Virtual GPU Software Documentation — docs.nvidia.com. https://docs.nvidia.com/grid/13.0/grid-vgpu-user-guide/index.html. Accessed 16 Sep 2022
14. NVIDIA (2022b) NVIDIA Multi-Instance GPU (MIG) — nvidia.com. https://www.nvidia.com/es-es/technologies/multi-instance-gpu/. Accessed 16 Sep 2022
15. Waheed A, Shah MA, Mohsin SM, Khan A, Maple C, Aslam S, Shamshirband S (2022) A comprehensive review of computing paradigms, enabling computation offloading and task execution in vehicular networks. IEEE Access
16. Mora H, Mora Gimeno FJ, Signes-Pont MT, Volckaert B (2019) Multilayer architecture model for mobile cloud computing paradigm. Complexity 2019
17. Dash S, Ahmad M, Iqbal T et al (2021) Mobile cloud computing: a green perspective. In: Intelligent Systems. Springer, pp 523–533
18. Mora Mora H, Gil D, Colom Lopez JF, Signes Pont MT (2015) Flexible framework for real-time embedded systems based on mobile cloud computing paradigm. Mobile Inf Syst 2015
19. Qiu T, Chi J, Zhou X, Ning Z, Atiquzzaman M, Wu DO (2020) Edge computing in industrial internet of things: architecture, advances and challenges. IEEE Commun Surv Tutorials 22(4):2462–2488. https://doi.org/10.1109/COMST.2020.3009103
20. Yuan H, Zhou M (2021) Profit-maximized collaborative computation offloading and resource allocation in distributed cloud and edge computing systems. IEEE Trans Autom Sci Eng 18(3):1277–1287. https://doi.org/10.1109/TASE.2020.3000946
21. Zheng G, Zhang H, Li Y, Xi L (2020) 5g network-oriented hierarchical distributed cloud computing system resource optimization scheduling and allocation. Comput Commun 164:88–99. https://doi.org/10.1016/j.comcom.2020.10.005
22. Etemadi M, Ghobaei-Arani M, Shahidinejad A (2020) Resource provisioning for iot services in the fog computing environment: An autonomic approach. Comput Commun 161:109–131. https://doi.org/10.1016/j.comcom.2020.07.028
23. Mora H, Peral J, Ferrandez A, Gil D, Szymanski J (2019) Distributed architectures for intensive urban computing: a case study on smart lighting for sustainable cities. IEEE Access 7:58449–58465
24. Fang Z, Xu X, Dai F, Qi L, Zhang X, Dou W (2020) Computation offloading and content caching with traffic flow prediction for internet of vehicles in edge computing. In: 2020 IEEE International Conference on Web Services (ICWS). IEEE, pp 380–388
25. Li M (2021) Computation offloading and task scheduling on network edge. PhD thesis, University of Waterloo. http://hdl.handle.net/10012/17188
26. Ribes VS, Mora H, Sobecki A, Gimeno FJM (2020) Mobile cloud computing architecture for massively parallelizable geometric computation. Comput Ind 123:103336
27. Martinez-Noriega EJ, Yazaki S, Narumi T (2021) Cuda offloading for energy-efficient and high-frame-rate simulations using tablets. Concurr Comput Pract Experience 33(2):e5488
28. Tsog N, Mubeen S, Bruhn F, Behnam M, Sjödin M (2021) Offloading accelerator-intensive workloads in cpu-gpu heterogeneous processors. In: 2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA). IEEE, pp 1–8
29. Jimeno-Morenilla A, Azariadis P, Molina-Carmona R, Kyratzi S, Moulianitis V (2021) Technology enablers for the implementation of industry 4.0 to traditional manufacturing sectors: a review. Comput Ind 125:103390. https://doi.org/10.1016/j.compind.2020.103390
30. Morell-Giménez V, Jimeno-Morenilla A, García-Rodríguez J (2013) Efficient tool path computation using multi-core gpus. Comput Ind 64(1):50–56. https://doi.org/10.1016/j.compind.2012.09.009
31. Tang M, Tong R, Narain R, Meng C, Manocha D (2013) A gpu-based streaming algorithm for high-resolution cloth simulation. In: Computer Graphics Forum, vol 32. Wiley Online Library, pp 21–30
32. Vassilev TI (2016) Garment simulation and collision detection on a mobile device. Int J Mob Comput Multimedia Commun (IJMCMC) 7(3):1–15
33. XueChuanyu DH, Mingmin Z, Zhigeng P (2017) Real-time simulation on virtual dressing based on virtual human body model. J Syst Simul 29(11):2847
34. Hui Z, Zhen L, Yanjie C et al (2018) Real-time collision detection method for fluid and cloth. J Comput Aided Des Graph 30(4):602–610
35. Leaf J, Wu R, Schweickart E, James DL, Marschner S (2018) Interactive design of periodic yarn-level cloth patterns. ACM Trans Graph 37(6). https://doi.org/10.1145/3272127.3275105
36. Li Z, Xiong G, Zhang X, Shen Z, Luo C, Shang X, Dong X, Bian GB, Wang X, Wang FY (2019) A gpu based parallel genetic algorithm for the orientation optimization problem in 3d printing. In: 2019 International Conference on Robotics and Automation (ICRA). pp 2786–2792. https://doi.org/10.1109/ICRA.2019.8793989
37. Huang R, Dai N, Li D, Cheng X, Liu H, Sun D (2018) Parallel non-dominated sorting genetic algorithm-ii for optimal part deposition orientation in additive manufacturing based on functional features. Proc IME C J Mech Eng Sci 232(19):3384–3395. https://doi.org/10.1177/0954406217737105
38. Talib MA, Majzoub S, Nasir Q, Jamal D (2021) A systematic literature review on hardware implementation of artificial intelligence algorithms. J Supercomput 77(2):1897–1938
39. Jimeno-Morenilla A, Sanchez-Romero JL, Migallon H, Mora-Mora H (2019) Jaya optimization algorithm with gpu acceleration. J Supercomput 75(3):1094–1106
40. Chen Z, Wang J, He H, Huang X (2014) A fast deep learning system using gpu. In: 2014 IEEE International Symposium on Circuits and Systems (ISCAS). pp 1552–1555. https://doi.org/10.1109/ISCAS.2014.6865444
41. Li B, Arora R, Samsi S, Patel T, Arcand W, Bestor D, Byun C, Roy RB, Bergeron B, Holodnak J, Houle M, Hubbell M, Jones M, Kepner J, Klein A, Michaleas P, McDonald J, Milechin L, Mullen J, Prout A, Price B, Reuther A, Rosa A, Weiss M, Yee C, Edelman D, Vanterpool A, Cheng A, Gadepally V, Tiwari D (2022) Ai-enabling workloads on large-scale gpu-accelerated system: characterization, opportunities, and implications. In: 2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA). pp 1224–1237. https://doi.org/10.1109/HPCA53966.2022.00093

42. Herrera A (2015) Nvidia grid vgpu: delivering scalable graphics-rich virtual desktops. Retrieved Aug 10:2015
43. vikancha MSFT (2022) Serie NVv4 - Azure Virtual Machines — learn.microsoft.com. https://learn.microsoft.com/es-es/azure/virtual-machines/nvv4-series. Accessed 26 Sep 2022
44. Cloud T (2022) GPU Cloud Computing Instance Types. https://main.qcloudimg.com/raw/document/intl/product/pdf/tencent-cloud_560_11625_en.pdf. Accessed 26 Sep 2022
45. Li H, Yu D, Kumar A, Tu YC (2014) Performance modeling in cuda streams - a means for high-throughput data processing. In: 2014 IEEE International Conference on Big Data (Big Data). pp 301–310. https://doi.org/10.1109/BigData.2014.7004245
46. NVIDIA (2015) GPU Pro Tip: CUDA 7 Streams Simplify Concurrency — developer.nvidia.com. https://developer.nvidia.com/blog/gpu-pro-tip-cuda-7-streams-simplify-concurrency/. Accessed 27 Sep 2022
47. Olmedo IS, Capodieci N, Martinez JL, Marongiu A, Bertogna M (2020) Dissecting the cuda scheduling hierarchy: a performance and predictability perspective. In: 2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS). pp 213–225. https://doi.org/10.1109/RTAS48715.2020.000-5
48. Corporation N (2021) Multi-Process Service :: GPU Deployment and Management Documentation — docs.nvidia.com. https://docs.nvidia.com/deploy/mps/index.html. Accessed 15-Sep-2022
49. Chen D, Zhao H (2012) Data security and privacy protection issues in cloud computing. In: 2012 International Conference on Computer Science and Electronics Engineering, vol 1. pp 647–651. https://doi.org/10.1109/ICCSEE.2012.193
50. Mushtaq MF, Akram U, Khan I, Khan SN, Shahzad A, Ullah A (2017) Cloud computing environment and security challenges: a review. Int J Adv Comput Sci Appl 8(10)
51. Singh A, Chatterjee K (2017) Cloud security issues and challenges: a survey. J Netw Comput Appl 79:88–115. https://doi.org/10.1016/j.jnca.2016.11.027
52. Ate SZA (2020) Improved cloud data transfer security using hybrid encryption algorithm. Institute of Advanced Engineering and Science Vol 20, No 1: October 2020. https://ijeecs.iaescore.com/index.php/IJEECS/article/view/21787/14240
53. Selvamani K, Jayanthi S (2015) A review on cloud data security and its mitigation techniques. Proc Comput Sci 48:347–352. https://doi.org/10.1016/j.procs.2015.04.192. International Conference on Computer, Communication and Convergence (ICCC 2015)
54. Rosenbrock H (1960) An automatic method for finding the greatest or least value of a function. Comput J 3(3):175–184
55. Barral D, Perrin JP, Dombre E, Liegeois A (2001) Simulated annealing combined with a constructive algorithm for optimising assembly workcell layout. Int J Adv Manuf Technol 17(8):593–602
56. Ns WEE (1994) Bezier curve approximation in cad/cam system. Commun Korean Math Soc 9(1):253–259
57. Deepa N, Prabadevi B, Maddikunta PK, Gadekallu TR, Baker T, Khan MA, Tariq U (2021) An ai-based intelligent system for healthcare analysis using ridge-adaline stochastic gradient descent classifier. J Supercomput 77(2):1998–2017
58. Khasanov D, Tojiyev M, Primqulov O (2021) Gradient descent in machine learning. In: 2021 International Conference on Information Science and Communications Technologies (ICISCT). pp 1–3. https://doi.org/10.1109/ICISCT52966.2021.9670169