Hardware-accelerated High-resolution Video Coding in Virtual Network Functions

P. Comi and P. Secondo Crosta
Innovation & Research, Italtel S.p.A
Castelletto di Settimo Milanese (MI)
Email: paolomaria.comi@italtel.com
paolosecondo.crosta@italtel.com

M. Beccari and P. Paglierani
Research & Development, Italtel S.p.A
Castelletto di Settimo Milanese (MI)
Email: marco.beccari@italtel.com
pietro.paglierani@italtel.com

G. Grossi, F. Pedersini and A. Petrini
Dipartimento di Informatica
Università degli Studi di Milano
Via Comelico 39, I-20135 Milano, Italy
Email: {grossi,pedersini,petrini}@di.unimi.it

***Abstract*—Network Function Virtualization (NFV) has become a widely acclaimed approach to facilitate the management and orchestration of network services. However, after rapidly achieving a widespread success, NFV is now challenged by the overwhelming demand of computing power originated by the never-ending growth of innovative applications coming from the Internet world. To overcome this problem, the use of h/w acceleration combined with NFV has been proposed. This way, the computing performance of commodity servers can be greatly enhanced, without losing the advantages offered by NFV in service management. In this paper, to demonstrate the potentialities of NFV and h/w acceleration, a Virtual Network Function for video coding (video Transcoding Unit – vTU) is presented. The vTU is accelerated by a General Purpose GPU, and is based on Open Source software packages for media processing. The vTU architecture is firstly described in details. A thorough characterization of its computing performance is then reported, and the obtained results are compared to those achieved with non-accelerated and/or non-virtualized versions of the vTU itself. Also, the performance provided by an original, GPU accelerated version of the VP8 encoder is presented. The activities described in this paper have been carried out within the EU FP7 T-NOVA project.**

I. Introduction

In the telecommunication sector, Network Function Virtualization (NFV) has rapidly become a widely acclaimed approach to facilitate the management and orchestration of network applications [1]. NFV allows the migration of network functionalities from bespoke hardware to virtualized IT infrastructures based on commodity servers, where they run as software components. Adopting NFV, Telco operators expect to achieve significant cost reductions, gaining at the same increased flexibility, accelerated service deployment and facilitated infrastructure management [2]. However, after rapidly achieving a wide success, the NFV approach is now challenged by the overwhelming demand of computing power, originated by the huge growth of innovative applications and services coming from the Internet world. To overcome this problem, the use of ad hoc h/w accelerators within the NFV framework has been proposed. This way, the computational performance of commodity servers can be greatly enhanced, without losing the advantages brought about by NFV in service management [3]. In this paper, to demonstrate the potentialities of NFV combined with h/w acceleration, a Virtual Network Functions (VNF) for video coding (video Transcoding Unit – vTU) is presented. The vTU is accelerated by a General Purpose GPU (GP-GPU), and based on the most popular Open Source software packages for media processing, such as AVConv or FFMPEG. First, the architecture of the vTU is presented in details. The experimental characterization of its video coding performance is then reported, and compared to the performance obtained with non-virtualized and/or non-accelerated versions of the vTU itself. The use of commercial GPU based H.264 and H.265 encoders has been evaluated, and an original GPU-accelerated version of the VP8 encoder is presented and discussed. The results presented in this paper have been obtained within the activities of the EU FP7 T-NOVA project. The T-NOVA framework allows and facilitates the use of VNFs which rely on h/w accelerators [4]. Thus, management of h/w accelerators within the T-NOVA framework is also presented and briefly discussed. The structure of the paper is as follows. In Section II, the overall T-NOVA framework is briefly introduced, and its approach to use h/w acceleration within NFV is summarized. In Section III the video coding problem is addressed, while in Section IV the vTU architecture is described. Finally, in Section V, the obtained experimental results are presented and discussed.

II. H/W acceleration management in T-NOVA

The T-NOVA project [5] aims to design and implement an integrated management architecture, including an Orchestrator platform, for the automated provision, management, monitoring and optimization of Virtualised Network Functions over Network/IT infrastructures. Also, T-NOVA aims to introduce and promote a novel Marketplace for NFV, introducing new business cases and considerably expanding market opportunities by attracting new entrants to the networking market. T-NOVA exploits and extends Software Defined Networking (SDN) aspects, focusing on the OpenFlow [6] technology, for efficient management of network resources, including network slicing, traffic redirection and QoS provision. A notable aspect of T-NOVA framework is the ability to extend the classical cloud resources such as computation, network, and storage, with software and hardware accelerators. Within T-NOVA a VNF can operate using both the basic cloud resources and also the accelerated ones whenever they are available and it is requested to guarantee certain service level agreement (SLA).

The information for achieving this goal is written the VNF descriptor (VNFD) that is the document that accompaines all the VNFs. The VNFD provides the technical description of the VNF behaviour, the deployment requirements and the VNF's ability to exploit hardware acceleration. In T-NOVA the VNFD accommodates also the business description of the VNF such as the definition of SLA together with the billing schema associated to each level of service. The VNFD therefore provides key information to the mapping function that determines the best place where to instantiate a VNF and its internal components according to the requested SLA. The management of h/w accelerated VNF in T-NOVA is based on a three step mechanism. As explained above, the VNF must first declare in the VNFD its capabilities to exploit h/w acceleration, and which type of acceleration is required. This information is inserted into T-NOVA Infrastructure Repository, that records also the location of the various types of h/w accelerators. A fundamental block of the T-NOVA Orchestrator, the Service Mapping block, accommodates the VNF in the infrastructure location where the required acceleration devices are available. This implies the Service Mapping block to execute a mapping function that combines the information in the Infrastructure Repository with the specific requests in the VNFD for the given VNF, yielding to the optimal assignment of the infrastructure resources. In the case of the vTU, the h/w accelerators required are Nvidia GPGPU. However, details on this mechanism are out of the scope of this paper. Conversely, in the following sections, we present how the vTU exploits the available GPGPU resources to dramatically increase its performances in video transcoding.

III. Video encoding

In the NFV panorama, the Network Functions and Network Services dealing with video data, such as video processing or transcoding functions, cover a role of primary importance, considering that nowadays video contents take approximately 80% of the global network traffic [7], with increasing trend. Video-related VNF's, in particular those related to video coding (encoding/decoding), are typically computationally-intensive, and, in addition, they often require the fulfillment of rather strict real-time constraints (for instance, a maximum allowed latency of one frame). For this reason, video coding functions would greatly benefit from the availability of High-Performance Computing (HPC) resources. Under the several kinds of HPC resources available for standard server architectures, many-core GP-GPU's represent the mostly cost-effective (in terms of MFlops/price ratio), as long as the inherently parallel GPU architecture can be effectively exploited. Moreover, GPU programming has become a simpler task, with respect to other h/w acceleration resources (e.g. FPGA's), thanks to parallel high-level languages and tools, like OpenCL and NVIDIA's CUDA [8], made available by the GPU manufacturers. For these reasons, GPU-based acceleration has been chosen for the development of video encoding and decoding VNF's in the T-NOVA framework.

However, the achievement of significant performance improvements in video encoding/decoding by means of GPU's is not so straightforward. Such tasks are difficult to parallelize because most video coding schemes (and, among them, H264 [9] and Google's VP8 [10], considered in the Project) are inherently sequential. In fact, coding processes try to maximally exploit the spatial and temporal correlation present in the data. For this reason, both encoding and decoding processes rely on the assumption that, while processing a macroblock, all previous macroblocks (i.e. those above and left of the considered one in the current frame, and some entire previous frames) have been already processed. This eliminates any chance of massive data parallelism; for instance, simply processing many macroblocks in parallel cannot be done. In addition, all standard coding schemes have been conceived for optimal CPU processing (e.g. macroblocks are small enough to be processed by keeping all data within the innermost levels of CPU cache), therefore they are "hard to beat" by different computing architectures.

In order to design a scheme for GPU-accelerated VP8 encoding, we analyzed the main limits and obstacles to the achievement of significant acceleration by adopting the traditional GPU-programming approaches. We considered two classical approaches for porting algorithms onto GPU's: one typical approach is to port the lowest-level, CPU-intensive computation onto GPU. This approach, often quite successful, is very inefficient in this case, because all low-level computing functions, besides the fact that they cannot be run in parallel, accomplish very tiny tasks, and are consequently called a huge amount of times. This makes the use of GPU totally inefficient, as the processing time is dominated by the GPU-CPU memory transfers for each call. Also the other, somehow complementary approach, that is, to port the whole CPU computation onto a GPU kernel, would solve for the GPU-CPU transfer issue, but encounters the problem that a significant part of the encoding algorithms is inherently sequential, therefore it must be processed by a single GPU core, which is, of course, much slower that a CPU core.

For this reason we propose an alternative, cooperative CPU-GPU approach, in which the main framework of the algorithms is kept running on the standard CPU, while the most demanding task in terms of computing effort, like the Motion Estimation (ME) procedure, is delegated to the GPU and, therefore, computed independently (and in parallel with the CPU). As Motion Estimation is completed, the main algorithm, running on the CPU, can just "pick" the computed Motion Vectors (MV) without having to run the computationally-expensive motion search. Moreover, in order to further maximize the processing speed, the whole ME process on the GPU have been partitioned into many streams, each working on a different set of macroblocks. This configuration gives the advantage of a minimized latency by the CPU's main thread, which could have to wait for the Motion Vectors of the initial macroblocks.

As starting point for this development we took Google's open-source WebM Project [11], an open project for VP8/VP9 video coding, and added to this framework the functions for

Motion Estimation on GPU. The aim is to develop a GPU-accelerated version of the VP8 encoder, that could be then officially contributed to the WebM Project.

As shown in Section V, this approach has reached a maximum speed-up factor of almost two, which is an encouraging result, compared to what obtained in literature, for instance on GPU-accelerated encoders for H264 [12], [13].

IV. The Video Transcoding Unit architecture

The vTU is a VNF, i.e. it runs as a virtual machine instantiated in a Virtualized Infrastructure, according to the definition given in [14]. Its development has followed the guidelines provided to Function Developers by the T-NOVA framework. For the sake of brevity, in the following we will not give further details on these aspects, which are fully described with abundance of examples in [4]. vTU's VNF is available as a preconfigured virtual image which can be customized through a graphical interface (Marketplace) and incorporated into a Network Service; then, instantiation is automatically performed by T-NOVA orchestrator (TeNOR). One of the benefit of customization, is that different configurations can be tailored to the customer requirements in order to meet performance levels or time constraints, i.e. by increasing the number of virtual cores in case of multiple concurrent transcoding operations.

The non-accelerated vTU version can run on any Virtualized Infrastructure. Conversely, the h/w accelerated virtualized vTU can run on any commodity server that hosts in one of its PCIe slots (at least) one Nvidia GPU. In particular, in the experiments described in this paper the Nvidia Quadro M4000 has been used [15]. Such device, as shown in Section V, can provide a great increase in coding performance, at a low cost and low power consumption. In order to be used by the virtualized vTU, the GPU must be virtualized. The problem of GPU virtualization can be stated as follows: a guest Virtual Machine (VM), running on a hardware platform provided with GPU-based accelerators, must be able to concurrently and independently access the GPUs, without incurring in security issues. The technique adopted to achieve GPU virtualization is based on PCI-Passthrough, sometimes referred to also as Direct Device Assignment. GPU virtualization achieved through PCI-Passthrough techniques is thoroughly described in [16].

In the vTU, the audio and video transcoding capabilities are provided by the Libav library [17], a very popular open source library, which can perform encoding and decoding according to a wide set of coding standards. AVConv tool from Libav is used for performing the conversion between audio and video formats and containers; while it already supports a wide variety of hardware accelerations, native GPU support in encoding tasks is quite limited, experimental and restricted only to H.264 and H.265 standards, exploiting the NVidia NVENC hardware encoder of medium and high level NVidia GPUs [18]. The AVConv tool running in the vTU has been modified so that VP8 encoding tasks can also greatly benefit from the virtualization of GPUs.

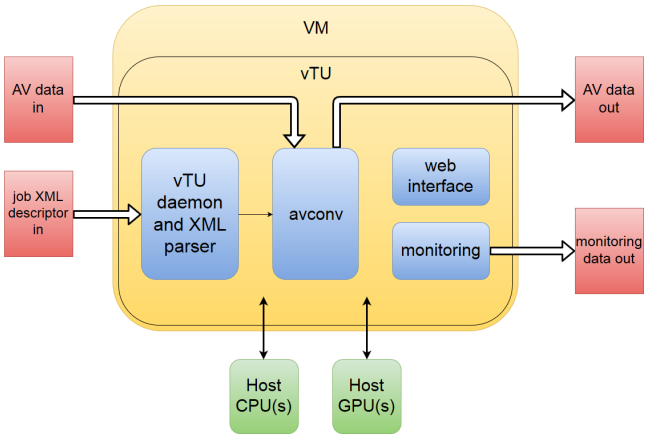

Fig. 1. vTU internal structure

A short description of vTU internal structure and main operations follows.

The vTU can operate on static multimedia files (local or remote) and audio and/or video streams; multiple jobs can run concurrently.

Fig. 2. vTU web interface

The vTU is provided with a web interface (fig. 2) which exposes the user accessible folders containing multimedia data ("input" and "output" folders), jobs descriptors ("spool", "run", "done") and logs ("log", "undone" and "error"). At any time, the user can access the web interface to peek the current status of the vTU.

Jobs inside the vTU are fully automatized: transcoding tasks are started by providing the vTU a XML descriptor of the job; this simple descriptor provides the vTU all the information needed by avconv for performing the transcoding task: such information are represented by, but not limited to, source and output data in form of links to static files (local or remote) or rtsp multimedia streams, as well as input and output format, encapsulation, frame size, frame rate, output bitrate and so

on.
The descriptor is processed by the vTUdaemon as soon as it is copied into the ”spool” folder; then, after validating the XML descriptor, the vTUdaemon invokes one or several instances of avconv for fulfilling job requests and its descriptor is moved to the ”running” folder. At the end of the task, the descriptor is moved to one of the ”done”, ”undone” or ”error” folder (in case of successful, cancelled or unsuccessful outcome). Also, if the vTU is requested to operate on static files, the user is required to upload the input multimedia file in the ”input” folder, while the transcoded output will be available on the ”output” folder at the completion of the job.

Fig. 3. vTU Grafana dashboard

In order to monitor the vTU status and performance, several metrics are generated and constantly written at regular and modifiable basis to a user defined ip address and port. For demoing purpose, currently, monitoring data generated by the vTU is redirected to a local instance of InfluxDB database, a popular open source database solution, oriented to collecting and managing time-series data [19]; then, such records are graphically arranged and visualized into a browser window by Grafana (Fig. 3), a popular dashboard for displaying time series metrics [20].
Currently, metrics collected are related, but not limited, to CPU, GPU (when available), memory, and disk usage; also encoding performances are captured and visualized, such as the number of transcoded frames per second.

V. Experimental results

The main aim of the experimental tests carried out on the vTU is to quantitatively evaluate the performance of the GPU-accelerated version of this VNF, with respect to the “standard” version, just running on one or more CPU cores. Although performance comparisons among coding schemes are often evaluated in terms of obtained image quality vs. coded bit-rate, in this application it is more meaningful to compare the coding speed (keeping the same target image quality), because timing constraints are typically the most important aspect in applications considering network-distributed video contents.

Two kinds of experiments are presented in the following: in the first, the entity of the speed-up provided by the GPU-accelerated encoder is evaluated, with respect to that running on CPU; the second is aimed to assess the benefit of GPU acceleration in case of high loads of video encoding tasks, that is, when multiple encoding sessions need to be processed simultaneously.

TABLE I
Speed-up provided by GPU-acceleration – VP8 encoder

VP8 encoding - Processing time			
Sequence	CPU (`libvpx`)	CPU+GPU	Speed-up
Battledrift (720p)	16.82 s	13.03 s	1.29x
Right (1080p)	38.33 s	29.55 s	1.30x
Jockey (4k)	106.73 s	57.27 s	1.86x

A. Computational speed-up

Table I summarizes some of the speed tests, carried out for different image resolutions, in which we compare the processing time of the proposed cooperative GPU-accelerated VP8 encoder to the standard CPU-based encoder available in the WebM `libvpx` library [11]. Since the processing time of a video encoder strongly depends on many coding parameters (like coding *speed factors*, target bit-rate, image quality, and computing time itself), in order to obtain a meaningful comparison, the presented data have been obtained by adopting the parameters that provide the same image quality (i.e. mean PSNR) and the same compression ratio (same size of the encoded stream) in both cases.

As Table I shows, the proposed algorithm reaches a speed-up of approx. +30% for HD and Full-HD resolutions, and increases to +86% for 4k resolution. This is a quite encouraging result; we considered for comparison the H264 encoder[1] provided by X264, the publicly available video coding library by VideoLAN [13], because it is also available both in standard and GPU-accelerated version. By running both versions of this encoder on the same sequences, we experienced a speed-up by the GPU-accelerated encoder, with respect to its standard version, of +11%, +15% and +24% for HD, Full-HD and 4k resolutions, respectively.

B. Performance running multiple sessions

A significant further benefit of exploiting GPU resources comes from the consequent off-loading of the CPU cores, as the most significant part of the computation effort is delegated to the GPU. In such applications, it often happens that the GPU computing resources are inherently “underemployed” by a single process. In our case, for instance, a single encoding process takes approximately 10% of the resources made available by a 2048-cores GPU device.

[1] A comparison with a VP8 encoder would have been more meaningful, but, to our knowledge, there is no open-source GPU-accelerated VP8 encoder.

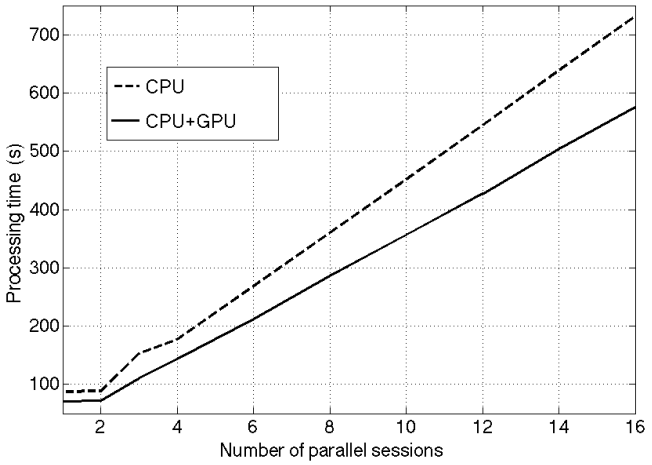

Fig. 4. Processing time with multiple encoding sessions in parallel, with and without GPU support.

Since the computing load at the CPU side has been reduced by delegating the GPU, this has the consequence that, in case of multiple sessions running in parallel, the processing time grows more slowly than in the case of entire computing load assigned to CPU's only, because of the lower CPU load for each new session in the GPU-accelerated case. For this reason, by running multiple coding sessions in parallel with GPU support, the advantage in performance of GPU-accelerated operation over GPU-less should grow further, as long as the GPU is able to "scale well", that is, to accommodate increasing loads of parallel computation without significant switching overload.

In order to quantitatively evaluate this concept, we carried out this comparison by running multiple sessions in parallel, both with and without GPU support. We ran the comparison on a VM equipped with 2 Xeon CPU cores and a GPU with 2048 cores. Considering that the measured GPU load for a single session is about 10%, we considered a number of parallel sessions from 1 to 16, in order to be sure to come to GPU overloading, making therefore any possible scaling problem visible. Figure 4 shows the obtained processing times.

The graphs show that, indeed, the processing time with GPU support is always lower, and grows with a lower steepness, than without GPU. This proves that the proposed cooperative CPU-GPU approach scales well, at least up to 16 parallel sessions. The speed-up (i.e. the ratio between the two processing times) holds rather constant, around 25–30%, actually with a light increasing trend, thus suggesting a tendency of the system to even improve its efficiency with increasing processing loads.

VI. Conclusion

In this work, the combination of NFV combined with h/w acceleration has been addressed, within the T-NOVA Project. In particular, the use of a General Purpose Graphical Processing Unit to accelerate a VNF for video transcoding in the T-NOVA framework has been analyzed and discussed. The architecture of such a specific VNF - the video Transcoding Unit - has been introduced, and the results obtained in a number of experiments to characterize its performance have been presented and discussed. The obtained results clearly show that the use of h/w acceleration can greatly increase the VNF video transcoding performance, without affecting in any way the advantages in service management and application development brought about by virtualization.

Acknowledgments

This work was undertaken under T-NOVA Integrated Project, co-funded by the European Commission - 7 Framework Programme - Grant Agreement No. 619520. Duration: 36 months (Jan 2014 – Dec 2016)

References

[1] "Network function virtualization (vnf); management and orchestration," eTSI GS NFV-MAN 001 V1.1.1 (2014-12). [Online]. Available: https://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_NFV-MAN001v010101p.pdf

[2] A. Manzalini and R. Saracco, "Software networks at the edge: A shift of paradigm," in *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*, 2013, pp. 1–6.

[3] P. Paglierani, "High performance computing and network function virtualization: A major challenge towards network programmability," in *Communications and Networking (BlackSeaCom), 2015 IEEE International Black Sea Conference on*, May 2015, pp. 137–141.

[4] G. Xilouris, E. Trouva, F. Lobillo, J. Soares, J. Carapinha, M. McGrath, G. Gardikis, P. Paglierani, E. Pallis, L. Zuccaro, Y. Rebahi, and A. Kourtis, "T-nova: A marketplace for virtualized network functions," in *Networks and Communications (EuCNC), 2014 European Conference on*, 2014, pp. 1–5.

[5] T-NOVA project. [Online]. Available: http://www.t-nova.eu

[6] OpenFlow. [Online]. Available: https://www.opennetworking.org/sdn-resources/openflow

[7] "Cisco visual networking index: Forecast and methodology, 2014–2019," Cisco Systems Inc., Tech. Rep. CISCO White Paper No. c11-481360, May 2015.

[8] S. Cook, *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*, ser. Applications of GPU Computing Series. Elsevier Science, 2012. [Online]. Available: https://books.google.it/books?id=EX2LNkSqViUC

[9] J. Ostermann, J. Bormans, P. List, D. Marpe, M. Narroschke, F. Pereira, T. Stockhammer, and T. Wedi, "Video coding with h.264/avc: tools, performance, and complexity," *Circuits and Systems Magazine, IEEE*, vol. 4, no. 1, pp. 7–28, First 2004.

[10] P. Wilkins, Y. Xu, L. Quillio, J. Bankoski, J. Salonen, and J. Koleszar, "VP8 Data Format and Decoding Guide," ISE RFC 6386, Oct. 2015. [Online]. Available: https://rfc-editor.org/rfc/rfc6386.txt

[11] "The webm project." [Online]. Available: http://www.webmproject.org/

[12] M. Schwalb, R. Ewerth, and B. Freisleben, "Fast motion estimation on graphics hardware for h.264 video encoding," *Multimedia, IEEE Transactions on*, vol. 11, no. 1, pp. 1–10, Jan 2009.

[13] VideoLAN x264 Project. [Online]. Available: http://www.videolan.org/developers/x264.html

[14] M. C. . VA, "Network functions virtualization - introductory white paper," October 2012.

[15] NVidia Quadro M4000 Graphic Processing Unit datasheet. [Online]. Available: http://images.nvidia.com/content/pdf/quadro/data-sheets/12489_NV_DS_Quadro_M4000_US_NV_FNL_HR.pdf

[16] C. T. Yang, H. Y. Wang, W. S. Ou, Y. T. Liu, and C. H. Hsu, "On implementation of gpu virtualization using pci pass-through," in *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*, Dec 2012, pp. 711–716.

[17] Libav. [Online]. Available: http://libav.org/documentation/

[18] "Nvidia video encoder application note," NVIDIA Inc., Tech. Rep. NVENC_DA-06209-001_v07, 2015.

[19] InfluxData InfluxDB project. [Online]. Available: http://docs.influxdata.com/influxdb/v0.9/

[20] Grafana project. [Online]. Available: http://grafana.org