ISSN 1213-6972 Journal of WSCG http://www.wscg.eu Vol.28, No.1-2, 2020

Real-time Service-based Stream-processing of High-resolution Videos

Florian T. Wagner
Hasso Plattner Institute,
Digital Engineering Faculty,
University of Potsdam, Germany
florian.wagner@student.hpi.de

Jürgen Döllner
Hasso Plattner Institute,
Digital Engineering Faculty,
University of Potsdam, Germany
juergen.doellner@hpi.de

Matthias Trapp
Hasso Plattner Institute,
Digital Engineering Faculty,
University of Potsdam, Germany
matthias.trapp@hpi.de

ABSTRACT

This paper reports on a service-based approach to enable real-time stream-processing of high-resolution videos. It presents a concept for integrating black-box image and video processing operations into a streaming framework. It further describes approaches to optimize data flow between the processing implementation and the framework to increase throughput and decrease latency. This enables the composition of streaming services to allow scaling for further throughput increase. We demonstrate the effectiveness of our approach by means of two real-time streaming-processing application examples.

Keywords: stream processing, high-resolution video, real-time

1 INTRODUCTION

1.1 Motivation and Challenges

This work aims at executing fast video processing operations, such as color enhancement or stylization, in a service-based environment using streaming. Currently, such operations are performed on the user's own mobile or fixed device using specialized software and hardware, e.g., Graphics Processing Units (GPUs). However, the increasing complexity of such operations coupled with the increase of spatial and temporal resolution of the input data requires more powerful hardware. Additionally, more complex operations need to be optimized for each possible target device to achieve good performance, requiring high investments in developer time even after the initial development. As the user expects low latency feedback of the chosen combination of video, operation, and parameters, extensive online preprocessing approaches becomes unfeasible. The preprocessing cannot be performed in an offline fashion, as the video data may be read-only and transient, e.g., in a live-streaming scenario. Therefore a suitable approach has the following challenges/constraints:

Low Latency (C.1): In order for the user to effectively choose an operation/parameter combination, the latency to process videos should be in the sub-second range.

High Throughput (C.2): An approach should be able to process high-resolution videos with state-of-the-art operations with at least the same frame rate as the input video.

Hardware Independence (C.3): The user should be able to use the approach on arbitrary devices. This especially includes mobile devices such as smart phones or tablets.

Low Integration Costs (C.4): The overhead to integrate a given operation into the processing approach should be low. This is necessary to provide the user with new and updated operations in a short time frame, e.g., to facilitate short time-to-market.

Challenges C.1 and C.2 represent the requirement for an efficient end-to-end solution. This means that the system is required not only to introduce a small performance overhead, but also needs to improve the performance of the offered video processing operations.

1.2 Optimization Approaches

There are multiple combinable approaches to improve the performance of video processing operations. Each of these approaches is based on a trade-off: some sacrifice quality, others development time. The major approaches and their trade-offs are as follows:

Preprocessing (A.1): This approach speeds up computation by preprocessing input into a format that is advantageous for further processing. This often comprises an increase of data locality or indexing operations. While preprocessing can deliver significant improvements, it always introduces additional latency. On top of this there is a trade-off between generality of the preprocessing and the obtained speedup.

Manual Optimization (A.2): This approach increases hardware utilization through operations and hardware specific optimizations. The speedup gained is proportional to the time invested and the skill of the optimizing engineer. This approach can not improve the performance of operations that are already sufficiently optimized. Additionally, this approach requires a thorough understanding of the implementation to be optimized and thus further introduces development costs for new operations.

Approximate Computing (AC) (A.3): This approach sacrifices some quality to speed up the computation. It is often not possible to do this without making modifications to the operation implementation, due to the tight coupling between the implementations and the underlying graphics libraries and hardware drivers. This means that additional developer time is required to either introduce approximations into the operation implementation itself or bring it into a form where it can be automatically approximated.

Domain-specific Language (DSL) (A.4): The above optimizations can often be made in an automated fashion, if the operation is described in a DSL. These automated optimizations often outperform manual optimizations both in achieved performance as well as performance to development overhead ratio. However, most implementations are not developed using these DSLs and translating/transpiling them will still incur a per-operation overhead.

Task Parallelization (A.5): While data parallelism is utilized in current video processing operations, their scope is too limited to implement task parallelism. Task parallelism can instead be added on a higher systemic level to fully utilize all available hardware. Because task parallelism is provided at a higher level, it can be implemented only once and then reused across multiple video processing operations and hardware architectures. However, it still requires the user to provide enough hardware to meet their performance requirements.

Off-device Processing (A.6): By performing the computation on a different device, the user is freed from the cost of operations and ownership of current processing hardware. Additionally they can easily scale the hardware to their performance requirements. This approach incurs additional latency when transferring the initial video to the computation device and the processed video to the consumption device. However, if the initial video is not already present on the consumption device, then the latency for transferring the initial video might even be lower with this operation, due to higher bandwidth of the processing hardware.

As can be seen, none of these approaches alone is capable of solving all the challenges. Additionally, some approaches require access to the operation's source code to perform effectively, rendering them impossible in scenarios where the implementations are provided by a third party. Therefore, a combination of approaches seems promising.

1.3 Approach and Contributions

Our approach combines parallelization (A.5) with off-device processing (A.6) to alleviate the challenges associated with current video processing systems. Off-device processing allows usage of high-powered processing hardware without the need to actually own this hardware. The added transmission latency is low and reduces with fast connections, increasingly. Additionally, processing latency is lowered due to more powerful processing hardware, yielding a low total latency. Further, parallelization is used to gain not just the benefit of more powerful but also more plentiful processing devices to be had with off-device processing. This further improves throughput and potentially can also lower the processing latency even further.

To summarize, this paper makes the following contributions: (1) it presents a concept for integrating black-box software components for image and video processing operations into a streaming framework, (2) it describes approaches to optimize data flow between the processing implementation and the framework to increase throughput and decrease latency, (3) it enables composition of streaming services to allow scaling for further throughput increase.

The remainder of this paper is structured as follows. Sec. 2 reviews related and previous work w.r.t. various optimization approaches of image and video processing operations and and service-based processing in general. Sec. 3 introduces the concept for enabling real-time service-based stream processing of high-resolution videos. Sec. 4 briefly describes implementation aspects of our proposed concept. Sec. 5 discusses the presented approach and implementation by means of different applications examples and a performance evaluation. Finally, Sec. 6 concludes this paper and outlines future research directions.

2 RELATED WORK

There is a vast body of related work w.r.t. video streaming and processing. This section focus on optimization approaches and specifics of service-based provisioning for video transformation operations.

2.1 Domain-specific Languages

DSLs are very appealing as they allow division of optimizations from algorithm description. Additionally they offer a degree of platform independence, as they abstract away the hardware specifics without sacrificing performance. One of the simplest optimizations is improving the locality of data and the parallelism of computation for a given operation [28]. Further domain knowledge helps with optimizing for specific hardware may be directly captured by the language [17]. Alternatively it may be specified separately from the algorithm, thus an optimization expert can quickly tune new operations for specific hardware. Some frameworks enable optimization that operates over all operations, therefore using all available information about a given operation [21]. Additionally, the abstraction provided by a

DSL can enable higher developer productivity without sacrificing performance [29]. In the same vein, DSLs can enable optimizations that are outside the usual domain expertise of developers [20]. They can enable developers to automatically optimize for distributed systems [1]. On the other end of the scale, they can be used to use more powerful but specialized hardware than is usual for image and video processing operations, such as ASICs and FPGAs [26].

2.2 Approximate Computing

A different approach is found in AC, which trades quality for performance. One major aspect of AC are the singular operations that speed-up computation, such as skipping samples in an image [15] or modifying a kernel [16]. These have been accumulated into collections that can be automatically applied to existing kernels and optimized in a subsequent tuning phase, either by selecting between different kernels [31] or by generating variations of the original kernel [30] based on a quality target. Further approaches use alternative representations of the input to efficiently sample the approximation-parameter space [13]. This tuning has been extended and refined for video processing [41]. All these operations have in common that they *intrusively modify* the algorithm and trade quality for performance. In contrast thereto, our approach does not rely on preprocessing of algorithms and avoids quality control directed by the user. However, the above operations can be used in combination with our approach.

2.3 Visual Media Processing Services

Several software architectural patterns are feasible for implementing service-based image-processing [5, 22]. However, one prominent style of building a web-based processing system for any data is the service-oriented architecture [35]. It enables server developers to set up various processing endpoints, each providing a specific functionality and covering a different use case. These endpoints are accessible as a single entity to the client, i.e., the implementation is hidden for the requesting clients, but can be implemented through an arbitrary number of self-contained services. Since web services are usually designed to maximize their reusability, their functionality should be simple and atomic. Therefore, the composition of services [9] is critical for fulfilling more complex use cases [14]. The two most prominent patterns for implementing such composition are *choreography* and *orchestration* [23]. The choreography pattern describes decentralized collaboration directly between modules without a central component. The orchestration pattern describes collaboration through a central module, which requests the different web services and passes the intermediate results between them [27].

In the field of image analysis, Wursch *et al.* [39, 40] present a web-based tool that enables users to perform various image analysis methods, such as text-line extraction, binarization, and layout analysis. It is implemented using a number of Representational State Transfer (REST) web services and application examples include multiple web-based applications for different use cases. Further, the viability of implementing image-processing web services using REST has been demonstrated by Winkler *et al.* [37], including the ease of combination of endpoints. Another example for service-based image-processing is Leadtools (https://www.leadtools.com), which provides a fixed set of approx. 200 image-processing functions with a fixed configuration set via a web Application Programming Interface (API). In this work, however, a similar approach using REST is chosen, although with a different focus in terms of granularity of services.

Applications with respect to medical image processing are presented by Yuan *et al.* [43] as well as Moulick and Gosh [18]. Both propose a web-based platform to present and process medical images by using server-side computing for a series of image processing algorithms. Further, in the field of geodata, the Open Geospatial Consortium (OGC) set standards for a complete server-client ecosystem. As part of this specification, different web services for geodata are introduced [19]. Each web service is defined through specific input and output data and the ability to self-describe its functionality[42]. In contrast, in the domain of general image-processing there is no such standardization yet. However, it is possible to transfer concepts from the OGC standard, such as unified data models. These data models are implemented using a platform-independent operation format [4]. In the future, it is possible to transfer even more concepts set by the OGC to the general image-processing domain, such as the standardized self-description of services.

2.4 Distributed Processing Approaches

Over the last decade, a number of generalized approaches to distributed processing of streaming data have been developed. The most important ones, as identified by Karimov *et al.* [10] are Apache Storm [34], Spark [44], and Flink [3]. While these frameworks are developed for moving data between processing nodes, they do not address processing of image data on GPUs, which is required for high-throughput video processing applications. With respect to this, Scanner [25] is a system that is optimized for these kinds of applications. However, it does require preprocessing of the input data and does not work on a stream-based abstraction.

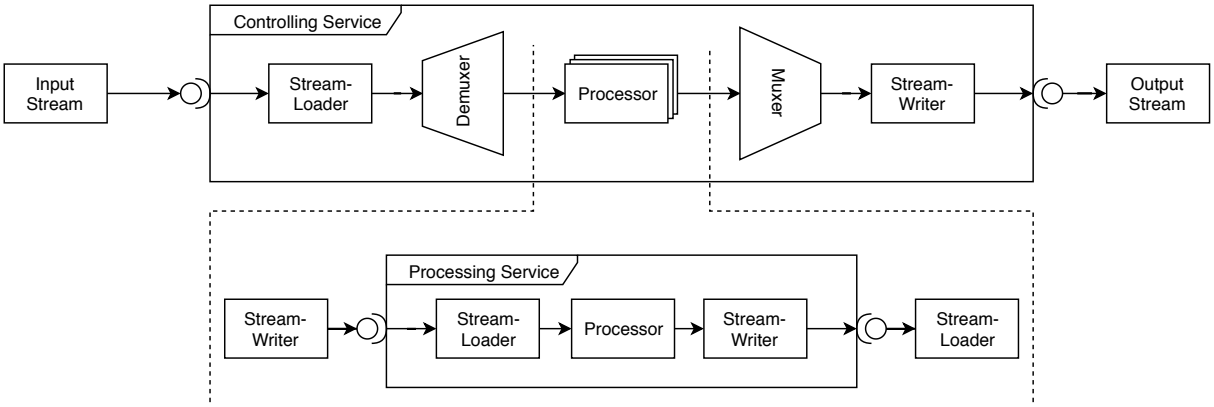


Figure 1: Conceptual overview comprising the components as well as data and control flow of the processing pipeline.

2.5 Visual Media Processing Operations

In this work, we focus on edge-aware and content-preserving image-processing as a fundamental tool in computational photography and non-photorealistic rendering for abstraction and artistic stylization for application and testing purposes. Typical approaches that operate in the spatial domain for abstraction use a kind of anisotropic diffusion [24, 36] and are designed for parallel execution, such as approximated by the bilateral filter [33] and guided filter [8].

A plenitude of stylization operations exist using these filters as building blocks to simulate traditional painting media and effects [12], such as cartoon [38] and oil paint [32]. However, these may become computationally expensive when applied in an iterative multi-stage process. This particularly applies to operations using global optimizations to separate detail from base information, e.g., based on weighted least squares [6] or locally weighted histograms [11], and recent operations that separate style from content using neural networks [7]. Because of their global optimization scheme, they are typically not suited for real-time application, in particular not on mobile devices. To this end, we implemented a variety of these operations using the proposed image-processing service including stylization, High Dynamic Range (HDR) tone mapping and compression, JPEG artifact removal and colorization, to demonstrate its versatile application. We used a representative subset of these operations for performance evaluation.

3 CONCEPTUAL OVERVIEW

This section describes our approach to real-time service-based stream processing of high-resolution videos. Based on preliminaries and assumptions, a conceptual overview of our system is given (Sec. 3.1), and components are described (Secs. 3.2 to 3.4).

3.1 Preliminaries & Assumptions

We base our work on software previously developed. For the handling of video data we use a framework designed to load videos and process them frame by frame, based on a pipeline concept. For the processing of the video frames, we use a software component comprising state-of-the art image abstraction operations that is called using a library interface.

Additionally, we make assumptions about the processing hardware and operation implementations used by our approach. We do not assume that the video is on the local host or has even been fully recorded when processing starts, as is the case in live-streaming contexts. Offline processing is therefore not possible. Additionally we assume that all operations work on a locality of the time and space domains. We also assume that the operation is being callable through a library interface and therefore resides in the same process space as our video processing pipeline. This means that memory and other resources (such as GPU handles) can be easily shared between the operation processor and the video pipeline, if the processor exposes appropriate interfaces. Finally, we assume the software component to be a black-box implementation, i.e., we cannot change or rely on the implementation details of the operation apart from the assumptions made above.

Fig. 1 shows a conceptual overview comprising the components as well as data and control flow of the processing pipeline of the proposed approach. It basically comprises the following components, which are described in greater detail in the remainder of this section; components (1) for receiving and sending video streams, (2) for tiling video in the spatial or temporal domain, and (3) that wrap the video processing operation.

3.2 Stream Separation (De-Muxing)

There are two domains, in which we can separate the data into tiles for parallel processing: spatial and temporal. If the applied processing for a target sample con-

siders source samples at multiple points in a domain, then an additional border around the original tile is provided by the split. This prevents quality degradation due to missing source samples. The size of this border depends on the range of source samples in the given domain and is therefore dependent on the operation and its parameters. Because the software component is a black box, is is not possible to exempt the border from processing. Computational overhead is therefore introduced by each sample in the border. Because of this, the relative size of the border compared to the content should be minimized to minimize overhead.

Spatial Stream Separation. Spatial Separation splits the video in the spatial domain. Each spatially split frame produces exactly one frame-tile for each parallel strand of execution. If a border is required, it is added to the frame-tile dimensions. As an optimization the separation can be performed in only one dimension of the spatial domain. This eliminates borders in the other dimensions and is especially effective for large frame-tiles and operations that necessitate large borders. Spatial separation also facilitates computing operations using limited resources; While computation time is potentially unbounded, Video Random Access Memory (VRAM) and other hardware resources are limited. The usage of these resources is often coupled to the size of the input and output images. Spatial separation enables the processing of complex and resource intensive operations on hardware, which is otherwise not capable of handling these on the requested input complexity.

Temporal Stream Separation. Temporal separation is of advantage whenever the operation is *time-local*. In this case, no border frames need to be created and therefore the amount of data that is transferred between the distributed stages is minimized. Additionally, this method is the only possible method when the technique is global in the spatial domain. If the technique is also non-local in the time domain, border frames need to be introduced. To ensure a good ratio of content vs. border, multiple consecutive frames should be sent to each node for processing. Depending on the throughput of the nodes, this will introduce a significant increase in latency, as the border frames need to be processed before the content frames.

3.3 Per-Frame Video Processing

The actual video processing module is accessed via an API. This necessitates wrapping the module within a stage to embed it in our pipeline. The wrapper requires a way to pass data into and out of this processing module. To this end, we implement two approaches to pass data between the wrapper and the processing module, both exhibiting different computational overhead and flexibility. The first approach passed data via shared memory. If the processing is performed on the GPU, the overhead might significantly deteriorate the throughput. Additionally, it can not be guaranteed that the module does not first copy the data for processing, leading to potentially more overhead. The second approach is passing the data as a texture in a shared GPU context. To enable this, a GPU context can be obtained from the module. This context is then used to allocate the respective data memory in VRAM. The handles to this memory are then passed into the module for processing. Even if the data is copied, the asynchronous and parallel nature of GPU processing allows progress to be made on other work items while the data is copied.

3.4 Stream Compositing (Muxing)

Stream muxing combines the previously separated data into a final composition result. It receives the data, which is not defined as border values and stitches them to create the final video stream. Therefore, it is required to work in the same domain as the separation operation. If the separation is performed in the spatial domain, then the frame tiles are copied into a final frame representation. If the separation is performed in the temporal domain, then sufficient frames need to be buffered to unblock the processing nodes. These frames are then aligned into the same order as their originating frames, discarding border frames if they exist. This stage also maintains the synchronization with the input audio stream.

4 IMPLEMENTATION ASPECTS

This section describes implementation specifics for our approach. Based on the concept, the implementations of the codec handling (Sec. 4.1), separation and recombination (Sec. 4.2), as well as the service-based specifics (Sec. 4.3) are covered.

4.1 GPU-based En-/Decoding

Modern GPUs often have dedicated units for de- and encoding of videos in commonly occurring formats. Especially the encoders offer a significantly higher performance than the software implementations run on Central Processing Unit (CPU). Because of this our framework allows to utilize GPU de- and encoders where supported by the hardware. If the processing of the frames is also implemented on the GPU then an additional advantage may be gained by forgoing the uploading and downloading of data to and from the GPU. However, the interface of the GPU coding units might not always be exposed in the same GPU abstraction as the processing is required.

To bridge this gap, we support translation of the data between different GPU abstractions. This happens transparently to the processing module. Due to driver limitations, registering a resource from one GPU abstraction in another might introduce an additional allocation of memory. This might lead to performance

degradation or even memory shortage if performed every frame. To this end we implement the adapter as a texture that is registered once at the initialization of the pipeline and then copied to and from. While this introduces additional data copies on the GPU, it does not degrade performance and allows processing of large data even on drivers where registering resources allocates memory.

4.2 Tiling Implementation

We use different implementations for the tiling and compositing stages of the pipeline, depending on the domain of the division and the location of the data. If data separation is performed on the spatial domain, it is implemented using a copy operation, depending on the location where the data resides. Data residing on the GPU is separated using texture copies, while data residing on the CPU is separated using *memcpy*. This means, that at least twice the memory footprint of a single frame is consumed during spatial separation - more if a border is present (e.g., required for neighborhood filtering). In effect, this overhead is often negligible since it only occurs during the separation itself, i.e., only one frame at a time consumes twice the amount of memory. Such additional use of resources is significantly smaller than the use of resources required to have multiple frames in flight.

If the separation should be performed in the temporal domain, it can be implemented by routing individual or chunks of frames to different processing stages. If a temporal border is required, all frames that are within a border need to be routed to multiple processing stages. Because the processing stages do not consume the memory of the frames, no copies need to be made during this routing. Copies are often made during transmission to different nodes, such that memory overhead might occur in this case as well. In a worst case scenario, each frame consumes $(1 + \mathit{numberOfInstances}) \cdot \mathit{memoryOfSingleFrame}$ memory at any point in time. Similar to spatial separation, this only occurs for at most one frame at a time, thus the total memory needed is bounded and near constant in a high-throughput pipeline.

4.3 Provisioning and Streaming

Service Provisioning. In order to rapidly deploy the service to new hosts, it is bundled as a Docker image [2]. This allows instancing up the respective image on different hosts without rebuilding the complete system. However, due to the required tight coupling to the GPU, all required processing libraries and APIs (such as Open Graphics Library (OpenGL) or CUDA) need to be present and supported on the host. This is a limitation of Docker with respect to GPUs. Additionally, as GPUs are often limited regarding the number of physical en/decoding units, using more than one instance per graphics card can lead to a degradation of performance.

Application of Streaming Protocols. There are three ways in which data may be streamed by the system: (1) ingesting streams from outside the system, (2) streaming between system instances, and streaming outside the system to consumers. All three cases raise different requirements on the streaming protocols used: For ingestion, common existing streaming protocols should be supported to enable easy interaction with existing stream sources, such as YouTube or similar. For streaming inside the system, low latency and high throughput is desired, so as to not degrade performance when scaling over smaller nodes. For streaming to the consumer, the protocol should be supported by web-browsers for easy consumption. For consumption, it is further desired to allow for some degree of asynchronicity, as the user might drop-out of the session and return to it at a later point in time.

Because of these differing requirements, we choose to support different protocols as follows. For stream ingestion and streaming inside the system, Real-Time Messaging Protocol (RTMP) was chosen. This protocol allows for low overhead, low latency streaming and is used by popular stream ingestion APIs such as Twitch. For streaming to the consumer we use Hypertext Transfer Protocol (HTTP) Live Streaming (HLS). This protocol allows to use readily available technologies on both ends of the connection. The server can be a HTTP driven and does not require prior knowledge about the nature of the video. On the client side the stream can be consumed using a standard video player in the web browser. Additionally, this allows for asynchronicity as the processed stream can be stored for as long as desired.

5 RESULTS & DISCUSSION

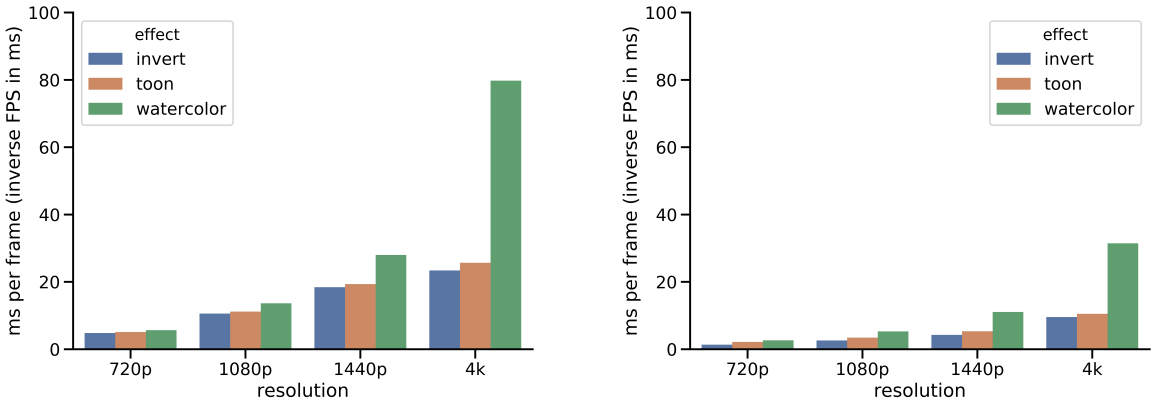
This section demonstrates our approach by means of different application examples (Sec. 5.1) and discusses its runtime performance as well as technical and conceptual limitations (Sec. 5.2).

5.1 Application Examples

We tested our approach and its service-based integrations by means of different applications in the domain of web-based video processing as follows (please refer also to the accompanying video).

Interactive Web-based Video Editing. This application example demonstrate how the streaming can be used to provide visual feedback of chosen operations and parameters.The user first selects a video, specific processing operations parameters and triggers processing. Subsequently, a preview of the video with the operations applied is presented to the user.

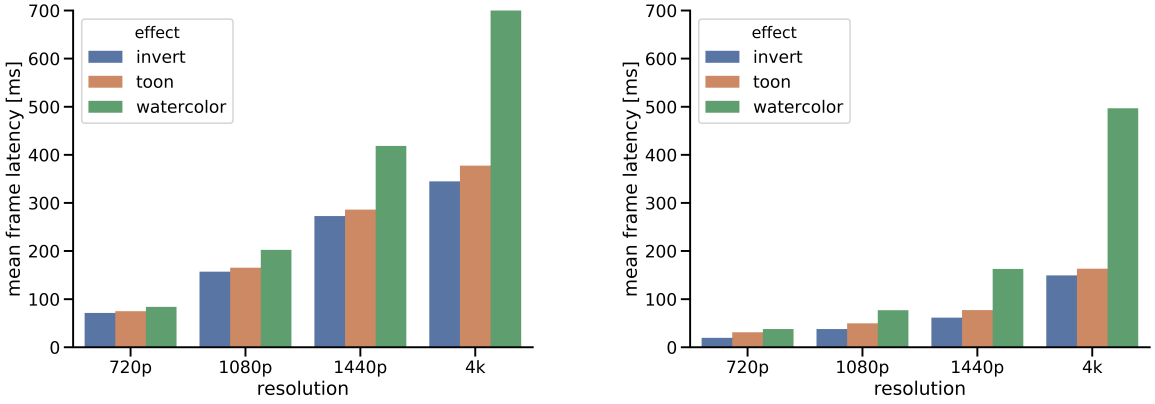
The user can easily chain multiple operations without an impact on the responsiveness of the application. This



(a) NVIDIA GTX 1050.

(b) NVIDIA GTX1080.

Figure 2: Throughput measurements in seconds using a consumer desktop (a) and a dedicated server (b).



(a) Latencies for different operations.

(b) Latencies for different spatial resolutions.

Figure 3: End to end frame latencies on a consumer desktop (a) and a dedicated server (b)..

is possible even if the underlying processing framework does not support chaining per se. Once the user has decided on a set of operations and parameters, they can download the result.

Real-time Stream-processing of Videos. This application example demonstrates how a consumer can take existing videos hosted online and process them using our system. The user provides the link to a video hosted on a video streaming platform, chooses an operation to apply to this video, and starts the pipeline to consume the processed video. As the video is processed in a streaming fashion, the user can start watching the video before it is fully processed.

5.2 Performance Evaluation

With respect to the application examples, a performance evaluation using different test data sets and operations are performed.

Test Data and Operations. Different image resolutions were tested with different image-processing techniques to estimate the run-time performance regarding the spatial resolution of a video as well as the complexity of processing techniques. The following common spatial resolutions (in pixels) were chosen: 1280×720 (HD), 1920×1080 (FHD), 2560×1440 (QHD), and 3840×2160 (4K). The source videos had a frame rate of 24 Frames-per-Second (FPS) and a total duration of 150 s each.

In addition to various spatial resolutions, different image processing techniques were tested in order to cover a broad spectrum and obtain variant performance estimates and behavior with respect to different types of processing tasks. These operations range in complexity from simple point based operations (invert) to complex stylization operations (watercolor).

Test Hardware and Setup. To cover a common hardware spectrum, the performance test are conducted on two different test hosts: (1) a desktop PC with commodity hardware with an Intel i5-4690 CPU (4 cores) at 3.5 GHz, 16 GB Random Access Memory (RAM), and

GeForce GTX 1050 Ti GPU 4 GB VRAM as well as a (2) dedicated GPU server with an Intel i7-7700 CPU (4 cores) at 3.6 GHz, 64 GB RAM, and a GTX 1080 GPU 24 GB VRAM.

The videos were streamed in and out of the processor by separate processes running on the same host. All configurations are run 20 times to eliminate external factors from the measurements.

Performance Results and Discussion. As depicted in Fig. 2, the system is capable of processing the given effects in real-time on the consumer device. For high-resolution videos processed with compute-intensive effects, more powerful hardware is required.

As depicted in Fig. 3, the presented approach achieves sub-second latencies for the processing of all given effects. Most of this latency is incurred by the batching of frames on the codec, as is evidenced by the comparatively similar latency over the different effects per resolution. Only the watercolor effect introduces significant latency.

Inspection of the running process shows that we achieve close to 100 % GPU-utilization using our system. This means that our system is capable of fully utilizing the GPU with processing the operations. Whether this utilization represents a relatively high or low throughput depends (1) on the input data, (2) the chosen operations and parameters, and (3) the implementation of the operations in the given processing system. However, the presented approach can not overcome bandwidth limitations inside or between processing nodes, since frames are not preprocessed. This means that the connection to the video source must be sufficiently fast to deliver the video data in real-time. With respect to processing hardware, we assume that each node has the capabilities to process two encoded streams of the desired tile resolution. This includes the encoder and decoder on this node being capable of processing the stream at a frame rate that is greater or equal to the frame rate of the input video.

6 CONCLUSIONS

This paper presents a concept and prototypical implementation to integrate 3rd-party image and video processing software components into services cable of real-time streaming. The system achieves this without the modification or optimization of the specific processing algorithm's implementations. It supports scaling-up and scaling-out to cover different input and processing complexities. This paper further presents use cases in which such an approach is required to facilitate new applications or features for users.

For future work, this system could be improved to be more adaptive to the kind of workload that is imposed. This includes automated provisioning of new nodes to scale-out as well as easy re-use of provisioned nodes to handle multiple tasks without incurring start-up and tear-down costs. In addition thereto, the system's behavior can be runtime-adaptive, thus the appropriate scale can be computed automatically – even for unknown operations or input data complexities. To support this, research with respect to tuning the parallelization process to balance the overhead associated with tiling and the gain in performance must be conducted. To lower latencies in complex processing pipelines, a transitioning from traditional video streaming protocols to new data exchange protocols such as a messaging queue, can be considered.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their valuable feedback. This work was funded by the Federal Ministry of Education and Research (BMBF), Germany, for the "mdViPro" project (01IS15041).

REFERENCES

[1] Riyadh Baghdadi, Jessica Ray, Malek Ben Romdhane, Emanuele Del Sozzo, Abdurrahman Akkas, Yunming Zhang, Patricia Suriana, Shoaib Kamil, and Saman Amarasinghe. Tiramisu: A Polyhedral Compiler for Expressing Fast and Portable Code. *arXiv:1804.10694 [cs]*, 2018.

[2] Carl Boettiger. An Introduction to Docker for Reproducible Research. *SIGOPS Oper. Syst. Rev.*, 49(1):71–79, January 2015.

[3] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. Apache flink^{TM}: Stream and batch processing in a single engine. *IEEE Data Eng. Bull.*, 38(4):28–38, 2015.

[4] Tobias Dürschmid, Maximilian Söchting, Amir Semmo, Matthias Trapp, and Jürgen Döllner. ProsumerFX: Mobile Design of Image Stylization Components. In *SIGGRAPH Asia 2017 Mobile Graphics & Interactive Applications*, SA '17, pages 1:1–1:8, New York, NY, USA, 2017. ACM.

[5] Thomas Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[6] Zeev Farbman, Raanan Fattal, Dani Lischinski, and Richard Szeliski. Edge-preserving Decompositions for Multi-scale Tone and Detail Manipulation. *ACM Transactions on Graphics*, 27(3):67:1–67:10, 2008.

[7] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Image Style Transfer Using Convolutional Neural Networks. In *Proc. Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2414–2423, Los Alamitos, 2016. IEEE Computer Society.

[8] Kaiming He, Jian Sun, and Xiaoou Tang. Guided Image Filtering. In *Proc. European Conference on Computer Vision (ECCV)*, pages 1–14. Springer, 2010.

[9] Alexander Jungmann and Bernd Kleinjohann. Automatic Composition of Service-based Image Processing Applications. In *Proc. IEEE International Conference on Services Computing (SCC)*, pages 106–113. IEEE Computer Society, 2016.

[10] Jeyhun Karimov, Tilmann Rabl, Asterios Katsifodimos, Roman Samarev, Henri Heiskanen, and Volker Markl. Benchmarking distributed stream data processing systems. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 1507–1518. IEEE.

[11] Michael Kass and Justin Solomon. Smoothed Local Histogram Filters. *ACM Transactions on Graphics*, 29(4):100:1–100:10, 2010.

[12] Jan Eric Kyprianidis, John Collomosse, Tinghuai Wang, and Tobias Isenberg. State of the 'Art': A Taxonomy of Artistic Stylization Techniques for Images and Video. *IEEE Transactions on Visualization and Computer Graphics*, 19(5):866–885, 2013.

[13] Michael A. Laurenzano, Parker Hill, Mehrzad Samadi, Scott Mahlke, Jason Mars, and Lingjia Tang. Input Responsiveness: Using Canary Inputs to Dynamically Steer Approximation. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '16, pages 161–176. ACM, 2016.

[14] Angel Lagares Lemos, Florian Daniel, and Boualem Benatallah. Web Service Composition: A Survey of Techniques and Tools. *ACM Computing Surveys*, 48(3):33:1–33:41, 2015.

[15] Liming Lou, Paul Nguyen, Jason Lawrence, and Connelly Barnes. Image Perforation: Automatically Accelerating Image Pipelines by Intelligently Skipping Samples. *ACM Trans. Graph.*, 35(5):153:1–153:14, 2016.

[16] Daniel Maier, Biagio Cosenza, and Ben Juurlink. Local Memory-aware Kernel Perforation. In *Proceedings of the 2018 International Symposium on Code Generation and Optimization*, CGO 2018, pages 278–287. ACM, 2018.

[17] Richard Membarth, Oliver Reiche, Frank Hannig, Jurgen Teich, Mario Korner, and Wieland Eckert. HIPA^{cc}: A domain-specific language and compiler for image processing. *IEEE Transactions on Parallel and Distributed Systems*, 27(1):210–224, 2016.

[18] Himadri Nath Moulick and Moumita Ghosh. Medical image processing using a service oriented Architecture and Distributed Environment. *American Journal of Engineering Research (AJER)*, 2(10):52–62, 2013.

[19] Matthias Mueller and Benjamin Pross. *OGC WPS 2.0.2 Interface Standard*. Open Geospatial Consortium, 2015. http://docs.opengeospatial.org/is/14-065/14-065.html.

[20] Ravi Teja Mullapudi, Andrew Adams, Dillon Sharlet, Jonathan Ragan-Kelley, and Kayvon Fatahalian. Automatically Scheduling Halide Image Processing Pipelines. *ACM Transactions on Graphics*, 35(4):83:1–83:11, July 2016.

[21] Ravi Teja Mullapudi, Vinay Vasista, and Uday Bondhugula. PolyMage: Automatic Optimization for Image Processing Pipelines. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '15, pages 429–443. ACM, 2015. event-place: Istanbul, Turkey.

[22] Mike P. Papazoglou and Willem-Jan Heuvel. Service Oriented Architectures: Approaches, Technologies and Research Issues. *The VLDB Journal*, 16(3):389–415, 2007.

[23] Chris Peltz. Web Services Orchestration and Choreography. *Computer*, 36(10):46–52, October 2003.

[24] Pietro Perona and Jitendra Malik. Scale-space and Edge Detection using Anisotropic Diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(7):629–639, 1990.

[25] Alex Poms, Will Crichton, Pat Hanrahan, and Kayvon Fatahalian. Scanner: Efficient video analysis at scale. *ACM Trans. Graph.*, 37(4), July 2018.

[26] Jing Pu, Steven Bell, Xuan Yang, Jeff Setter, Stephen Richardson, Jonathan Ragan-Kelley, and Mark Horowitz. Programming Heterogeneous Systems from an Image Processing DSL. *ACM Trans. Archit. Code Optim.*, 14(3):26:1–26:25, 2017.

[27] Ricardo Queirós and Alberto Simões. SOS - Simple Orchestration of Services. In Ricardo Queirós, Mário Pinto, Alberto Simões, José Paulo Leal, and Maria João Varanda, editors, *6th Symposium on Languages, Applications and Technologies (SLATE 2017)*, volume 56 of *OpenAccess Series in Informatics (OASIcs)*, pages 13:1–13:8, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[28] Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, FrÃ©do Durand, and Saman Amarasinghe. Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image Processing

Pipelines. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '13, pages 519–530. ACM, 2013. event-place: Seattle, Washington, USA.

[29] Mahesh Ravishankar, Justin Holewinski, and Vinod Grover. Forma: A DSL for Image Processing Applications to Target GPUs and Multi-core CPUs. In *Proceedings of the 8th Workshop on General Purpose Processing Using GPUs*, GPGPU-8, pages 109–120. ACM, 2015. event-place: San Francisco, CA, USA.

[30] Mehrzad Samadi, Davoud Anoushe Jamshidi, Janghaeng Lee, and Scott Mahlke. Paraprox: Pattern-based Approximation for Data Parallel Applications. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, pages 35–50. ACM, 2014.

[31] Mehrzad Samadi, Janghaeng Lee, D. Anoushe Jamshidi, Amir Hormati, and Scott Mahlke. SAGE: Self-tuning Approximation for Graphics Engines. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 13–24. ACM, 2013.

[32] Amir Semmo, Daniel Limberger, Jan Eric Kyprianidis, and Jürgen Döllner. Image Stylization by Interactive Oil Paint Filtering. *Computers & Graphics*, 55:157–171, 2016.

[33] Carlo Tomasi and Roberto Manduchi. Bilateral Filtering for Gray and Color Images. In *Proc. International Conference on Computer Vision (ICCV)*, pages 839–846. IEEE, 1998.

[34] Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M. Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, Nikunj Bhagat, Sailesh Mittal, and Dmitriy Ryaboy. Storm@twitter. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 147–156. ACM. event-place: Snowbird, Utah, USA.

[35] Mircea-Florin Vaida, Valeriu Todica, and Marcel Cremene. Service oriented Architecture for Medical Image Processing. *International Journal of Computer Assisted Radiology and Surgery*, 3(3):363–369, 2008.

[36] Joachim Weickert. *Anisotropic Diffusion in Image Processing*, volume 1. Teubner Stuttgart, 1998.

[37] Robert P. Winkler and Chris Schlesiger. Image Processing REST Web Services. Technical Report ARL-TR-6393, Army Research Laboraty, Adelphi, MD 20783-119, 2013.

[38] Holger Winnemöller, Sven C. Olsen, and Bruce Gooch. Real-Time Video Abstraction. *ACM Transactions on Graphics*, 25(3):1221–1226, 2006.

[39] M. Würsch, R. Ingold, and M. Liwicki. SDK Reinvented: Document Image Analysis Methods as RESTful Web Services. In *2016 12th IAPR Workshop on Document Analysis Systems (DAS)*, pages 90–95, 2016.

[40] Marcel Würsch, Rolf Ingold, and Marcus Liwicki. DivaServices - A RESTful web service for Document Image Analysis methods. *Digital Scholarship in the Humanities*, 32(1):i150–i156, 2017.

[41] Ran Xu, Jinkyu Koo, Rakesh Kumar, Peter Bai, Subrata Mitra, Sasa Misailovic, and Saurabh Bagchi. VideoChef: Efficient Approximation for Streaming Video Processing Pipelines. In *2018 {USENIX} Annual Technical Conference ({USENIX} {ATC} 18)*, pages 43–56, 2018.

[42] Xiaoxia Yang. Remotely Sensed Image Processing Service Automatic Composition. State Key Laboratory of Information Engineering in Surveying, Mapping and Remote Sensing, Wuhan University, 2009.

[43] Rong Yuan, Ming Luo, Zhi Sun, Shuyue Shi, Peng Xiao, and Qingguo Xie. Rayplus: a web-based platform for medical image processing. *J. Digital Imaging*, 30(2):197–203, 2017.

[44] Matei Zaharia, Tathagata Das, Haoyuan Li, Scott Shenker, and Ion Stoica. Discretized streams: An efficient and fault-tolerant model for stream processing on large clusters. In *Proceedings of the 4th USENIX Conference on Hot Topics in Cloud Ccomputing*, HotCloud 2012, page 10, USA, 2012. USENIX Association.