High-accuracy and video-rate lifetime extraction from Time Correlated Single Photon Counting data on a Graphical Processing Unit[a)]

A. Margara,[1] P. Peronio,[1] G. Acconcia,[1] G. Cugola,[1] and I. Rech[1]

Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Piazza Leonardo da Vinci 32, 20133 Milano, Italy

(Dated: 9 October 2019)

Graphical Processing Units (GPUs) are a powerful alternative to CPUs, especially for data-parallel, video-rate processing of large data volumes. In the complex scenario of high-performance multichannel Time Correlated Single Photon Counting (TCSPC), a huge amount of data is potentially generated by the acquisition system. Exploiting a dedicated external, programmable elaboration unit enables a high degree of flexibility to perform different types of analysis. In this paper, we present a GPU-based application that leverages the CUDA API for video-rate and accurate lifetime extraction from TCSPC data acquired at a rate of up to 10 Gbit/s.

[a)] Corresponding author: Alessandro Margara, alessandro.margara@polimi.it

I. INTRODUCTION

Time Correlated Single Photon Counting for Fluorescence Lifetime Imaging

Fluorescence Lifetime Imaging Microscopy (FLIM) has gained a prominent role in the study of biological processes occurring at the cellular and sub-cellular level as it allows high-sensitivity and non-invasive analysis of tissues. The Time Correlated Single Photon Counting (TCSPC) technique is a powerful tool to perform FLIM[1]. A TCSPC measurement basically consists in the periodical excitation of a sample by means of a pulsed laser and in the record of the time of arrival of re-emitted photons. The histogram of the recorded time of multiple photons corresponds to the waveform of the re-emitted light in the time domain and it can be used to study such waveform[2]. The extraction of the single- or multi-exponential lifetime from the histogram, for example, is extremely useful to get an insight not only on the target element (e.g., a single molecule), but also on its surrounding environment as the decay time is typically influenced by temperature, pH, and proximity of other elements as fluorescence quenchers[3]. The TCSPC approach is clearly advantageous with respect to its analog counterpart, especially when ultra-fast and weak optical signals have to be measured. Nonetheless, TCSPC is an intrinsically slow technique: the average photon detection rate must be kept well below the excitation rate (typical ratio is between 1% and 5%) to avoid the so-called pile-up distortion[2]. Moreover, to achieve a proper reconstruction of the time-domain signal, a statistically-significant number of events has to be collected resulting in a relatively long acquisition time, especially if a single channel is exploited[4].

Multichannel TCSPC acquisition systems: advantages and issues

A real breakthrough in TCSPC analysis would be the reduction of the overall time needed to perform the measurements. In the complex process of drug discovery, for example, the exploitation of FLIM analysis to investigate protein interactions could open the way to determine the exact pathway a candidate drug is modulating and then lead to discovery the best drug for a given pathology. Unfortunately, one of the most time/cost-consuming stages of drug discovery is in the identification of drug leads: thousands of samples, which have been treated with varied dosages of a large number of drug leads, have to be analyzed[5].

The use of N channels operating in parallel would reduce the overall time needed to perform the measurement by a factor of N. For this reason, in recent years there has been a

trend towards the development of multichannel TCSPC acquisition systems. Parallelism not only opens the way to speedup measurement, but it also enables advanced measurements, as Fluorescence Correlation Spectroscopy (FCS)[6], which requires the simultaneous acquisition of data from the same sample at different spots and/or the acquisition of different wavelengths of the signal at the same time.

Technology has been proven to be ready to develop densely integrated TCSPC acquisition systems featuring hundreds or even thousands of channels[7–9]. Nevertheless, densely integrated systems can potentially produce a huge amount of data (in the order of tens of Gigabits/s), which poses critical issues on data extraction and management. First of all, such a huge amount of data needs to be extracted from the acquisition system. Various readout approaches have been proposed to address this issue[7,8,10,11]. Secondly, collected information has to be sent to an elaboration unit and processed there. Several solutions have been conceived at the system level: exploitation of FPGAs along with data compression mechanism, like histogram creation, significantly reducing the amount of data to be transferred downstream, has allowed on-board data elaboration[9,12–14]. However, applications could require more complex and varied data elaboration: correlation measurements in FCS, lifetime extraction in FLIM, 3D reconstruction of scenes using single-photon lidar[15], just to name a few. Lifetime extraction, for example, can be carried out following several approaches: a traditional approach consists in fitting decays at each pixel using one or two exponentials and identifying decay times and amplitudes with molecular species and their relative abundances[1,16]; on the other hand, the recently-developed phasors approach[17,18] is based on a rapid Fourier analysis that translates lifetime information into a graphical representation called the phasor plot, in which lifetime differences across various regions of the image can be easily distinguished. The traditional approach could be carried out on board when a single time constant is present, but it becomes extremely challenging when multiple time constants concur to determine the shape of a multi-exponential signal.

Raw data transfer towards a PC can be the solution of choice to achieve very high accuracy along with the flexibility to perform different types of operations. To make external elaboration possible, a large bandwidth is required to download the huge amount of data produced by the acquisition system. Modern communication protocols such as Ethernet 10G and USB 3.1 can provide a transfer rate up to 10 Gbit/s, while optical links can reach a rate as high as 100 Gbit/s.

In this scenario, the main bottleneck in the acquisition chain becomes the PC that has to perform the actual processing of data, which inevitably limits the maximum sustainable throughput and, as a consequence, the count rate. On the other hand, traditional CPUs are not the only elaboration units available today. Modern Graphical Processing Units (GPUs) have shown to be a valid alternative to CPUs for data elaboration, particularly when data-parallel tasks have to be performed. In particular, the introduction of the Common Unified Device Architecture (CUDA)[19] has opened the way to the exploitation of GPUs in general purpose computing. In this paper, we present a GPU-based application aimed at achieving video-rate processing of data based on the traditional exponential fitting. Our solution can process a TCSPC data stream as high as 10 Gbit/s (Ethernet link) potentially generated by a 32×32 SPAD camera.

Paper outline

The paper is organized as follows: Section II describes a high-performance multichannel TCSPC acquisition system requiring a dedicated data-processing unit; Section III discusses the choice of a GPU as external elaboration unit and introduces the CUDA API; Section IV proposes a GPU-based algorithms for lifetime extraction, Section V evaluates its performance, comparing it with state-of-the-art solutions. Finally, Section VI draws some concluding remarks.

II. THE ACQUISITION SYSTEM

Fig. 1 shows a high-level view of the components of a TCSPC acquisition and processing system. In this section, we focus on the *camera* and its internals, while next sections discuss the `PC`.

The camera, under development at Politecnico di Milano, aims at breaking the tradeoff between number of channels and performance that currently limits the available TCSPC acquisition systems.

A high-performance TCSPC acquisition system should provide high Photon Detection Efficiency (PDE), combined with low noise (Dark Count Rates and afterpulsing), low timing jitter, and high linearity. Moreover, the speed of the system is a key factor in a large number of applications, as discussed in the introduction. Among currently-available systems, the

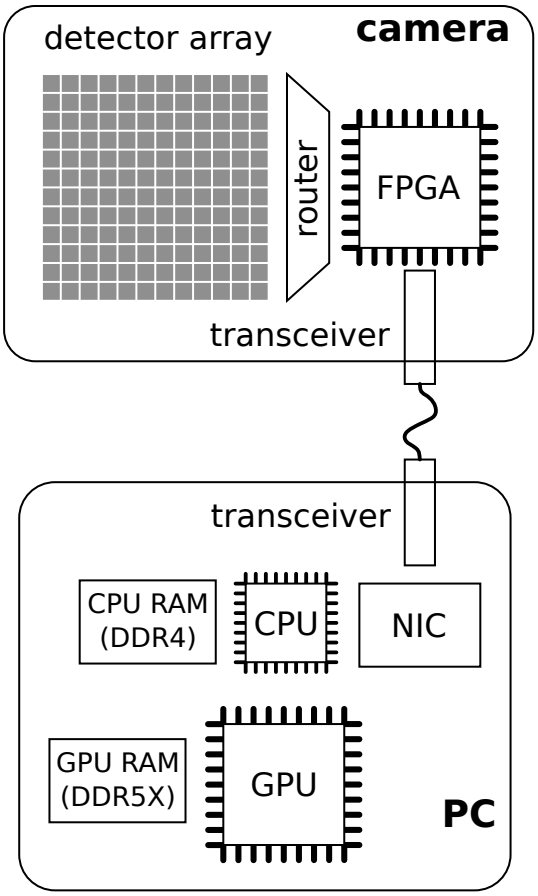

FIG. 1: The architecture of our system

best performance in terms of jitter and linearity are provided by timing commercial boards (not including the detector) which have been limited so far to a low number of channels (up to 8), while systems featuring hundreds or even thousands of channels have been designed exploiting the standard CMOS technology to integrate both the detector and the timing electronics on the same chip, but their performance are quite poor compared to the best single- or few-channels systems.

To overcome this tradeoff, a new approach has been proposed at Politecnico di Milano, which exploits different technologies for the various parts of the system, each one specifically selected to optimize a single aspect of the problem[20]. The system targets the use of custom-technology SPAD detectors, which have been proven to provide best-in-class performance in terms of a combination of PDE, noise, and detector jitter[21]. These detectors require dedicated electronics developed on purpose to attain the best performance, and in fact, in the past few years several dedicated circuits have been presented in the literature[22–24],

paving the way to the development of such a high-performance densely integrated TCSPC system.

Now, if we consider, for example, a 32×32 detector array, the system could pose crucial issues on data management, first and foremost the amount of data generated. Indeed, considering a typical excitation rate of 80 MHz and a detector count rate of 4 Mcps, with a timing measurement circuit associated to each pixel featuring two bytes to encode the timing information, the amount of data produced during a TCSPC measurement would be as high as 64 Gbit/s. Depending on the readout mechanism, this amount of data could be further increased by the data needed to identify the source pixel of each readout in the 32×32 detector array, resulting in a data rate that may exceed 100 Gbit/s: an amount that is hard to manage.

Moving from this consideration, the camera under development at Politecnico di Milano has been designed starting from the data rate that can be reasonable handled by currently available technology: a target throughput of 10 Gbit/s. Such an amount of data will be generated exploiting five conversion circuits[20] along with 10 bits to address the 32×32 detector array pixels. The exploitation of only five conversion circuits shared with a 32×32 detector array poses tight constraints on the resource sharing mechanisms to avoid distortions and inefficiencies. Acconcia et al.[20] proposed in 2016 a smart routing algorithm for data extraction in densely integrated TCSPC imagers, which has been proven to outperform the alternative readout solutions proposed so far[25]. This smart routing algorithm is able to extract five signals out of a 32×32 SPAD array at each excitation cycle and under any operating condition, while providing a fair treatment of the detectors to avoid any distortion. In this way, the acquisition system can truly generate a throughput as high as 10 Gbit/s, that can be transferred to the elaboration unit using a standard 10 Gbit/s Ethernet link.

In particular (see Fig. 1), UDP datagrams encapsulated into Ethernet frames are built on board by the camera using a dedicated FPGA module and transferred using an off-the-shelves transceiver (i.e., an optical fiber physical module).

III. DATA PROCESSING ON A GPU

Managing data produced at the rate of 10 Gbit/s pushes the hardware of modern computers to its limits. As a reference, the best Solid State Disks (SSDs) nowadays provide

between 5 Gbit/s (SATA III) and 16 Gbit/s (NVME) sequential write capabilities, depending on their connecting interface, and they are rarely bigger than 1 TByte (at 10 Gbit/s they would fill in few minutes). These numbers show the technical difficulty of even reading and storing data received at the rate of 10 Gbit/s, and motivate the need for video-rate processing of raw data.

Fortunately, the problem we address in this paper has the advantage of being intrinsically *data parallel*, meaning that it requires the very same computation to be performed on multiple partitions of the input data ($32 \times 32 = 1024$ partitions in our case, one for each detector array pixel). *Graphical Processing Units* (GPUs) provide much better performance than CPUs in data-parallel tasks, and the CUDA platform offers an innovative *Single Program Multiple Data* (SPMD) programming model and a new instruction set to leverage GPUs for general purpose programming.

The remainder of this section presents the key aspects of the programming and execution models in the CUDA platform. Next, Section IV will explain how these aspects influenced the design of our algorithm.

CUDA Programming and Execution Model

Kernels. In CUDA, developers define computations to be performed on the GPU as special functions denoted *kernels*, which outline a single flow of execution for multiple threads, organized into groups called *blocks*. When invoking a kernel k, developers specify the number of blocks to be used and the number of threads within each block. The kernel code has access to two special variables provided by the CUDA runtime: the *blockId* and the *threadId*, which together uniquely identify each thread among those executing the kernel. Blocks are independently scheduled for execution on the GPU when enough resources become available.

Execution model. Within each block, threads are further split in small groups of 32 threads called *warps*. Full hardware utilization can be achieved only if all the threads within a warp execute the exact same instructions, that is, if they do not differ in terms of control flow.

Separation of host and device. CUDA kernels execute on a physically separate *device* (the GPU), which operates as a co-processor for a *host* (the CPU) running a C/C++ program. The host and the device maintain their own separate memory spaces. Therefore, before starting a computation, it is necessary to explicitly allocate memory on the device

and copy the data needed during execution there. Similarly, at the end of the computation, results must be copied from the device to the host memory, and the device memory must be deallocated.

Memory model. CUDA threads may access data from multiple memory spaces during their execution: each thread has a *private local memory* for automatic variables; each block has a *shared memory* visible to all threads in the same block; finally, all threads have access to the same *global memory*. In this hierarchy, local memory is the fastest but smallest and global memory is the largest but slower. The memory access pattern (to global and shared memory) also affects performance. If threads with contiguous ids access contiguous memory locations, the hardware can organize the interaction into several memory-wide operations, thus maximizing throughput, otherwise memory access may easily become a bottleneck for most algorithms.

Synchronization. CUDA offers two main classes of synchronization constructs: (i) barrier synchronization within individual thread groups, (ii) atomic operations on individual variables, such as atomic compare-and-set on numerical values.

IV. LIFETIME EXTRACTION IN CUDA

Lifetime extraction can be decomposed in two phases: first, readings coming from each detector are aggregated into a histogram that counts the number of photons that fall into each delay interval; second, histograms are analyzed to determine the time constants (one or more) that describe the lifetime decay rule of the specific molecules under examination. In the remainder we denote these two phases *counting* and *lifetime extraction*, respectively. In our solution, both steps are performed by a PC that receives data produced by the camera through a 10 Gbit/s Ethernet, as shown in Fig. 1.

Protocol

We designed a simple protocol to transfer readings from the camera to the PC. It encodes each reading using 24 bits: 10 bits represent the address of the detector in the 32×32 array, while the remaining 14 bits represent the actual delay measured. Readings are packed into UDP datagrams by the FPGA of the camera as soon as they are produced, with no further processing. Each datagram holds 3200 readings, i.e., 9600 bytes, which is enough to limit

the total number of packets while avoiding fragmentation, since the Ethernet card we use (Intel 82599ES) supports jumbo frames up to 9700 bytes.

The FPGA of the camera is the master of the entire process and controls PC operations. When enough readings are collected (this is a design parameter whose impact on performance will be studied in Section V) a special *end-of-frame* datagram is sent. This is the only datagram that is acknowledged by the PC and resent if no ack is received.

Receiving data and counting on the PC

The PC receives UDP datagrams, extracts the data they contain, and collects it in RAM, before starting the counting phase.

Counting is a relatively easy algorithm whose goal is to fill $32 \times 32 = 2^{10} = 1024$ integer arrays (the *histograms*), each holding $2^{14} = 16384$ elements. At the end of the counting phase, position D of histogram A holds the number of photons observed by detector A after delay D from the laser excitation. In practice, from each reading we extract address A (first 10 bits) and delay D (last 14 bits) and we increment histogram A at position D.

This simple algorithm has a negligible computational cost but is very expensive in terms of memory accesses. The total memory occupied by the histograms for each frame is at least (if we use a short int, i.e., 16 bits, for each counter) $32 \times 32 \times 16384 \times 2 = 32$ MB, which is larger than the cache size of most microprocessors (the i7-8700 CPU we use has 12 MB of cache). This results in a large number of cache misses and the latency of the RAM used in PCs is not sufficient for our needs. Indeed, DDR4 RAM has around 14 ns of latency, which means a program may access at most 70 M "cells" per second if this latency has to be payed at each access. In practice, by implementing the counting step on the CPU we were not able to "count" more that 180 M readings per second, which is far from our target of 10 Gbit/s, i.e., 426 M readings per second. Moving from this observation, we decided to implement the counting step on the GPU, where it can be easily parallelized and can leverage the much faster GDDR5X RAM.

The resulting algorithm works as follows: the CPU interacts with the networking controller (NIC in Fig. 1) to receive UDP datagrams, extract their payload, i.e., the camera readings, and collect them in RAM with no further processing. When the end-of-frame packet is received, the CPU initiates a memory transfer of the collected readings from the CPU to the GPU and launches the `counting` kernel. This kernel uses as many parallel

threads as the number of cores available in the GPU (3584 threads in the GPU we use for our experiments). Each thread takes a portion of the readings, decodes them, separating the address from the delay, and increments the corresponding element of the appropriate histogram. Each increment is performed as an atomic operation using a `atomicInc` primitive offered by CUDA.

While the memory transfer happens and the kernel runs on the GPU, the CPU (using a separate data space) starts receiving the next frame. This way we may overlap CPU and GPU tasks, maximizing the usage of available processing resources. As an additional optimization, we also repeat the read-transfer-count sequence above 8 times before actually starting the next phase. This allows to fully exploit the GPU hardware for the lifetime extraction operation.

Lifetime extraction

Differently from counting, lifetime extraction is indeed a complex and computationally intensive task. Each histogram represents the sum of one or more exponential decay functions whose main parameters must be identified.

In this work we consider the cases of one or two exponential decays plus a noise factor, which represent the most common cases for our applications. These two cases are modeled by the equations below:

$$y = \mu + \beta\, e^{\alpha x} \tag{1}$$

$$y = \mu + \beta_1\, e^{\alpha_1 x} + \beta_2\, e^{\alpha_2 x} \tag{2}$$

and lifetime extraction is the problem of finding the values μ, α, β or $\mu, \alpha_1, \alpha_2, \beta_1, \beta_2$ in these formulas, which better approximate the data we have (i.e., the histograms).

This is a regression problem complicated by the fact that the functions to approximate are exponential and include a constant (noise) factor, such that easily reduce them to a linear regression problem. Moreover, to reach the level of performance we target, we need a non iterative regression algorithm.

This can be obtained along the lines suggested in[26], which we report here for the case of a single exponential function. In particular, integrating equation (1) we obtain:

$$\int_{x_0}^{x} y(u)\, du = \mu(x - x_0) + \frac{\beta}{\alpha}\, e^{\alpha x} - \frac{\beta}{\alpha}\, e^{\alpha x_0}$$

and reusing equation (1) for the second term of the right hand side of the equality, we obtain:

$$\int_{x_0}^{x} y(u)\ du = \mu(x - x_0) + \frac{y - \mu}{\alpha} - \frac{\beta}{\alpha}\, e^{\alpha x_0}$$

which we can rewrite as:

$$y - (\mu + \beta\, e^{\alpha x_0}) = \alpha \int_{x_0}^{x} y(u)\ du - \mu\, \alpha(x - x_0)$$

If we calculate this formula for $x = x_0$ we derive:

$$y_0 = (\mu + \beta\, e^{\alpha x_0})$$

such that:

$$y - y_0 = \alpha \int_{x_0}^{x} y(u)\ du - \mu\, \alpha(x - x_0)$$

The definite integral $S_k = \int_{x_0}^{x_k} y(u)\ du$ can be easily approximated using the trapezoidal rule, yielding:

$$\begin{cases} S_0 = 0 \quad \text{for } k = 0 \\ S_k = S_{k-1} + \frac{1}{2}(y_k + y_{k-1})(x_k - x_{k-1}) \quad \text{for } k = 1..n \end{cases}$$

such that we can derive:

$$y_k - y_0 \simeq -\mu\, \alpha(x_k - x_0) + \alpha S_k$$

To determine factors μ, β, and α, we minimize the sum of squares:

$$\sum_{k=0}^{n} (A(x_k - x_0) + B\, S_k - (y_k - y_0))^2$$

where $A = -\mu\alpha$ and $B = \alpha$. This is a linear regression problem whose solution is:

$$\begin{bmatrix} A \\ B \end{bmatrix} = M^{-1} \cdot N \tag{3}$$

where:

$$M = \begin{bmatrix} \sum_{k=0}^{n} (x_k - x_0)^2 & \sum_{k=0}^{n} (x_k - x_0) S_k \\ \sum_{k=0}^{n} (x_k - x_0) S_k & \sum_{k=0}^{n} S_k^2 \end{bmatrix}$$

and

$$N = \begin{bmatrix} \sum_{k=0}^{n} (y_k - y_0)(x_k - x_0) \\ \sum_{k=0}^{n} (y_k - y_0) S_k \end{bmatrix}$$

By calculating B from formula (3), we may find a good approximation of the main parameter of our exponential: $\alpha = B$.

To find μ and β we go back to equation (1) and we use squared error minimization again, by minimizing:

$$\sum_{k=0}^{n} (\mu + \beta\, e^{\alpha x_k} - y_k)^2$$

The solution of such linear regression problem is:

$$\begin{bmatrix} \mu \\ \beta \end{bmatrix} = \begin{bmatrix} n & \sum\limits_{k=0}^{n} e^{\alpha x_k} \\ \sum\limits_{k=0}^{n} e^{\alpha x_k} & \sum\limits_{k=0}^{n} (e^{\alpha x_k})^2 \end{bmatrix}^{-1} \begin{bmatrix} \sum\limits_{k=0}^{n} y_k \\ \sum\limits_{k=0}^{n} y_k e^{\alpha x_k} \end{bmatrix} \tag{4}$$

In summary, our approximation algorithm computes, for each histogram, the elements of matrices M and N above to determine $\alpha = B$ using equation (3), then it computes μ and β using equation (4). A similar approach is used to approximate the double exponential decay given by equation (2). In this case, we use the formulas in Formulas 1.

$$\begin{bmatrix} A \\ B \\ C \\ D \\ E \end{bmatrix} = \begin{bmatrix} \sum\limits_{k=0}^{n} SS_k^2 & \sum\limits_{k=0}^{n} SS_k S_k & \sum\limits_{k=0}^{n} SS_k x_k^2 & \sum\limits_{k=0}^{n} SS_k x_k & \sum\limits_{k=0}^{n} SS_k \\ \sum\limits_{k=0}^{n} SS_k S_k & \sum\limits_{k=0}^{n} S_k^2 & \sum\limits_{k=0}^{n} S_k x_k^2 & \sum\limits_{k=0}^{n} S_k x_k & \sum\limits_{k=0}^{n} S_k \\ \sum\limits_{k=0}^{n} SS_k x_k^2 & \sum\limits_{k=0}^{n} S_k x_k^2 & \sum\limits_{k=0}^{n} x_k^4 & \sum\limits_{k=0}^{n} x_k^3 & \sum\limits_{k=0}^{n} x_k^2 \\ \sum\limits_{k=0}^{n} SS_k x_k & \sum\limits_{k=0}^{n} S_k x_k & \sum\limits_{k=0}^{n} x_k^3 & \sum\limits_{k=0}^{n} x_k^2 & \sum\limits_{k=0}^{n} x_k \\ \sum\limits_{k=0}^{n} SS_k & \sum\limits_{k=0}^{n} S_k & \sum\limits_{k=0}^{n} x_k^2 & \sum\limits_{k=0}^{n} x_k & n \end{bmatrix}^{-1} \cdot \begin{bmatrix} \sum\limits_{k=0}^{n} SS_k y_k \\ \sum\limits_{k=0}^{n} S_k y_k \\ \sum\limits_{k=0}^{n} x_k^2 y_k \\ \sum\limits_{k=0}^{n} x_k y_k \\ \sum\limits_{k=0}^{n} y_k \end{bmatrix}$$

where:

$$\begin{cases} S_0 = 0 \quad \text{for } k = 0 \\ S_k = S_{k-1} + \frac{1}{2}(y_k + y_{k-1})(x_k - x_{k-1}) \quad \text{for } k = 1..n \\ SS_0 = 0 \quad \text{for } k = 0 \\ SS_k = SS_{k-1} + \frac{1}{2}(S_k + S_{k-1})(x_k - x_{k-1}) \quad \text{for } k = 1..n \end{cases}$$

allows to calculate:

$$\begin{cases} \alpha_1 = \frac{1}{2}(B + \sqrt{B^2 + 4A} \\ \alpha_2 = \frac{1}{2}(B + \sqrt{B^2 - 4A} \end{cases}$$

while μ, β_1, and β_2 are obtained from:

$$\begin{bmatrix} \mu \\ \beta_1 \\ \beta_2 \end{bmatrix} = \begin{bmatrix} n & \sum\limits_{k=0}^{n} e^{\lambda_1 x_k} & \sum\limits_{k=0}^{n} e^{\lambda_2 x_k} \\ \sum\limits_{k=0}^{n} e^{\lambda_1 x_k} & \sum\limits_{k=0}^{n} (e^{\lambda_1 x_k})^2 & \sum\limits_{k=0}^{n} e^{\lambda_1 x_k} e^{\lambda_2 x_k} \\ \sum\limits_{k=0}^{n} e^{\lambda_2 x_k} & \sum\limits_{k=0}^{n} e^{\lambda_1 x_k} e^{\lambda_2 x_k} & \sum\limits_{k=0}^{n} (e^{\lambda_2 x_k})^2 \end{bmatrix}^{-1} \cdot \begin{bmatrix} \sum\limits_{k=0}^{n} y_k \\ \sum\limits_{k=0}^{n} e^{\lambda_1 x_k} y_k \\ \sum\limits_{k=0}^{n} e^{\lambda_2 x_k} y_k \end{bmatrix}$$

Formulas 1: Formulas to solve the double exponential regression problem

Both in the case of a single exponential decay and in the case of a double exponential decay, we execute the algorithm in parallel on the GPU using one thread for each histogram (out of $32 \times 32 \times 8 = 8192$ histograms). Since the computation of each histogram is independent with respect to the others, differently from the counting phase, in this case there

is no need for synchronization among threads. Moreover, all threads execute the very same sequence of instructions: they access different data elements (histograms) but their control flow does not differ. As mentioned in Section III this maximizes GPU utilization.

To further optimize processing, histograms filled by the counting kernels are stored in GPU's global memory as a huge, single array, which first contains all the elements in position 0 for all the histograms, then all the elements in position 1, and so on. In this way, when the lifetime extraction kernel accesses element i in the histograms, all threads that run such kernel access contiguous memory regions, and the GPU can fully exploit its wide access memory bandwith to load data from global memory, while all intermediate results required to compute formulas above fit in the fast local memory of each thread.

While the use of GPUs to speed up FLIM has been proposed in recent work[27], the approach on a different analog mean-delay method that could not capture double-exponential decaying, as we do.

V. EVALUATION

To evaluate the effectiveness of our solution we have to show that it is capable of performing the lifetime extraction task at the expected *speed* (10 Gbit/s) with a *precision* comparable with that of state-of-the-art tools. These two aspects are detailed in the sub-sections below.

The PC we use in our evaluation is an off-the-shelf machine equipped with an Intel i7-8700 CPU (6 cores, 12 threads) and 64 GB of DDR4 RAM. The CUDA card we use is an NVIDIA Titan X GPU with 12 GB of on-board GDDR5X RAM.

A. Speed

As a first test we connected the processing PC with the networking module of the camera (i.e., the FPGA that implements the UDP communication protocol and the transceiver that implements the physical protocol, as in Fig. 1), using an Intel 82599ES 10 Gbit/s Ethernet card on the processing PC and an optical fiber link as a cable. On the camera-side we generate samples at the maximum rate allowed by the network card and we measured if the PC was able to process such data rate. We also repeat this experiment using a second PC in place of the camera. Even in this case the second PC generates samples at the maximum

decays	**readings**	**throughput**
single	1000	10.60 Gbit/s
	2000	13.92 Gbit/s
	3000	15.01 Gbit/s
	4000	15.58 Gbit/s
	5000	15.96 Gbit/s
double	5000	10.38 Gbit/s
	10000	12.90 Gbit/s
	20000	14.65 Gbit/s
	30000	15.42 Gbit/s
	40000	15.71 Gbit/s
	50000	15.86 Gbit/s

TABLE I: Sustainable throughput.

rate allowed by the network card and sends them to the processing PC, using the same UDP based protocol implemented by the camera (see Sections II and III). In both cases we verified that the processing PC was able to effectively process 10 Gbit/s of data, with the Ethernet link being the bottleneck of the whole system.

Afterwards, in order to measure the raw performance of our processing sub-system, removing the potential bottleneck of the network link, we repeated our measures, this time running both the data generator and the data processing application on the same machine, communicating through UDP datagram as in the previous case, but using the loopback interface as the underlying networking link. Table I shows the results we measured.

As discussed in Section IV, our algorithm consists of two phases: counting and lifetime extractions, with the second being more computationally expensive than the first. As a consequence, the more readings we collect and count before launching a lifetime extraction, the more data we are able to process per second, as confirmed by the results of our experiments.

This reflects in Table I, where we notice that for a single exponential decay we reach and pass our target of 10 Gbit/s by processing 1000 readings per detector. For a 32×32 camera this means collecting ≈ 1 M readings. Since each reading is encoded in 3 bytes, this means processing more than 430 "camera frames" per second.

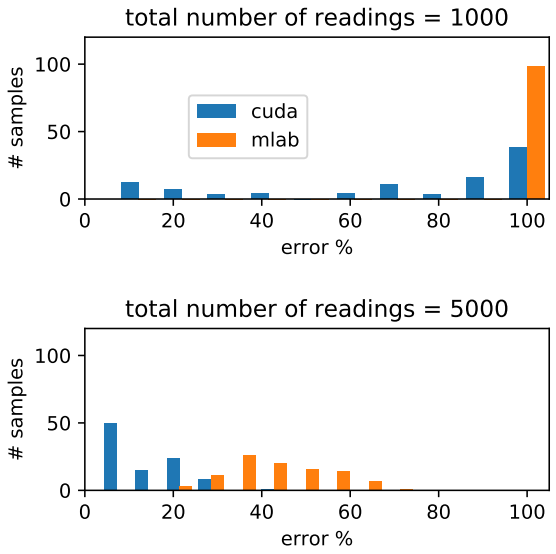

FIG. 2: Comparison with real life data

The case of double exponential decays is more complex and we need to process 5000 readings per detector to reach our target. Next section shows that this is also the lowest possible number of readings if we want to approximate our double decay with a sufficient precision. By repeating the calculations above, we observe that taking a snapshot every 5000 readings means processing more than 86 "camera frames" per second.

B. Precision

To evaluate the precision of our algorithm, we compare it against the off-line regression algorithms provided by the MATLAB *Curve Fitting Toolbox*[28]. It represents a ready-to-use, state-of-the-art solution when speed of calculation is not fundamental. Indeed, using an iterative algorithm, MATLAB continues processing until the required precision is reached.

This comparison is performed both on real life data and on syntetic data. The latter analysis is fundamental to study how the various parameters of the decay phenomena under study affect the precision of our algorithm.

Real life data

Fig. 2 shows the results we measured using FLIM data acquired with a 32x1 module[29] that captures the fluorescence lifetime of Coumarin 6 (details can be found in[30]). In particular, we had data from several experiments (the reading of each of the 32 detectors) made under

parameter	range
num. readings	5000 ÷ 50000
λ_1	0.001 ÷ 0.01
λ_2	0.0001 ÷ 0.0009
% of noise readings	0.1% ÷ 1%
β_1/β_2	0.25 ÷ 4

TABLE II: Parameters.

different conditions (different laser data rate). For each experiment we measure the error made by the two algorithms in calculating the rateo of the decay and we plot the distribution of such error in Fig. 2. It shows how the error performed by our CUDA algorithm is lower (more precisely, it has a better distribution) than the error made by MATLAB and this is true independently from the number of readings we considered (1000 or 5000). Fig. 2 also shows how an increased number of readings is beneficial both to our algorithm and to MATLAB.

Synthetic data

To precisely evaluate how the various parameters of the decay affect the precision of our algorithm, we generate random readings that follow a given decay curve and compare the two algorithms in terms of their ability in capturing the parameters of the decay. In particular, we focus on double exponential decay as this is the hardest case to fit and it is affected by many more parameters. Below we rewrite the formula that describes such decay (equation 2 in Section IV) using $\lambda_i = -\alpha_i$ to put in evidence the fact that we are studying "decays", i.e., negative exponentials:

$$y = \mu + \beta_1 \, e^{-\lambda_1 x} + \beta_2 \, e^{-\lambda_2 x}$$

Table II lists the main parameters we considered in our comparison, with the typical ranges that we encounter in biological analysis. For each case we generate 50 sets of data, using different seeds for the random generator, and we calculate the average and 90% confidence interval of the error made by MATLAB and by our fitting algorithm in capturing the parameters of the decay: λ_1, λ_2, and the *ratio* β_1/β_2.

Fig. 3 shows the results we measured with the smallest number of readings we consider

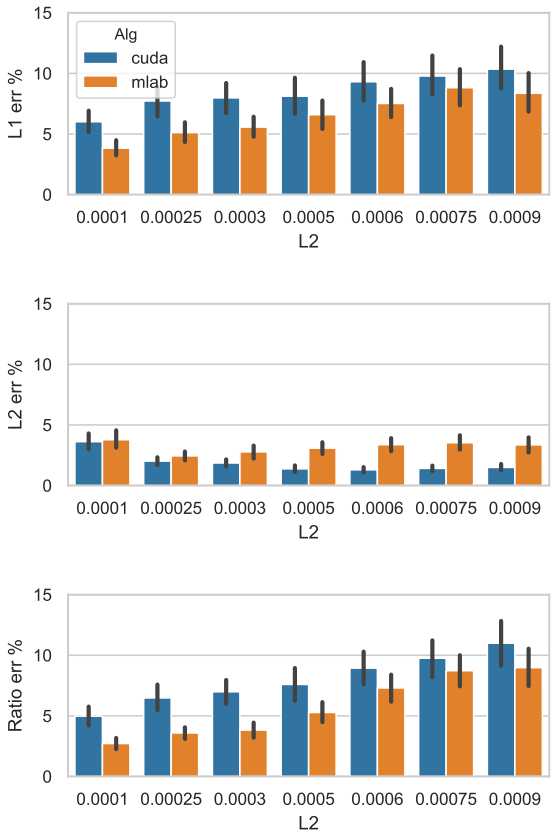

FIG. 3: Comparison with 5000 readings, $\lambda_1 = 0.01$, $ratio = 0.25$, $noise = 0.1\%$

(5000), when varying λ_2, with $\lambda_1 = 0.01$, $ratio = 0.25$, and $noise = 0.1\%$. We may appreciate that the error of our CUDA algorithm is low in absolute terms (in most cases less than 5%) and it is comparable if not lower than the error of MATLAB, which is a very good result if we consider that the latter requires an average of 100 ms to fit each curve (that is, each detector) while our algorithm can fit all $32 \times 32 = 1024$ detectors in 11 ms.

A closer look to Fig. 3 shows that the largest error we make (close to 8%) is on estimating λ_1. This is expected, since a ratio of $\beta_1/\beta_2 = 0.25$ means we have less readings for λ_1 than for λ_2. Indeed, the results improve when we increase the absolute number of readings – Fig. 4(a) – or the ratio – Fig. 4(b).

For space reasons we omit the results we got when varying the ratio and the noise: they are very similar to those presented so far.

Fig. 5(a) shows the precision when λ_1 changes. The error in estimating our parameters (especially the ratio) grows considerably when λ_1 and λ_2 get close. However, the iterative MATLAB algorithm suffers similarly if not more than our algorithm in these conditions,

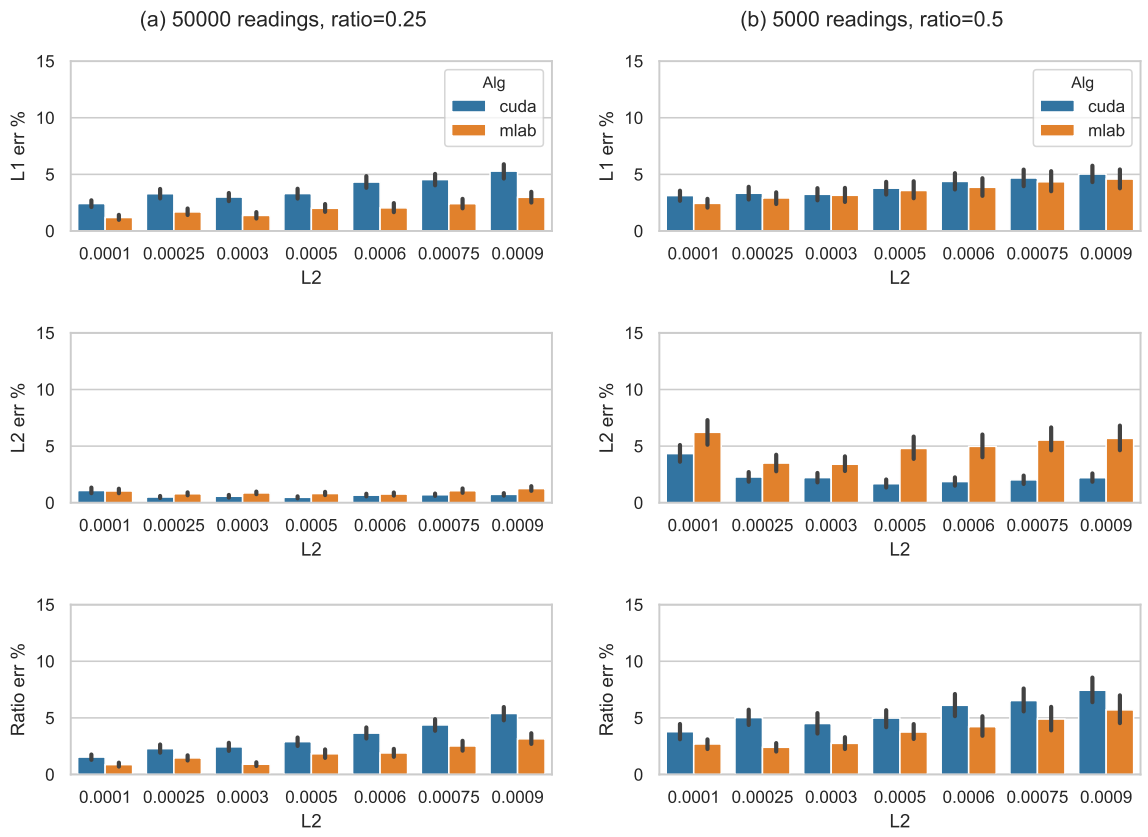

FIG. 4: Comparison with $\lambda_1 = 0.01$, $noise = 0.1\%$.

which suggests that the problem is complex per se. Fig. 5(b) shows that the situation gets better when the number of readings is increased to reach 50 K, but it is still critical for very close values of λ_1 and λ_2.

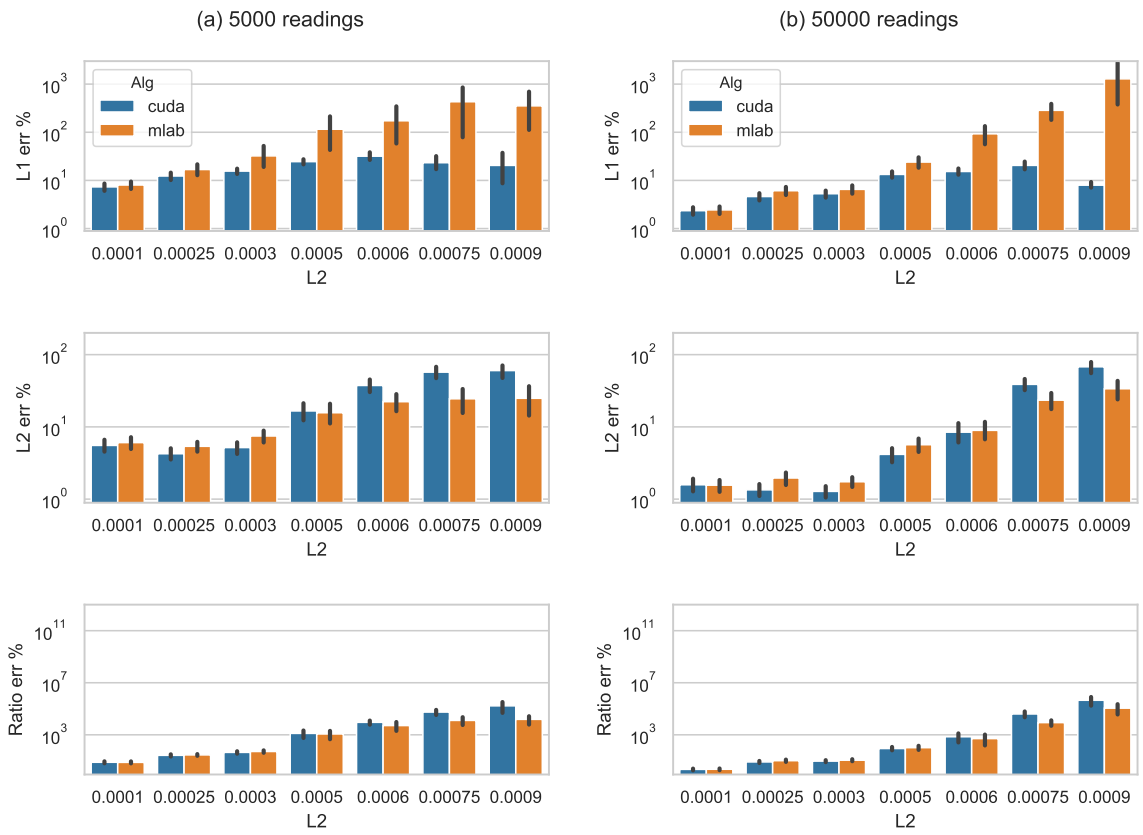

FIG. 5: Comparison with $\lambda_1 = 0.001$, $ratio = 0.25$, $noise = 0.1\%$

VI. CONCLUSIONS

In this paper, we presented a GPU-based algorithm that performs accurate lifetime extraction from TCSPC data. The algorithm enables video-rate processing of a TCSPC datastream up to a rate of 10 Gbit/s, potentially generated by a 32×32 SPAD camera. We study both the processing performance and the precision of the algorithm, and demonstrate that it can reach comparable or better precision with respect to state-of-the-art iterative algorithms at a fraction of their execution time.

REFERENCES

[1]W. Becker, "Fluorescence lifetime imaging–techniques and applications," Journal of microscopy **247**, 119–136 (2012).

[2]W. Becker, *Advanced time-correlated single photon counting techniques*, Vol. 81 (Springer Science & Business Media, 2005).

[3]K. Suhling, P. M. French, and D. Phillips, "Time-resolved fluorescence microscopy," Photochemical & Photobiological Sciences **4**, 13–22 (2005).

[4]H. Gerritsen, M. Asselbergs, A. Agronskaia, and W. Van Sark, "Fluorescence lifetime imaging in scanning microscopes: acquisition speed, photon economy and lifetime resolution," Journal of microscopy **206**, 218–224 (2002).

[5]F. Gasparri, "An overview of cell phenotypes in hcs: limitations and advantages," Expert opinion on drug discovery **4**, 643–657 (2009).

[6]P. Kapusta, M. Wahl, A. Benda, M. Hof, and J. Enderlein, "Fluorescence lifetime correlation spectroscopy," Journal of fluorescence **17**, 43–48 (2007).

[7]C. Niclass, C. Favi, T. Kluter, M. Gersbach, and E. Charbon, "A 128×128 single-photon image sensor with column-level 10-bit time-to-digital converter array," IEEE Journal of Solid-State Circuits **43**, 2977–2989 (2008).

[8]C. Veerappan, J. Richardson, R. Walker, D.-U. Li, M. W. Fishburn, Y. Maruyama, D. Stoppa, F. Borghetti, M. Gersbach, R. K. Henderson, *et al.*, "A 160×128 single-photon image sensor with on-pixel 55ps 10b time-to-digital converter," in *2011 IEEE International Solid-State Circuits Conference* (IEEE, 2011) pp. 312–314.

[9]R. M. Field, S. Realov, and K. L. Shepard, "A 100 fps, time-correlated single-photon-counting-based fluorescence-lifetime imager in 130 nm cmos," IEEE Journal of Solid-State Circuits **49**, 867–880 (2014).

[10]C. Niclass, K. Ito, M. Soga, H. Matsubara, I. Aoyagi, S. Kato, and M. Kagami, "Design and characterization of a 256x64-pixel single-photon imager in cmos for a mems-based laser scanning time-of-flight sensor," Optics Express **20**, 11863–11881 (2012).

[11]L. Parmesan, N. Dutton, N. J. Calder, N. Krstajic, A. J. Holmes, L. A. Grant, and R. K. Henderson, "A 256× 256 spad array with in-pixel time to amplitude conversion for fluorescence lifetime imaging microscopy," in *International Image Sensor Workshop, Vaals, Netherlands, Memory*, Vol. 900 (2015) p. M5.

[12]N. Krstajić, S. Poland, J. Levitt, R. Walker, A. Erdogan, S. Ameer-Beg, and R. K. Henderson, "0.5 billion events per second time correlated single photon counting using cmos spad arrays," Optics letters **40**, 4305–4308 (2015).

[13]S. Lindner, C. Zhang, I. M. Antolovic, M. Wolf, and E. Charbon, "A 252× 144 spad pixel flash lidar with 1728 dual-clock 48.8 ps tdcs, integrated histogramming and 14.9-to-1 compression in 180nm cmos technology," in *2018 IEEE Symposium on VLSI Circuits*

(IEEE, 2018) pp. 69–70.

[14]D. Tyndall, B. Rae, D. Li, J. Richardson, J. Arlt, and R. Henderson, "A 100mphoton/s time-resolved mini-silicon photomultiplier with on-chip fluorescence lifetime estimation in 0.13 μm cmos imaging technology," in *2012 IEEE International Solid-State Circuits Conference* (IEEE, 2012) pp. 122–124.

[15]J. Tachella, Y. Altmann, N. Mellado, A. McCarthy, R. Tobin, G. S. Buller, J.-Y. Tourneret, and S. McLaughlin, "Real-time 3d reconstruction of complex scenes using single-photon lidar: when image processing meets computer graphics," arXiv preprint arXiv:1905.06700 (2019).

[16]K. Suhling, J. Siegel, D. Phillips, P. M. French, S. Lévêque-Fort, S. E. Webb, and D. M. Davis, "Imaging the environment of green fluorescent protein," Biophysical journal **83**, 3589–3595 (2002).

[17]M. A. Digman, V. R. Caiolfa, M. Zamai, and E. Gratton, "The phasor approach to fluorescence lifetime imaging analysis," Biophysical journal **94**, L14–L16 (2008).

[18]S.-J. Chen, N. Sinsuebphon, A. Rudkouskaya, M. Barroso, X. Intes, and X. Michalet, "In vitro and in vivo phasor analysis of stoichiometry and pharmacokinetics using short-lifetime near-infrared dyes and time-gated imaging," Journal of biophotonics **12**, e201800185 (2019).

[19]CUDA, "CUDA," http://www.nvidia.com/object/cuda_home_new.html (2019), visited April 2019.

[20]G. Acconcia, A. Cominelli, I. Rech, and M. Ghioni, "High-efficiency integrated readout circuit for single photon avalanche diode arrays in fluorescence lifetime imaging," Review of Scientific Instruments **87**, 113110 (2016).

[21]D. Bronzi, F. Villa, S. Tisa, A. Tosi, and F. Zappa, "Spad figures of merit for photon-counting, photon-timing, and imaging applications: A review," IEEE Sensors Journal **16**, 3–12 (2015).

[22]M. Crotti, I. Rech, G. Acconcia, A. Gulinatti, and M. Ghioni, "A 2-ghz bandwidth, integrated transimpedance amplifier for single-photon timing applications," IEEE Transactions on Very Large Scale Integration (VLSI) Systems **23**, 2819–2828 (2015).

[23]F. Ceccarelli, G. Acconcia, A. Gulinatti, M. Ghioni, and I. Rech, "83-ps timing jitter with a red-enhanced spad and a fully integrated front end circuit," IEEE Photonics Technology Letters **30**, 1727–1730 (2018).

[24]G. Acconcia, M. Ghioni, and I. Rech, "37ps-precision time-resolving active quenching circuit for high-performance single photon avalanche diodes," IEEE Photonics Journal **10**, 1–13 (2018).

[25]A. Cominelli, G. Acconcia, P. Peronio, I. Rech, and M. Ghioni, "Readout architectures for high efficiency in time-correlated single photon counting experiments—analysis and review," IEEE Photonics Journal **9**, 1–15 (2017).

[26]J. Jacquelin, "Regressions and integral equations," (2009), https://www.scribd.com/doc/14674814/Regressions-et-equations-integrales.

[27]B. Kim, B. Park, S. Lee, and Y. Won, "Gpu accelerated real-time confocal fluorescence lifetime imaging microscopy (flim) based on the analog mean-delay (amd) method," Biomedical optics express **7**, 5055–5065 (2016).

[28]matfit, "Curve fitting toolbox," https://www.mathworks.com/products/curvefitting.html (2019), visited April 2019.

[29]A. Cuccato, S. Antonioli, M. Crotti, I. Labanca, A. Gulinatti, I. Rech, and M. Ghioni, "Complete and compact 32-channel system for time-correlated single-photon counting measurements," IEEE Photonics Journal **5**, 6801514 (2013).

[30]A. Tsikouras, P. Peronio, I. Rech, N. Hirmiz, M. Deen, and Q. Fang, "Characterization of spad array for multifocal high-content screening applications," Photonics **3**, 56 (2016).