Journal of Biomedical Optics

BiomedicalOptics.SPIEDigitalLibrary.org

GPU acceleration of time-domain fluorescence lifetime imaging

Gang Wu
Thomas Nowotny
Yu Chen
David Day-Uei Li

SPIE.

GPU acceleration of time-domain fluorescence lifetime imaging

Gang Wu,[a,b] Thomas Nowotny,[b] Yu Chen,[c] and David Day-Uei Li[a,b,*]
[a]University of Strathclyde, Strathclyde Institute of Pharmacy and Biomedical Sciences, Centre for Biophotonics, Glasgow G4 ORE, United Kingdom
[b]University of Sussex, School of Engineering and Informatics, Brighton BN1 9QJ, United Kingdom
[c]University of Strathclyde, Department of Physics, Glasgow G1 1XJ, United Kingdom

Abstract. Fluorescence lifetime imaging microscopy (FLIM) plays a significant role in biological sciences, chemistry, and medical research. We propose a graphic processing unit (GPU) based FLIM analysis tool suitable for high-speed, flexible time-domain FLIM applications. With a large number of parallel processors, GPUs can significantly speed up lifetime calculations compared to CPU-OpenMP (parallel computing with multiple CPU cores) based analysis. We demonstrate how to implement and optimize FLIM algorithms on GPUs for both iterative and noniterative FLIM analysis algorithms. The implemented algorithms have been tested on both synthesized and experimental FLIM data. The results show that at the same precision, the GPU analysis can be up to 24-fold faster than its CPU-OpenMP counterpart. This means that even for high-precision but time-consuming iterative FLIM algorithms, GPUs enable fast or even real-time analysis. [DOI: 10.1117/1.JBO.21.1.017001]

Keywords: fluorescence lifetime imaging microscopy; graphic processing units; parallel processing; noniterative algorithms; iterative algorithms.

Paper 150608R received Sep. 10, 2015; accepted for publication Nov. 19, 2015; published online Jan. 6, 2016.

1 Introduction

Fluorescence lifetime imaging microscopy (FLIM) is a powerful imaging technique that can not only locate fluorescent molecules (fluorophores) but also provide images based on the temporal decay rate of the fluorescence emitted from fluorophores. In contrast to traditional intensity imaging, FLIM is insensitive to the intensities of light sources and probe concentrations.[1] As lifetime characteristics of fluorophores are usually sensitive to their environment, FLIM can image physiological parameters such as pH, O_2, Ca^{2+}, Cl^-, or temperature linked to diseases or potential therapies. For example, FLIM has been combined with Förster resonance energy transfer (FRET) techniques (FLIM-FRET) for assessing drug efficacy and clearance in tumors,[2] cellular temperature in cancerous tissues under treatments,[3] and cerebral energy metabolism for understanding brain functions.[4]

In order to monitor fast dynamic biological events, it is necessary to build a high-speed or even real-time FLIM system.[5–7] FLIM can operate in the time domain or frequency domain.[8,9] Wide-field frequency-domain FLIM instruments are capable of capturing phase images at a high frame rate, but the acquisition will decrease if more than 10 phase images are required (which usually happens in realistic laboratory scenarios), and accurate analysis heavily relies on appropriate software.[10] Moreover, if there is more than one decay component in a fluorescence histogram, excitations at multiple frequencies[11] or at a single frequency with a sufficiently narrow pulse (using higher-order harmonic frequencies) are required,[12] greatly reducing the acquisition rate and increasing the system complexity. Time-domain techniques, on the other hand, include gated camera[13] and time-correlated single-photon counting (TCSPC) methods. Due to superior temporal resolution and recent developments in advanced laser sources and cameras, TCSPC techniques have been the gold standard solutions for FLIM experiments.[14–16] Traditional photomultiplier tube (PMT) based TCSPC instruments are capable of single-photon detection, and they can have multiple channels to boost the acquisition. The latest multichannel TCSPC systems[16–19] can acquire image raw data in seconds, but the FLIM images are mainly processed by slow analysis software. Recent advances in semiconductor technologies have allowed single-photon detectors to be fabricated in two-dimensional (2-D) arrays in a low-cost silicon process with integrated on-chip TCSPC modules.[20–22] These highly parallel single-photon avalanche diode (SPAD) arrays can be configured for wide-field or multifocal multiphoton microscopy systems to enhance the acquisition rate.[16] Faster acquisition means more image raw data are generated in a shorter period of time. Similar to the multi-PMT TCSPC system, 2-D SPAD+TCSPC arrays in a single chip significantly enhance the parallelism, enabling real-time acquisition.[21,23] And the data throughput of the latest 2-D SPAD arrays can easily exceed tens of gigabit/s, imposing significant challenges in data handling and image analysis.[21] In such systems, lifetime analysis has become a bottleneck for real-time FLIM applications.

To address this problem, several noniterative fast FLIM algorithms have been developed, including rapid lifetime determination (RLD),[24,25] the integral equation method (IEM),[26] the center of mass method (CMM),[23,27] the phasor method (PM),[28,29] moment methods,[30] and fast bi-exponential solvers, including bi-exponential centre of mass method (BCMM)[31] and the

*Address all correspondence to: David Day-Uei Li, E-mail: David.Li@strath.ac.uk

four-gate method.[32] To boost image generation, high-performance hardware can be employed to process the imaging data. For example, Li et al. have previously implemented field programmable gate array (FPGA) embedded FLIM processors capable of offering video-rate FLIM imaging.[23] These hardware embedded FLIM processors, however, only generate single-exponential FLIM images and are less flexible than traditional iterative algorithms, such as maximum likelihood estimation (MLE),[33] the least square method (LSM),[33,34] and global analysis (GA).[35–37] Compared with noniterative methods, iterative algorithms have the following advantages: (1) they have a wider working range, so that they can be employed in most situations; (2) they can support bi- or multiexponential analysis; and (3) they have better precision and accuracy performances. But iterative algorithms are generally slow and cannot be sped-up easily, and are also difficult to realize in hardware.

In this paper, we will focus on how the latest graphics processing units (GPUs) can be used to increase the lifetime estimation speed for both noniterative and iterative algorithms. The work developed in this paper can be a fast FLIM processor between the SPAD-array or multi-PMT acquisition front-ends and the GPU shown in Fig. 1. The realization of parallel FLIM analysis and its optimization will be given in Sec. 2, including a brief introduction of the algorithms. In Sec. 3, we will compare the GPU analysis with CPU parallel analysis and verify the new GPU FLIM analysis methods on both synthesized data and experimental data.

2 Methodology

To generate a lifetime image requires a FLIM analysis algorithm to process a large number of pixels with each pixel containing a decay histogram. Instead of calculating lifetimes in a pixel-by-pixel manner, it is desirable to analyze all histograms in parallel for real-time applications. Although FPGA technologies have already been introduced in this emerging area,[38] their usage is still limited since only the hardware-friendly algorithms, such as RLD, IEM, and CMM,[23–27] can be implemented. GPUs, on the other hand, contain a massive number of cores, offering significant computing power, and allow more complex algorithms for data analysis. Therefore, GPUs show great potential in real-time FLIM applications.

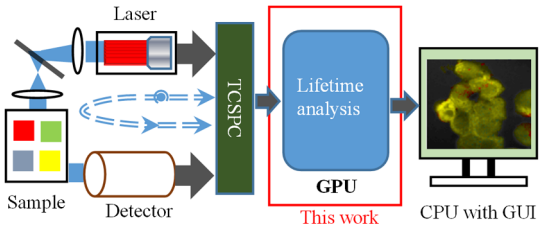

Fig. 1 Focus of this paper in assumed FLIM system.

2.1 *Fluorescence Lifetime Imaging Microscopy Algorithms*

Assume that the measured densities are $f(t) = K \cdot \exp(-t/\tau)$ and $f(t) = K \cdot [f_D \cdot \exp(-t/\tau_F) + (1 - f_D) \cdot \exp(-t/\tau_D)]$ for single- and bi-exponential decays, respectively. τ, τ_F, and τ_D are the lifetimes, K is the prescalar, and f_D is the proportion, and the instrumental response function and background noise are neglected as in the model demonstrated by Leray et al.[29] These assumptions allow a proper comparison among different algorithms. Many FLIM algorithms (both noniterative and iterative methods) have been developed for the past few decades. This paper examines some single-exponential and bi-exponential analysis algorithms, including IEM, CMM, LSM, PM, BCMM, and GA, to demonstrate how GPUs can speed up FLIM analysis. The fundamental function for each method can be found in Table 1 for noniterative algorithms and Table 2 for iterative algorithms:

$$u = \int_0^T f(t) \times \cos(wt)\mathrm{d}t \Big/ \int_0^T f(t)\mathrm{d}t,$$

$$v = \int_0^T f(t) \times \sin(wt)\mathrm{d}t \Big/ \int_0^T f(t)\mathrm{d}t, f(t) = \sum_{i=1}^{n}(A_i e^{-t/\tau_i})$$

$$f_D = [\tau_D(1+\tau_F^2\omega^2)(1-u-u\tau_D^2\omega^2)]/[(\tau_D - \tau_F) \times (1 - u - u\tau_F^2\omega^2 - \tau_F\tau_D\omega^2 - u\tau_D^2\omega^2 - u\tau_F^2\tau_D^2\omega^4)]$$

$$N = \sum_{j=0}^{M-1}(C_j N_j), \qquad X = \sum_{j=0}^{M-1}(C_j t_j N_j), \qquad K = N_0/h,$$

$$G = \frac{KY - NX}{KX - N^2}, \qquad Y = \sum_{j=0}^{M-1}\left(C_j \frac{t_j^2}{2} N_j\right).$$

2.2 *Graphic Processing Unit*

GPUs, originally used for real-time rendering, have been used increasingly in general purpose computing (for example, in many biomedical or clinical applications where image analysis is expected to be in real time[39–44]), with the advent of the Common Unified Device Architecture (CUDA)[45] and general purpose GPUs (e.g., NVIDIA Tesla).[46] Due to their highly parallel architecture (the NVIDIA Tesla K40 used in this work has 2880 parallel cores), high computational density, and memory bandwidth (250 + GB/s), GPUs can accelerate the computing

Table 1 Calculation function of fluorescence lifetime imaging microscopy (FLIM) noniterative algorithms.

IEM[26]	CMM[23,27]	PM[28,29]	$BCMM_1$[31]	$BCMM_2$ (τ_D unknown)[31]
$\tau_{IEM} = \frac{h\sum_{j=0}^{M-1}(C_j N_j)}{N_0 - N_{M-1}}$	$\tau_{CMM} = (\frac{\sum_{j=0}^{M-1}(jN_j)}{N_c} + \frac{1}{2})h$	$\tau_F = \frac{1-u-v\tau_D\omega}{\omega(v-u\tau_D\omega)}$	$\tau_F = \frac{\tau_D N - X}{\tau_D K - N}$	$\tau_F = 0.5[G - \sqrt{G^2 - 4(NG - X)/K}]$
		$\tau_{Avg} = f_D\tau_F + (1 - f_D)\tau_D$	$\tau_{Avg} = \frac{Nh}{N_0}$	$\tau_{Avg} = \frac{Nh}{N_0}$

Note: IEM, integral equation method; CMM, center of mass method; PM, phasor method; BCMM, bi-exponential centre of mass method; M is the number of time bins; N_j and t_j are the photon number and the delay time of the jth bin, respectively; N_0 is the count number of the first time bin; h is the width of the time bin; C_j is the coefficient of Simpson's rule; and τ_D is the lifetime of the donor.[29]

Table 2 Calculation function of FLIM iterative algorithms.

Algorithm	Function	
LSM[33,34]	$\chi^2 = \sum_{j=0}^{M-1} (\frac{N_j - Y_j}{\sigma_j})^2$	$Y_j = \sum_{k=1}^{n} (A_k e^{-t_j/\tau_k})$
GA[35–37]	$\chi^2 = \sum_{i=1}^{N_{GA}} \sum_{j=0}^{M-1} (\frac{N_{i,j} - Y_j}{\sigma_{i,j}})^2$	

Note: LSM, least square method; GA, global analysis; n is the number of lifetime components; $N_{i,j}$ is the photon number of the jth bin of the ith pixel; and N_{GA} is the number of pixels in the same segment for GA.

speed dramatically. The latest GPU can reach over 5500 Gflops (floating point operations per second).[47]

2.3 *Graphic Processing Unit Implementation*

All the algorithms introduced above can be translated into GPU programs using the CUDA application programming interfaces.[46–48] In this section, we describe how to implement noniterative and iterative algorithms on GPUs. From this section, one can also understand why a GPU works faster than a CPU.

We assume the FLIM image to be analyzed contains 512×512 pixels, and the histogram in each pixel has 256 time bins. In GPU programming, the program is launched as grids of blocks of threads (the left-hand charts of Figs. 2 and 3 show the relationships among thread, block, and grid).[47] And instead of being executed only once or a few times in a regular C program, it is executed N times in parallel by N different CUDA threads.[47]

2.3.1 *Thread-based computing for noniterative algorithms*

For noniterative algorithms, GPU and CPU programs share similar code, except for the specific configurations of hardware resources and memory accesses in GPU implementations. The histogram of each pixel is analyzed by an independent CUDA thread, as shown in Fig. 2, and each block contains 512 threads. This configuration allows analyzing a large number of pixels simultaneously, the exact number being determined by the number of streaming multiprocessors (SMPs). In this case, up to 30,720 pixels can be launched on the NVIDIA Tesla K40 GPU.[47] Then, following the single instruction multiple threads mechanics,[49] the lifetime estimations can be realized in parallel, instead of calculating lifetimes in a serial pixel-by-pixel manner as on a CPU.

2.3.2 *Block-based computing for iterative algorithms*

For iterative algorithms, the histogram for each pixel is analyzed by a separate CUDA block, as shown in Fig. 3 (here, each thread processes a single time bin, instead of an entire pixel as in noniterative algorithms), and each such block contains 256 threads that correspond to the 256 time bins. This configuration still also allows a quite large number of pixels to be analyzed simultaneously, and 120 pixels are expected on the NVIDIA Tesla K40. Here, we take LSM as an example to demonstrate how to implement iterative algorithms on GPUs. The CUDA operations can be divided into the following two categories.

Thread-independent operation. For those operations that are independent in each time bin, the CUDA provides a basic instruction. For example, subtraction or division for each time bin $(N_j - Y_j)/\sigma_j$ can be finished in only one instruction cycle of one instruction as shown in Fig. 4(a) (here, we just take subtraction as an example, since division follows the same procedure), whereas there are 64 subtraction cycles in the CPU-OpenMP operation [Fig. 4(b)]. In Fig. 4(a), N_j and Y_j are stored in the variables SubA and SubB, respectively, and every thread has these two variables, whereas the CPU defines two arrays (SubA[256] and SubB[256]) for loop operations.

Thread-exchanged operation. Alternatively, the CUDA provides a special solution for some operations where communication or data sharing with other time bins (threads) is necessary. As shown in Fig. 5(a),[50,51] the data $(N_j - Y_j)^2/\sigma_j^2$ of each time bin is stored in the jth element of a shared memory array SumA (shared memory is on-chip high-speed memory, which enables the threads in the same block to access the same memory content; arrays are defined by: __shared__ SumA[256],

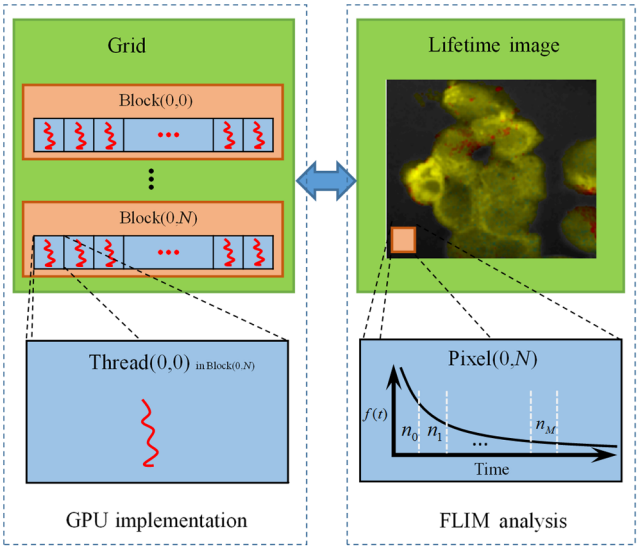

Fig. 2 Relationships among thread, block, and grid, and GPU implementation of noniterative algorithms for FLIM analysis.

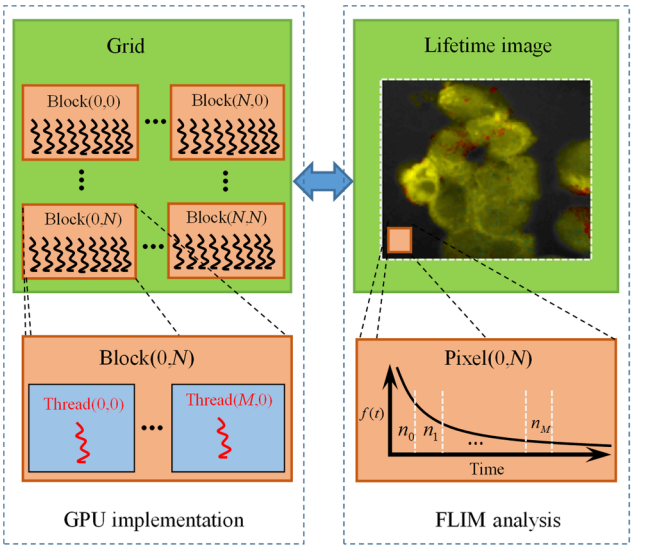

Fig. 3 Relationships among thread, block, and grid, and GPU implementation of iterative algorithms for FLIM analysis.

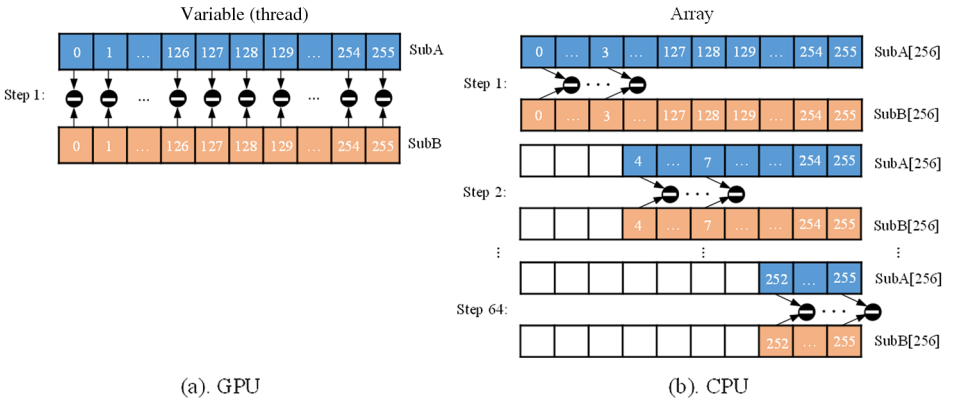

Fig. 4 Subtraction in GPU and CPU-OpenMP processing.

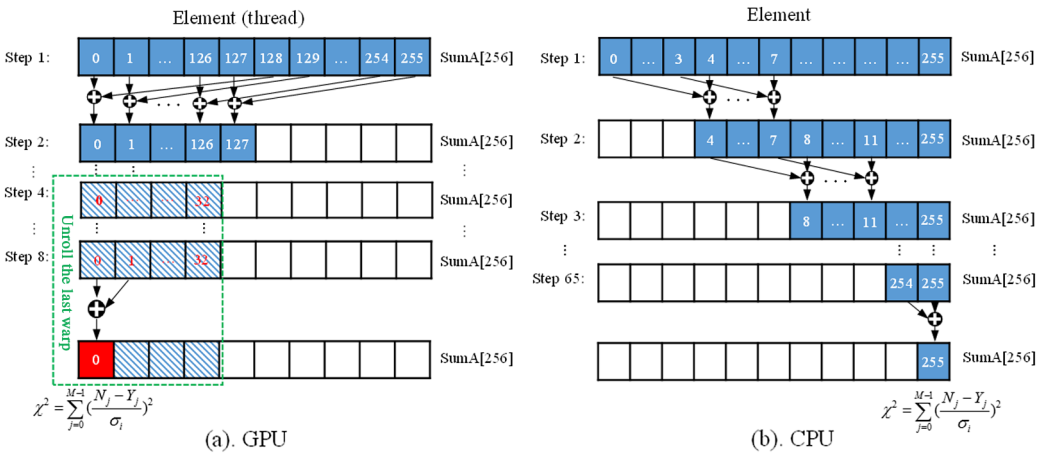

Fig. 5 Summation reduction in GPU and CPU processing.

in CUDA programming[47]). In step 1, the first $256/2 = 128$ threads compress the array SumA into 128 elements by combining the jth and $(j + 128)$th units ($j = 1, \ldots, 128$) simultaneously, followed by several similar steps. Finally, after only $\log_2 256 = 8$ steps (it is not a requirement that the number of threads in a block has to be a power of 2, but it maximizes performance), the summation reduction result can be acquired, instead of after 65 steps in the CPU-OpenMP version as shown in Fig. 5(b).

2.4 *Optimization of Graphic Processing Unit Programming*

In order to maximize the ability of the GPU in FLIM analysis, it is necessary to consider the specifications of GPU hardware, manage the memory properly, and design a specific programming strategy for each individual algorithm. The following considerations are mainly for iterative algorithms, except the kernel configuration and overlap of data transfer, which are also applicable to noniterative algorithms.

2.4.1 *Kernel configuration*

When we launch the kernel in a host (CPU), which is the entrance to GPU processing, similar to the main function in the C language, a proper size of block (determining how many threads in a block) and grid (determining the number of blocks) should be given in order to optimize the GPU performance. On CUDA GPUs, 32 threads are bound together into a so-called warp, which is executed in lockstep. Accordingly, it is suggested that the number of time bins should be a multiple of 32, for instance, 256. Furthermore, according to the GPU hardware specification, at most 2048 threads or 16 blocks can be launched in an SMP, which is the workhorse of the GPU[46] and puts a different constraint on the number of threads. Table 3 illustrates that for thread numbers that are not multiples of 32, although fewer threads are launched, the processing time can be longer. This is due to the so-called warp divergence, which occurs if threads in the same warp follow different conditional branches in the code. In CUDA, diverging threads in a warp are executed serially, and branch divergences therefore lead to considerably slower execution. Additionally, when the thread number is 224, although it is a multiple of 32, its operation time is longer than expected: $1662.5 > (224/256) \times 1870.2 = 1636.4$ ms. This is because for each SM, only $2016 (< 2048)$ threads have been launched, so that not all the hardware resources are being used.

In this case, considering a 512×512 FLIM image to be generated, we examine the performance of GPU FLIM with a different grid size. As shown in Table 4, for FLIM analysis, the calculation times for different sizes are almost the same, so there is no need to configure the block dimension as long as

Table 3 Operation time under different block size of LSM graphic processing unit (GPU).

Threads per block	Warp divergence	Full launch	Time (ms)	Time/bin (ms)
256	No	Yes	1870.2	7.305
255	Yes	No	1898.9	7.447
253	Yes	No	1912.8	7.560
250	Yes	No	1914.2	7.657
225	Yes	No	1960.4	8.713
224	No	No	1662.5	7.422

Table 4 Operation time under different grid size of LSM-GPU.

First dimension	Second dimension	Time (ms)
512×512	1	1869.4
512×16	32	1867.2
512×2	256	1871.3
512	512	1870.2
32	16×512	1872.6
2	256×512	1870.7

it follows the limitation (maximum grid size of NVIDIA K40: $\langle 2^{31} - 1, 65, 535, 65535 \rangle$[47]).

Although simple algorithms use different configurations (i.e., different size of grid or block), the same features can be found as illustrated above.

2.4.2 *Memory optimization*

In order to achieve higher performance, it is necessary to optimize the memory usage and maximize the memory throughput. For FLIM analysis, three aspects should be considered as follows.

First, for a real-time FLIM process, the GPU has to access the histogram data from the CPU for every image frame, so mapped pinned (M-pinned) memory should be encouraged, which is the host memory and has been page-locked and mapped for direct access by the GPU, as shown in Fig. 6.[46] Also, to minimize data transfers between the host (CPU) and device (GPU), the GPU FLIM analysis only uses device memory once the histogram data have been transferred from the host.

Moreover, when a large amount of data is used, it is important to consider asynchronous and overlapping transfers with computations.[52] CUDA streams (which are sequences of operations that are performed in order on the device) can be used to fulfill this mission. As shown in Fig. 7(a), in sequential mode, the kernel can only be launched after all the data have been transferred to the GPU memory. In contrast, for the concurrent mode, the pixels of an FLIM image are divided into several groups, and each pixel within the same group will be launched

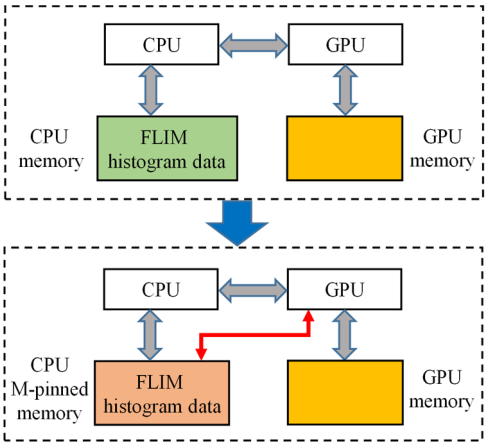

Fig. 6 GPU-FLIM analysis with mapped pinned memory.

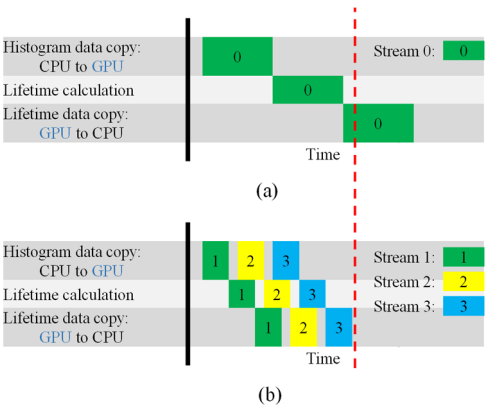

Fig. 7 Timeline comparison of (a) sequential and (b) concurrent executions.

in the same stream. As illustrated in Fig. 7(b), histogram or lifetime data transfer and calculation kernels are separated into different streams. This configuration allows data transfer and kernel execution to run simultaneously, and can reduce the whole processing time measurably.

Last but not least, GPUs have several memory spaces including the shared, global, constant, texture, register, and local memory,[47,52] and for GPU performance, the most important area is memory management.[52] Among these memories, the shared memory is on-chip memory, and it has much higher bandwidth and much lower latency than the local or global memory. It is highly recommended that the shared memory should be used as much as possible, especially when there is communication between threads in the same block (e.g., summation reduction operations). However, shared memory is very limited in size, and it is unavoidable to use the large global memory. Because this is by far the slowest memory accessible from the GPU, care should be taken to access it in the most efficient way. CUDA supports so-called coalesced memory access, where several threads of a warp access a congruent area of device memory in a single memory fetch. Using coalesced access as much as possible is very important to minimize the necessary bandwidth. Figure 8 illustrates the difference between the coalesced access and noncoalesced access, and the ensuing large speed difference.[47,53] To be specific, this implies, for

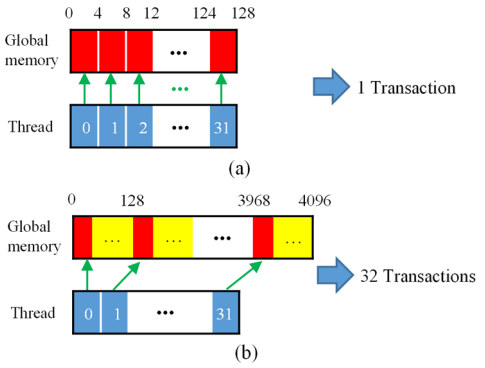

Fig. 8 Global memory access: (a) coalesced access and (b) noncoalesced access.

example, that for noniterative algorithms, the histogram data need to be arranged such that the data of the same time bin but different pixels are directly adjacent, since each thread will access data of the same time bin of the corresponding pixel. For iterative algorithms, however, data of each time bin of a pixel have to be stored in continuous memory locations. Besides, the constant and texture (optimized for 2-D spatial locality) memory spaces, which are cached and read only, can be used to save constant arrays or matrices used in the algorithm and can also speed up the processing. For example, for IEM, every pixel uses the same coefficients C_j, and we can calculate the value before processing, so C_j can be kept in the constant memory (declared by "__constant__ float CoIEM[]").

2.4.3 *Programming strategies*

The first consideration is to trade the precision for the speed when it does not affect the result, such as using single-precision operations instead of double-precision ones. Then the number of divergent warps (threads in the same warp have different operations) that cause delay is minimized. Figure 9 shows a simple example how to avoid divergence. It demonstrates that shifting elements by changing the index can be used to reduce the divergence in some circumstances. More importantly, we try to reduce the number of waiting threads; for example, we execute two summation reductions together in different directions, as shown in Fig. 10, and unroll the last warp in Fig. 5(a).

Moreover, CUDA provides some libraries (e.g., cuBLAS) and GPU versions of math functions, and using these resources can not only boost the performance but also reduce the difficulty of CUDA programming, for example, using "__fdividef()" for floating point division.

3 Results

To demonstrate the advantage of the proposed GPU-FLIM tool, algorithms were tested on both synthesized and experimental data, and the results were compared with CPU solutions. Results were compared in terms of the performances, that is, precision in FLIM analysis, and the time of processing, that

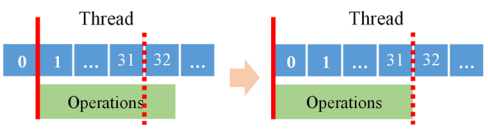

Fig. 9 Example of avoiding warp divergence.

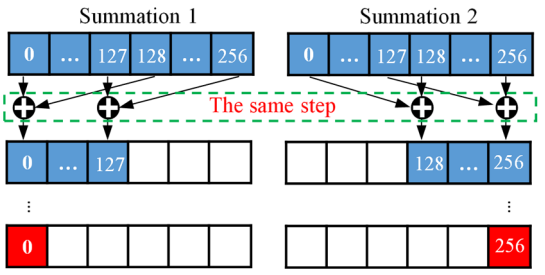

Fig. 10 Example of maximizing hardware usage.

is, their speed. The results are based on an NVIDIA Tesla K40 GPU and an OpenMP CPU implementation on an Intel (R) Xeon(R) E5-2609 v2 processor with four cores. Worth mentioning is that all the results of GPU processing time include the data transfer between CPU and GPU, namely, transferring histogram data to the GPU for processing and moving lifetime results back to the CPU.

3.1 *Simulations on Synthesized Data*

For single-exponential decays, we assume the target FLIM image is a square with bars, where the lifetime gradually increases from left to right and every pixel within the same bar has the same lifetime (2, 2.5, 3, 4 ns), as shown in Fig. 11(a). The image contains 512×512 pixels, <2000 photons have been collected for each pixel, and each histogram has 256 bins with the bin width of 100 ps. Both CPU- and GPU-based algorithms use single-precision operations and have identical outputs, although they have different architectures, namely, latency oriented versus throughput oriented, respectively. We observe that all algorithms generate effective results in agreement with the known ground truth and that the iterative algorithm provides similar or better results, as shown in Figs. 11(b)–11(d).

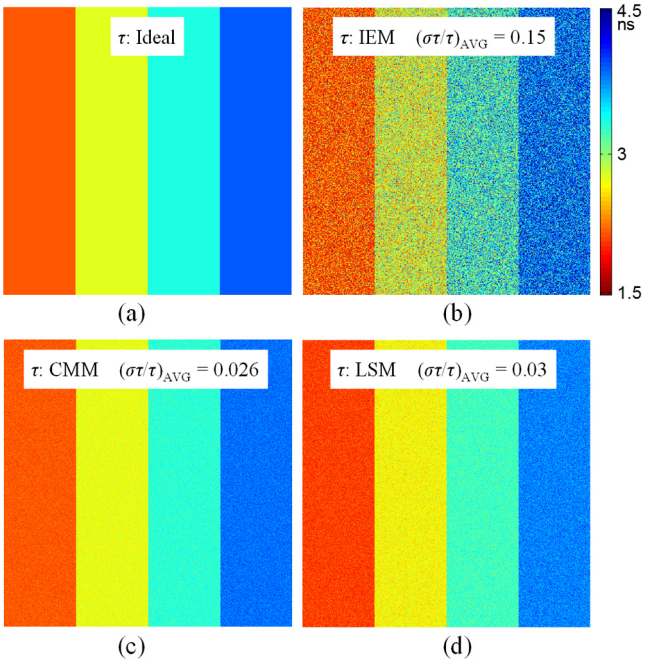

Fig. 11 FLIM images of single-exponential decays: (a) theoretical lifetime image, (b) IEM, (c) CMM, and (d) LSM.

Table 5 Details of bi-exponential simulations.

Image size	Bin number	Bin width (ps)	τ_D (ns)	τ_F (ns)
512×512	256	100	3	1

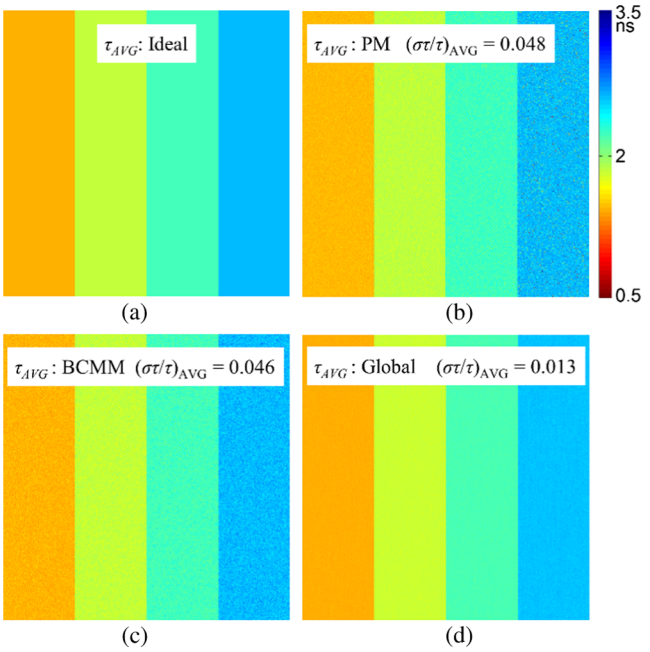

Fig. 12 Images based on synthesized bi-exponential data: (a) theoretical average lifetime image, (b) PM(c) BCMM, and (d) GA-32.

For bi-exponential decays, every pixel has the same lifetime (τ_D, τ_F), whereas f_D decreases from 0.8 to 0.2 from left to right of the image. Table 5 demonstrates the details of this simulation. The results of different algorithms are given in Figs. 12(b)–12(d), where 32 means a segment of GA contains 32 pixels,[35–37] compared with the true image as in Fig. 12(a). From these images, we can learn that all the algorithms generate satisfactory results when $f_D > 0.5$, whereas the GA shows great potential as it can resolve a wider range of f_D.

Tables 6 and 7 show the processing time and speed enhancement of the GPU implementation against the CPU parallel implementation. In Tables 6 and 7, we also included the acquisition rates of recently reported FLIM systems[18,19] for comparison. These systems are all for 256×256 images, but we extended them for 512×512 ones in order to make a proper comparison. For the latest FLIM systems, the acquisition can range from 0.02 to 2.5 fps (note that the frame rate reported in Ref. 13 will be much slower if more gates are used as suggested in their experimental section). It shows that the acquisition would be the bottleneck if simple algorithms are applied, unless wide-field acquisition is applied.[23] Tables 6 and 7 also show that GPUs do offer speed enhancement when a more precise analysis (using GA-32 or LSM) is needed, and the analysis frame rate is comparable to the acquisition rate. Due to different structure and processing schemes, the enhancement factor for each algorithm is different. Table 8 gives a deeper understanding about each algorithm implemented in the GPU by using the CUDA profiler, where one can see the time of data transfer between GPU and CPU, achieved GPU occupancy, and so forth. Also, the acquisition and the image analysis with different image resolutions are shown in Fig. 13. This chart shows that the acquisition and the GPU analysis take comparable time for LSM.

3.2 *Experiment*

We demonstrate the performances of the GPU-based BCMM on two-photon FLIM images of gold nanorods (GNRs) Cy5 labeled A375 cells. GNRs were conjugated with Cy5 labeled oligonucleotide through a procedure described elsewhere.[54] The A375 cells were incubated with nanoprobes (GNR-Cy5) and fixed with paraformaldehyde. FLIM was performed using a confocal microscope (LSM 510, Carl Zeiss) equipped with a TCSPC module (SPC-830, Becker & Hickl GmbH). A femtosecond Ti:sapphire laser (Chameleon, Coherent) was tuned at 800 nm. The laser pulse has a repetition rate of 80 MHz and duration <200 fs.

Figure 14(a) shows the intensity–τ_F merged image obtained by BCMM, and it locates the (GNRs-Cy5)s by calculating their lifetimes ($\tau_F \sim 100$ ps, in agreement with Chen et al.[55]) and f_D. From the intensity image alone, it is impossible to identify the GNR sites. Combined with the average lifetime map, $f_D\tau_F + (1 - f_D)\tau_D$, in Fig. 14(b) and the f_D map in Fig. 14(c), the lifetime histogram of $\tau_D(\sim 2.93 \pm 0.16$ ns) at the GNR sites can be obtained to observe the energy transfer between Cy5 and GNRs, which offers a good indicator to study the hybridization of nanoprobes with targeting RNA in cells.[56]

Table 9 compares the CPU and GPU parallel processing speed based on the experimental data when using the BCMM algorithm. These results further verify the power of the proposed GPU-FLIM tool.

4 Discussion

In this study, the implementation and processing capabilities of a GPU-FLIM tool have been discussed and compared to traditional CPU-OpenMP based parallel analysis. We have also proposed some not unconventional but very important optimizations for GPU-based FLIM analysis.

Table 6 Processing time for single-exponential decay (unknown: τ, K).

Algorithm	CPU (ms)	GPU (ms)	Speedup (times)	Acquisition rates[13,18,19] (fps)	Image analysis frame rate (fps)
IEM	355.2	22.4	15.9	0.02 to 2.5	44
CMM	370.3	23.1	16.0		43
LSM	21,463.1	1170.2	18.3		0.85

Table 7 Processing time for double-exponential decay (unknown: τ_F, K, f_D).

Target	Algorithm	CPU (ms)	GPU (ms)	Speedup (times)	Acquisition rates[13,18,19] (fps)	Image analysis frame rate (fps)
τ_F	PM	472.2	22.8	20.7	0.02 to 2.5	44
	BCMM	537.1	26.3	20.4		38
	GA-32	76,410	3313.8	23.1		0.3
τ_{AVG}	PM	520	22.9	22.7	0.02 to 2.5	44
	BCMM	451.4	23.3	19.4		43
	GA-32	77,360.7	3320.2	23.3		0.3

Table 8 Results from CUDA profiler for each algorithm.

Algorithm	Data transfer (ms)	GPU computation (ms)	Occupancy (%)	Warp efficiency (%)
IEM		3.7	99.0	99.8
CMM		4.7	96.0	96.0
PM	18.7	4.1	99.0	100.0
BCMM		7.6	99.2	100.0
LSM		1151.5	54.1	97.8
GA		3295.1	52.9	98.0

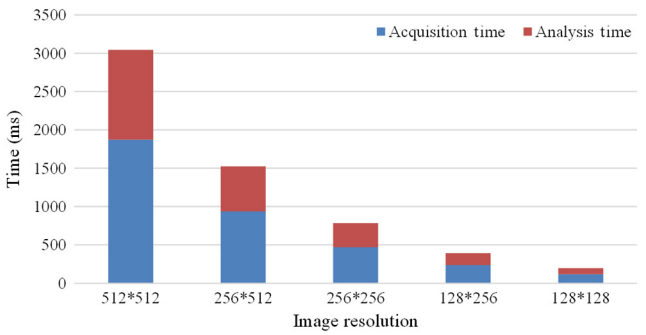

Fig. 13 Acquisition time (using the system reported in Ref. 18 as a reference) and image analysis time for LSM with different image resolutions.

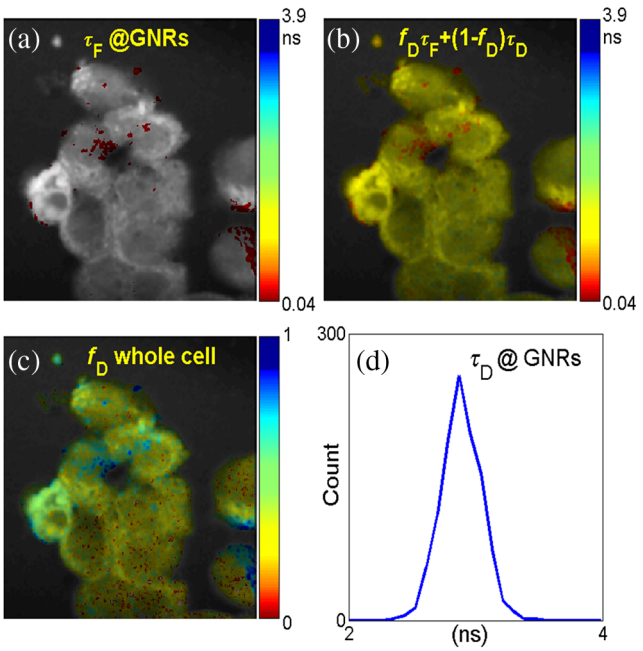

Fig. 14 (a) Intensity–τ_F merged image showing $\tau_F \sim 100$ ps and the locations of the gold nanorods (GNRs). (b) Image of $f_D\tau_F + (1 - f_D)\tau_D$, (c) f_D image, and (d) τ_D histogram at the GNR sites.

Table 9 Experiment results with CPU-OpenMP and GPU.

Target	Algorithm	CPU (ms)	GPU (ms)	Speedup (times)
τ_F	$BCMM_1$ (τ_D fixed)	111.24	5.4	20.6
	$BCMM_2$ (τ_D unknown)	169.3	9.3	18.2

FLIM analysis is well suited for GPU acceleration because it is highly parallelizable. Each pixel in an FLIM frame can be processed independently of any other pixel, and, depending on the details of the algorithm, there is a lot of room for parallelization even within the processing of an individual pixel.

From our simulations and experimental results, we observe that traditional iterative algorithms show better precision and working range and therefore remain the gold standard. On the other hand, although simple algorithms generally do not generate results of similar precision, they are much faster; for example, IEM is ~50 times faster than LSM. On the other hand, the photon efficiency (for single-exponential decays) of IEM is slightly worse than LSM- or MLE-based algorithms. To achieve the same precision for IEM, one needs to collect

2.5-fold more photons,[57] which offsets some of the speed advantage of IEM, albeit by no means all of it.[26] The requirements of precision and speed will vary depending on the research application, and a comprehensive FLIM analysis tool containing both noniterative and iterative algorithms is hence desirable. With the developed GPU-FLIM tool, we are able to achieve up to 24-fold enhancement over traditional CPU-based solutions, which now enables real-time or even video-rate FLIM analysis for simple algorithms (more than 37 fps can be achieved for 512×512 images). For the more precise iterative algorithms, smaller FLIM images can be analyzed in real time; for example, it takes 59 ms to generate a frame of a 128×128 FLIM image with 128 time bins using LSM. However, we expect to achieve up to 10 fps for 256×256 images (our tool is currently at about 3 fps) by optimizing the LSM algorithm and GPU implementation specifically for this image size in the near future. The acquisition of the fastest TCSPC FLIM systems is currently only $\sim$2 fps for 256×256 images. Therefore, the bottleneck still remains image acquisition; however, with rapid advances in image sensors and microscopy technologies, we believe that the acquisition rate will be further enhanced in the near future.

As mentioned above, lifetime estimations are parallelizable across all pixels (i.e., each pixel is independent), typically in a 512×512 array. However, GPU analysis is not 512×512-fold faster than CPU parallel computing. There are two major limitations to the achieved GPU speedup: limitations of GPU hardware and limitations of the parallelization of FLIM algorithms. With respect to the former, although processing for each pixel is assigned into a separate thread (for noniterative algorithms) or block (for iterative algorithms), these do not mean that every pixel can be processed simultaneously because each GPU has limited resources (e.g., cores, memory). The exact number of pixels that can be launched in parallel varies with different GPU hardware and algorithm details, and they are at most $15 \times 2048 = 30{,}720$ and $15 \times (2048/256) = 120$, respectively, for the GPU used here. The remaining pixels are processed when the first batch has completed and the GPU resources become available again. Another hardware limitation to GPU acceleration is that a single core of a GPU is not as powerful as a CPU core.

We can easily find that runtime kernels of noniterative and iterative algorithms have different configurations, that is, each pixel is processed in a single thread for noniterative algorithms, in contrast to a block of threads for iterative algorithms. This leads to the second consideration: speedups are also limited by the degree to which algorithms can be parallelized. None of the algorithms described above can be fully parallelized with respect to the time bins, which means GPU resources cannot be fully occupied during the entire process of lifetime estimation. This leads to less than linear speedups as a function of the number of threads used. For noniterative algorithms, since the computations for each pixel need less hardware resources, this configuration allows a large number of pixels to be launched simultaneously. For iterative algorithms, one could also consider the same solution where each thread on the GPU processes an entire pixel instead of sharing this work in a block of threads with one thread per time bin as was done here. This would be more fully parallelizable, but it is unlikely to lead to further speedups because such a scheme would quickly conflict with the small amounts of registers and shared memory that are available on GPU chips.

Between the two types of limitations, it is certainly the hardware capabilities that are more constraining at the moment. However, the development of GPU hardware continues with impressive advancements in short time scales, so that it is guaranteed that in the future GPUs will be able to boost FLIM analysis even further without a need for redevelopment of the parallelization strategy.

Due to their large, unused potential, we will continue to explore the use of GPU acceleration and the parallelization of existing algorithms. In addition, we will begin to develop GPU-friendly algorithms. Once we have established real-time, wide-range, and high-precision FLIM analysis, we will implement an entire FLIM system, including both high-speed acquisition and high-performance FLIM analysis, to be used in the biological sciences, chemistry, medical research, and so forth.

5 Conclusion

FLIM has recently gained attention as a powerful technique in biomedical imaging and clinical diagnosis applications, and there is an increasing demand for high-speed FLIM systems. In this article, we have proposed a flexible and reliable processing strategy for FLIM analysis using GPU acceleration, which can replace CPU-only solutions, allowing considerable speed improvements without loss of quality. The presented high-speed analysis tool has been implemented with some typical but important GPU code optimizations, tailored to the specific research area of fast, parallel lifetime analysis. The performance of the tool has been verified with synthesized and experimental data, demonstrating substantial potential for GPU acceleration in rapid FLIM analysis.

Acknowledgments

The authors would like to thank the China Scholarship Council, NVIDIA, and the Royal Society (RG140915) for supporting this work. The authors would also like to acknowledge the Biotechnology and Biological Sciences Research Council (BBSRC grant: BB/K013416/1), the Engineering and Physical Sciences Research Council (EPSRC grant: EP/J019690/1), and the technical support from G. Wei and J. Sutter.

References

1. L. C. Chen et al., "Fluorescence lifetime imaging microscopy for quantitative biological imaging," in *Digital Microscopy*,]Greenfield Sluder and David E. Wolf, Eds., 4th ed., Vol. 114, pp. 457–488, Academic Press, Waltham, Massachusetts (2013).
2. M. Nobis et al., "Intravital FLIM-FRET imaging reveals dasatinib-induced spatial control of SRC in pancreatic cancer," *Cancer Res.* **73**(15), 4674–4686 (2013).
3. K. Okabe et al., "Intracellular temperature mapping with a fluorescent polymeric thermometer and fluorescence lifetime imaging microscopy," *Nat. Commun.* **3**, 705 (2012).
4. M. A. Yaseen et al., "In vivo imaging of cerebral energy metabolism with two-photon fluorescence lifetime microscopy of NADH," *Biomed. Opt. Express* **4**(2), 307–321 (2013).
5. J.-D. J. Han et al., "Evidence for dynamically organized modularity in the yeast protein–protein interaction network," *Nature* **430**(6995), 88–93 (2004).
6. A. Miyawaki et al., "Dynamic and quantitative Ca2+ measurements using improved cameleons," *Proc. Natl. Acad. Sci.* **96**(5), 2135–2140 (1999).
7. A. Agronskaia, L. Tertoolen, and H. Gerritsen, "High frame rate fluorescence lifetime imaging," *J. Phys. D: Appl. Phys.* **36**(14), 1655–1662 (2003).

8. M. Zhao, Y. Li, and L. Peng, "FPGA-based multi-channel fluorescence lifetime analysis of Fourier multiplexed frequency-sweeping lifetime imaging," *Opt. Express* **22**(19), 23073–23085 (2014).
9. Q. Zhao et al., "Modulated electron-multiplied fluorescence lifetime imaging microscope: all-solid-state camera for fluorescence lifetime imaging," *J. Biomed. Opt.* **17**(12), 126020 (2012).
10. G. I. Redford and R. M. Clegg, "Polar plot representation for frequency-domain analysis of fluorescence lifetimes," *J. Fluoresc.* **15**(5), 805–815 (2005).
11. H. Chen and E. Gratton, "A practical implementation of multifrequency widefield frequency-domain fluorescence lifetime imaging microscopy," *Microsc. Res. Tech.* **76**(3), 282–289 (2013).
12. R. A. Colyer, C. Lee, and E. Gratton, "A novel fluorescence lifetime imaging system that optimizes photon efficiency," *Microsc. Res. Tech.* **71**(3), 201–213 (2008).
13. D. M. Grant et al., "High speed optically sectioned fluorescence lifetime imaging permits study of live cell signaling events," *Opt. Express* **15**(24), 15656–15673 (2007).
14. W. Becker, "Fluorescence lifetime imaging—techniques and applications," *J. Microsc.* **247**(2), 119–136 (2012).
15. L. Turgeman and D. Fixler, "Photon efficiency optimization in time-correlated single photon counting technique for fluorescence lifetime imaging systems," *IEEE Trans. Biomed. Eng.* **60**(6), 1571–1579 (2013).
16. S. P. Poland et al., "A high speed multifocal multiphoton fluorescence lifetime imaging microscope for live-cell FRET imaging," *Biomed. Opt. Express* **6**(2), 277–296 (2015).
17. S. Kumar et al., "Multifocal multiphoton excitation and time correlated single photon counting detection for 3-D fluorescence lifetime imaging," *Opt. Express* **15**(20), 12548–12561 (2007).
18. J. L. Rinnenthal et al., "Parallelized TCSPC for dynamic intravital fluorescence lifetime imaging: quantifying neuronal dysfunction in neuroinflammation," *PLoS One* **8**(4), e60100 (2013).
19. S. P. Poland et al., "A high speed multifocal multiphoton fluorescence lifetime imaging microscope for live-cell FRET imaging," *Biomed. Opt. Express* **6**(2), 277–296 (2015).
20. C. Veerappan et al., "A 160 × 128 single-photon image sensor with on-pixel 55 ps 10b time-to-digital converter," in *IEEE Int. Solid-State Circuits Conf.*, pp. 312–314 (2011).
21. 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 J. Solid-State Circuits* **49**(4), 867–880 (2014).
22. D. Tyndall et al., "A high-throughput time-resolved mini-silicon photomultiplier with embedded fluorescence lifetime estimation in 0.13 μm CMOS," *IEEE Trans. Biomed. Circuits Syst.* **6**(6), 562–570 (2012).
23. D. D. U. Li et al., "Video-rate fluorescence lifetime imaging camera with CMOS single-photon avalanche diode arrays and high-speed imaging algorithm," *J. Biomed. Opt.* **16**(9), 096012 (2011).
24. J. A. Jo et al., "Novel ultra-fast deconvolution method for fluorescence lifetime imaging microscopy based on the Laguerre expansion technique," in *Proc. of the 26th Annual Int. Conf. of the IEEE Engineering in Medicine and Biology Society*, Vol. **1**, pp. 1271–1274 (2004).
25. S. P. Chan et al., "Optimized gating scheme for rapid lifetime determinations of single-exponential luminescence lifetimes," *Anal. Chem.* **73**(18), 4486–4490 (2001).
26. D. U. Li et al., "Real-time fluorescence lifetime imaging system with a 32 × 32 0.13 microm CMOS low dark-count single-photon avalanche diode array," *Opt. Express* **18**(10), 10257–10269 (2010).
27. Y. J. Won, W. T. Han, and D. Y. Kim, "Precision and accuracy of the analog mean-delay method for high-speed fluorescence lifetime measurement," *J. Opt. Soc. Am. A* **28**(10), 2026–2032 (2011).
28. M. A. Digman et al., "The phasor approach to fluorescence lifetime imaging analysis," *Biophys. J.* **94**(2), L14–L16 (2008).
29. A. Leray et al., "Spatio-temporal quantification of FRET in living cells by fast time-domain FLIM: a comparative study of non-fitting methods," *PLoS One* **8**(7), e69335 (2013).
30. I. Gregor et al., "Fast algorithms for the analysis of spectral FLIM data," *Proc. SPIE* **7903**, 790330 (2011).
31. D. U. Li, H. Yu, and Y. Chen, "Fast bi-exponential fluorescence lifetime imaging analysis methods," *Opt. Lett.* **40**(3), 336–339 (2015).
32. M. Elangovan, R. N. Day, and A. Periasamy, "Nanosecond fluorescence resonance energy transfer-fluorescence lifetime imaging microscopy to localize the protein interactions in a single living cell," *J. Microsc.* **205**, 3–14 (2002).
33. P. Hall and B. Selinger, "Better estimates of exponential decay parameters," *J. Phys. Chem.* **85**(20), 2941–2946 (1981).
34. A. A. Istratov and O. F. Vyvenko, "Exponential analysis in physical phenomena," *Rev. Sci. Instrum.* **70**(2), 1233–1257 (1999).
35. P. R. Barber et al., "Global and pixel kinetic data analysis for FRET detection by multi-photon time-domain FLIM," *Proc. SPIE* **5700**, 171–181 (2005).
36. S. C. Warren et al., "Rapid global fitting of large fluorescence lifetime imaging microscopy datasets," *PLoS One* **8**(8), e70687 (2013).
37. P. J. Verveer, A. Squire, and P. I. Bastiaens, "Global analysis of fluorescence lifetime imaging microscopy data," *Biophys. J.* **78**(4), 2127–2137 (2000).
38. D. D. U. Li et al., "Time-domain fluorescence lifetime imaging techniques suitable for solid-state imaging sensor arrays," *Sensors* **12**(5), 5650–5669 (2012).
39. M. Broxton et al., "Wave optics theory and 3-D deconvolution for the light field microscope," *Opt. Express* **21**(21), 25418–25439 (2013).
40. Y. Jian, K. Wong, and M. V. Sarunic, "Graphics processing unit accelerated optical coherence tomography processing at megahertz axial scan rate and high resolution video rate volumetric rendering," *J. Biomed. Opt.* **18**(2), 026002 (2013).
41. G. Yan et al., "Fast cone-beam CT image reconstruction using GPU hardware," *J. X-ray Sci. Technol.* **16**(4), 225–234 (2008).
42. X. L. Dean-Ben, A. Ozbek, and D. Razansky, "Volumetric real-time tracking of peripheral human vasculature with GPU-accelerated three-dimensional optoacoustic tomography," *IEEE Trans. Med. Imaging* **32**(11), 2050–2055 (2013).
43. R. Zanella et al., "Towards real-time image deconvolution: application to confocal and STED microscopy," *Sci. Rep.* **3**, 2523 (2013).
44. M. Ploschner and T. Cizmar, "Compact multimode fiber beam-shaping system based on GPU accelerated digital holography," *Opt. Lett.* **40**(2), 197–200 (2015).
45. NVIDIA, "Parallel Programming and Computing Platform CUDA," NVIDIA, 23 June 2007, http://www.nvidia.com/object/cuda_home_new.html (29 February 2015).
46. N. Wilt, *The CUDA Handbook: A Comprehensive Guide to GPU Programming*, Addison-Wesley, Boston, Mass., London (2013).
47. N. Corporation, *CUDA C Programming Guide*, NVIDIA, Santa Clara, California 2014).
48. A. Neic et al., "Accelerating cardiac bidomain simulations using graphics processing units," *IEEE Trans. Biomed. Eng.* **59**(8), 2281–2290 (2012).
49. J. A. Goodman, D. Kaeli, and D. Schaa, "Accelerating an imaging spectroscopy algorithm for submerged marine environments using graphics processing units," *IEEE J. Sel. Topics Appl. Earth Obs. Remote Sens.* **4**(3), 669–676 (2011).
50. J. Sanders and E. Kandrot, *CUDA by Example: An Introduction to General-Purpose GPU Programming*, Addison-Wesley, Upper Saddle River; London (2011).
51. M. Harris, "Optimizing parallel reduction in CUDA," *NVIDIA Develop. Technol.* **2**(4), 1–38 (2007).
52. NVIDIA, *CUDA C Best Practices Guide*, NVIDIA Corporation, Santa Clara, California 2014).
53. D. Kirk and W. M. Hwu, *Programming Massively Parallel Processors Hands-on With CUDA*, Morgan Kaufmann Publishers, Burlington, MA (2010).
54. Y. Zhang et al., "FD 178: surface plasmon enhanced energy transfer between gold nanorods and fluorophores: application to endosytosis study and RNA detection," *Faraday Discuss.* **178**, 383–394 (2014).
55. Y. Chen et al., "Creation and luminescence of size-selected gold nanorods," *Nanoscale* **4**(16), 5017–5022 (2012).
56. Y. Zhang, D. J. Birch, and Y. Chen, "Energy transfer between DAPI and gold nanoparticles under two-photon excitation," *Appl. Phys. Lett.* **99**, 103701 (2011).
57. D. U. Li et al., "On-chip, time-correlated, fluorescence lifetime extraction algorithms and error analysis," *J. Opt. Soc. Am. A* **25**(5), 1190–1198 (2008).

Gang Wu is a PhD student at the University of Sussex and a visiting scholar at the University of Strathclyde. He received his MS degree in system analysis and integration from Zhejiang University in 2013. His current research interests include fluorescence lifetime imaging microscopy (FLIM) analysis, fluorescence-based sensing systems, graphic processing unit (GPU) computing, and biophotonics.

Thomas Nowotny is a professor in informatics at the School of Engineering and Informatics, University of Sussex, United Kingdom. He has a track record in modeling insect olfactories, hybrid computer/brain systems, GPU computing, and artificial neural networks. He has been a pioneer in GPU computing for neural network using NVIDIA® CUDA™ and developed the code-generation-based framework GeNN currently used in the Green Brain project (EPSRC) and in the Task 11.3.6 in the EU Human Brain Project.

Yu Chen is a senior lecturer in nanometrology at the Department of Physics, University of Strathclyde, United Kingdom. Her recent research focuses on fluorescent gold nanoparticles and surface plasmon enhanced effects, including two-photon luminescence, energy transfer, and SERRS from noble metal nanoparticles, arrays, and porous media, with strong links to biomedical imaging and sensing, as well as nanoparticle–cell interaction and cytotoxicity.

David Day-Uei Li is a senior lecturer in biophotonics at the Strathclyde Institute of Pharmacy & Biomedical Sciences, University of Strathclyde, United Kingdom. He has research projects in developing CMOS FLIM or spectroscopy systems, FPGA/GPU computing, tomography, FLIM imaging analysis, and mixed-signal circuits. His research exploits advanced sensor technologies to reveal low-light but fast biological phenomena. He has published more than 60 journal and conference papers and holds 12 patents.