Image processing tools for petabyte-scale light sheet microscopy data

Xiongtao Ruan[1*], Matthew Mueller[1], Gaoxiang Liu[1], Frederik Görlitz[1], Tian-Ming Fu[2], Daniel E. Milkie[2], Joshua Lillvis[2], Alison Killilea[1], Eric Betzig[1,2,3*†] and Srigokul Upadhyayula[1,4,5*†]

[1]Department of Molecular and Cell Biology, University of California, Berkeley, CA 94720.
[2]Howard Hughes Medical Institute, Janelia Research Campus, Ashburn, VA 20417.
[3]Department of Physics, Howard Hughes Medical Institute, Helen Wills Neuroscience Institute, University of California, Berkeley, CA 94720.
[4]Molecular Biophysics and Integrated Bioimaging Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720.
[5]Chan Zuckerberg Biohub, San Francisco, CA 94158.

*Corresponding authors. E-mails: xruan@berkeley.edu; betzige@janelia.hhmi.org; sup@berkeley.edu;
†These authors contributed equally, names are listed alphabetically

Abstract

Light sheet microscopy is a powerful technique for visualizing dynamic biological processes in 3D. Studying large specimens or recording time series with high spatial and temporal resolution generates large datasets, often exceeding terabytes and potentially reaching petabytes in size. Handling these massive datasets is challenging for conventional data processing tools with their memory and performance limitations. To overcome these issues, we developed LLSM5DTools, a software solution specifically designed for the efficient management of petabyte-scale light sheet microscopy data. This toolkit, optimized for memory and performance, features fast image readers and writers, efficient geometric transformations, high-performance Richardson-Lucy deconvolution, and scalable Zarr-based stitching. These advancements enable LLSM5DTools to perform over ten times faster than current state-of-the-art methods, facilitating real-time processing of large datasets and opening new avenues for biological discoveries in large-scale imaging experiments.

Keywords: Image processing, light sheet microscopy, deskew and rotation, deconvolution, stitching, large-scale processing, real-time processing

1 Introduction

In bioimaging, light sheet microscopy stands out as a key innovation that enables rapid and long-term 3D visualization of cell, tissue, and organ dynamics [1]. A transformative technique within this domain is lattice light sheet microscopy (LLSM) [2], which allows access to subcellular dynamics with high spatial and temporal resolution in single or multicellular environments [3], while maintaining reduced photobleaching and phototoxicity. When combined with expansion microscopy, LLSM enables imaging millimeter-scale expanded samples with effective super-resolution [4]. With an increase in imaging speed, volume, frequency, and duration, the data produced from a single experiment can be staggering, often ranging from hundreds of gigabytes to petabytes. This explosion in data generation rate introduces significant challenges in storage and processing.

At the heart of these challenges are several factors that complicate visualization, analysis, and consequently the assessment of data utility: the intrinsic three-dimensional nature of the data, its vast size, and its existence in a non-Cartesian coordinate space. Large individual volumetric stacks can be unwieldy, defying conventional processing methods. Moreover, processing entire datasets comprising thousands of volumetric tiles can be a compute-intensive task. These hurdles impede the ability to rapidly extract biological insights from large-scale imaging experiments.

Despite the development of tools designed for light-sheet data preprocessing, such as deskew and rotation [5, 6], deconvolution [7, 8], stitching [9], and visualization [10, 11], their utility wanes for data sizes surpassing the terabyte threshold due to inefficient and non-scalable implementations. Many of these tools exist as standalone applications, providing only partial processing steps in a specialized context and varying input formats and requirements. This often requires extensive manual effort to integrate them into multi-step workflows, limiting their utility especially when handling large-scale data.

To address these challenges, we developed LLSM5DTools, a software solution adept at facilitating real-time feedback and processing of petabyte-scale light sheet imaging data. The software suite contains commonly used pre- and post-processing tools, all finely tuned for optimal memory utilization and performance. These include 3D point source detection [12], deskew, rotation, deconvolution, and image stitching, to name a few, all integrated into a robust, high-performance computing framework capable of distributed processing.

2 Results

LLSM5DTools incorporates novel algorithms to enhance image input/output (IO) throughput and streamline critical processing steps, including geometric transformations, deconvolution, and stitching. By utilizing the Zarr data format for image storage and processing [13], alongside custom-built image readers and writers, the software achieves efficient parallel IO operations for high throughput processing. Our integrated approach to deskew and rotation

effectively bypasses the traditionally time-consuming and memory-intensive deskewing process. The instantly converged Richardson-Lucy deconvolution algorithm we developed ensures accurate signal reconstruction across a wide range of light sheets. Furthermore, our scalable stitching pipeline, featuring user-defined pre-processing, automatic cross-correlation-based registration, and block-based stitching fusion, can handle tens of thousands of tiles and petabyte-scale datasets.

Beyond these functionalities, LLSM5DTools brings together commonly used processing methods such as image cropping, projection, and resampling into a unified computing framework. This integration facilitates partitioned processing and merging, enabling efficient parallel processing of petabyte-scale datasets. The software is designed for the seamless execution of multiple processing steps and incorporates a real-time pre-processing feature designed for long-term time series imaging or high-throughput large sample imaging. To ensure broad accessibility, LLSM5DTools features a user-friendly graphical user interface (GUI), allowing users of all skill levels to operate the software with ease.

2.1 Overall design: distributed computing with the acceleration of processing

The integration of high frame rate and sensitivity modern cameras with advancements in microscopy and computing hardware has enabled light sheet microscopes to capture images at a staggering rate of nearly four terabytes per hour per camera. This throughput, while remarkable, presents formidable challenges in sustained image acquisition, real-time (de)compression, storage, and processing, especially when relying on conventional single workstation methods. To tackle these issues effectively at scale, we developed a distributed computing architecture comprising a cluster of computing nodes and networked data storage servers. This setup is essential for the uninterrupted streaming and real-time processing of vast quantities of data continuously acquired over extended periods. Our standard workflow, including data acquisition, storage, and preprocessing, is illustrated in Fig 1a.

Moving beyond the limitations of single-machine processing, our approach involves advanced processing strategies for scalability and efficiency in handling vast amounts of data. We created a lightweight distributed computing framework for user-defined functions. Central to this framework is a master job that orchestrates all operations, distributing tasks across the computing cluster and monitoring their progress to ensure completion (Fig 1b). The master job is designed to promptly resubmit any failed job with additional resources or to execute tasks and concurrently oversee tasks, maximizing computing resource utilization. Our framework offers greater flexibility for various task types, enhanced robustness against worker failures, and seamless integration with our processing methods compared to existing distributed computing frameworks like Spark [14] and Dask [15].

In this framework, we assess each task's memory requirements, enabling the master job to allocate tasks based on these metrics to optimize scalability. In practical application, we partition the complete dataset or set of tasks into distinct, self-contained subtasks, each appropriately sized for processing by an individual worker unit with one or more CPU cores or GPUs (Fig. 1c-d). Using this approach, we have carefully engineered distributed frameworks to effectively manage all processing steps in LLSM5DTools (Fig. 1e).

2.2 Fast image readers and writers

Efficient image reading and writing are essential for real-time image processing. Unfortunately, conventional image readers and writers for the widely used Tiff format are not designed for large-scale compressed data, being restricted to single-threaded operations (e.g., libtiff). For instance, reading and writing an 86 GB 16-bit Tiff file ($1,800 \times 512 \times 50,000$) with libtiff (LZW compression) in MATLAB takes approximately 8.4 and 16 minutes, respectively (Fig S1c, last group). These slow IO speeds pose a considerable bottleneck for efficient image processing.

To rectify this inefficiency, we have developed a highly optimized Tiff reader and writer in C++. This implementation leverages the OpenMP framework to facilitate concurrent multi-threaded reading and writing, and is seamlessly integrated with the underlying libtiff library. Our Cpp-tiff reader and writer are over 22x and 7-8x faster than conventional ones, respectively for compressed data (Fig 2a-b and Fig S1 a-b in a 24-core node). Moreover, our implementation significantly outperforms the fast Python reader and writer library for Tiff files (tifffile in Python, Fig 2a-b, and Fig S1 a-b). Importantly, our reader and writer demonstrate linear scalability with the number of CPU cores (Fig S1 e-f).

Although the Tiff format is commonly used for images, it is not the most efficient for parallel reading and writing, especially for very large image datasets. One major limitation is its single-container structure for file writes, which restricts it to single-threaded operations. To overcome this constraint, we adopted the Zarr format [13], a next-generation file format designed for multi-dimensional data. Zarr efficiently stores data in non-overlapping chunks of uniform size (border regions can be smaller) and saves them as individual files. This format is similar to other large-scale microscopy data formats like N5 [16], OME-Zarr [17], and TensorStore [18].

The design of the Zarr format allows individual workers to access specific data regions efficiently, minimizing unnecessary data retrieval. These workers can also save distinct regions to disk independently and in parallel, avoiding interference. We also developed highly optimized C/C++ code that leverages OpenMP for the Zarr reader and writer. Compared to the existing implementation, our optimized code is 10-23 faster for reading and 5-8 times faster for writing (Fig 2c-d, Fig S1 c-d), as well as almost linear scalability (Fig S1 g-h). Our implementation is also approximately 5-10 times and 5-8 times faster for

the reader and writer, respectively, when compared with native Zarr implementation in Python (Fig 2c-d, Fig S1 c-d). Furthermore, we found that the zstd compression algorithm at compression level 1 has the best overall performance for microscopy images, in terms of compression ratio, read and write times, when compared with the lz4 algorithm at level 5 (default in Zarr) (Fig S1 i-k).

2.3 Fast combined deskew and rotation

In the context of the dual-objective open-top light-sheet microscope, wherein the detection objective is oriented at an acute angle relative to the stage, it may be necessary to apply geometric transformations to the raw images to align them into the actual sample space (i.e., the scanning direction parallel to the stage). This process generally involves two distinct steps: first, deskewing to shear the raw images and reposition the z slices correctly, and second, rotation to orient the deskewed images horizontally, as depicted in Figure 3a-b. However, due to the drastically increased data size, the deskew operation may be considerably time-consuming or even encounter out-of-memory problems, particularly for large images with many z slices (Fig 3c).

To address the issue, we proposed to combine the deskew and rotation processes into a single, unified step. This approach is feasible because both operations are rigid geometric transformations, specifically shearing and rotation. The combination is seamless for continuous data from the mathematical theory. In practice, the performance of this combination depends on factors such as the scan step size and the xy voxel size, due to the discrete nature of image data, represented by voxels. We introduce a parameter called the "skew factor", defined as the ratio of the scan step size in the xy plane to the xy voxel size. When the skew factor is small (i.e., < 2.0), this combination is readily feasible (Fig 3d top). However, when the skew factor is large, as depicted in Fig S2 a-b, artifacts may manifest due to the interpolation of voxels that are spatially distant within the actual sample space during the shearing operation. To resolve the issue, our solution is to first interpolate the data in the skewed space before the combined deskew and rotation. We use pairs of adjacency frames to interpolate additional frames within the proper coordinate system to generate an image with a smaller effective step size and reduced skew factor (Fig 3d bottom), followed by combined deskew and rotation. As shown in Fig 3e and Fig S2 c-e, the combined deskew and rotation operations yield nearly identical results compared to the one from separate operations, with only minor numerical errors. Strikingly, the combined operation is increasingly faster and more memory efficient with the number of frames, which enables the processing of 10 times larger data with the same computational resources (Fig 3f and g, and Fig S2 f-g). Combining the fast Tiff reader and writer with the combined deskew and rotation operations, our processing speed is 20x faster than the conventional ones and can process much larger data (Fig 3h and Fig S2 h). Additionally, resampling and cropping, if involved, can be also

integrated with deskewing and rotation to optimize processing efficiency and minimize storage requirements for intermediate data.

2.4 OTF masked Wiener Richardson-Lucy deconvolution

Deconvolution plays a crucial role in reconstructing actual sample information for light sheet microscopy images, especially for light sheets with stronger side lobes for higher axial resolutions [19]. RL deconvolution is the most widely used deconvolution method for LLSM images due to its reconstruction performance and robustness [20, 21]. However, the RL method is an iterative method that requires 10-200 iterations to converge depending on the type of light sheet. Relative to deskew, rotation, and stitching, RL deconvolution is the most expensive step, which is exacerbated for big data, even with GPU computing. Leveraging the benefits of the combined deskew and rotation and eliminating the need for deconvolution in empty regions of deskewed images, we demonstrate that deconvolution on the raw skewed data not only accelerates the process but also yields better results with fewer edge artifacts compared to deconvolution on the deskewed data due to empty regions (Fig S3 a). The deskewed space and skewed space point spread functions (PSFs) are illustrated in Fig S3 b and c.

Inspired by the work of Zeng et al. (2001) [22] and Guo et al. (2020) [8], we optimized the iteration process by adapting certain backward projectors. Despite its notable speed improvement over the traditional RL method, the Butterworth-Wiener (BW) backward projector method introduced by Guo et al. (2020) was demonstrated on Gaussian light sheets with ellipsoid support, thus failing in achieving full-resolution reconstruction of lattice light sheet images, as illustrated in Fig 4a and b, and Fig S3 d-f. Upon thorough analysis, we have determined that the BW backward projector consistently truncates the spectrum for the Optical Transfer Function (OTF), resulting in the loss of information near the edges of the OTF support (Fig 4 a and b, Fig S3 d-f). This limitation is particularly pronounced when dealing with light sheets with high axial resolutions, such as Hexagonal, HexRect, and Multi-Bessel light sheets (refer to [19] for detailed descriptions of these light sheets). In these instances, the OTF supports are closer to rectangles in the xz and yz planes, largely deviating from the elliptical shape employed in the BM method. An additional concern within the BW method involves the potential excessive suppression of high-frequency regions near the border of the BW back projector's ellipsoid. This suppression may pose challenges in achieving an effective reconstruction of high-frequency information during the subsequent deconvolution process. To address these issues, we propose utilizing the OTF support to create an apodization function to eliminate noise close and beyond the OTF support, rather than employing the Butterworth filter (with an ellipse shape) in the BW method (Fig 4a, b, Fig S3 d-f, Fig S4 a-b). As illustrated in Fig S4 c-d, our approach involves creating a weight matrix based on the distance transform of the mask of the OTF support with decreasing weights from 1 to 0 near the OTF support, followed by multiplying with the Wiener filter. Compared to the

BW method, the OTF masked Wiener (OMW) technique covers full relevant frequencies in the Fourier space (Fig 4c and d, Fig S3 g and h), and achieves full-resolution image reconstruction while maintaining the convergence speed, as depicted in Fig 4e and f, and Fig S3 i-l. Since we utilize the OTF support from the provided PSF, the OMW method is more generic for PSFs with varying OTF support and requires fewer parameters compared to the BW method (i.e. skewed space deconvolution). Additionally, our implementation offers robust CPU and GPU support for deconvolution, boasting over a 10-fold speed improvement compared to traditional methods for both GPU and CPU (Fig 4g, h, Fig S5 a-f).

As an iterative method, the deconvolution performance of the RL method relies on the number of iterations. Balancing the number of iterations is crucial for precise reconstruction. Too few iterations yield a fuzzy, blurred result with incomplete sidelobe collapse, while too many iterations amplify noise, potentially collapsing and breaking object structures. To address this, we implemented the Fourier Shell Correlation (FSC) algorithm to estimate the optimal number of iterations [23]. For the conventional RL method, we utilize FSC to estimate the optimal number of iterations (Fig S5 g). As discussed in [19], it's evident that the ideal iteration count derived from the minimum mean resolution may fall short, necessitating a slightly higher threshold. Unlike the multi-region FSC in [19], we employ an iteration number that is closest to 101% of the minimum resolution number beyond the iteration with the minimum resolution number. In practice, for the OMW method, we found that conducting two iterations with suitable parameters yields the most optimal results. Consequently, we utilize FSC to determine the best Wiener parameter (Fig S5 h-i).

2.5 ZarrStitcher: Zarr-based scalable stitching

Stitching is a necessary processing step when imaging large specimens that require multiple fields of view to fully visualize, including organoids, tissues, organs, and whole organisms. Given the constraints of the field of view, small volumetric tiles are usually captured in evenly spaced grids with sufficient overlap regions to facilitate precise registration and stitching. With the advancement in high-resolution light sheet microscopy coupled with expansion microscopy, the scale of image data has drastically increased, reaching hundreds of terabytes and rapidly moving toward petabytes. This growth presents substantial challenges for existing stitching software, particularly in handling the large number of tiles, overall data size, and operational efficiency. To address these issues, we introduced a Zarr-based stitching framework ZarrStitcher for petabyte-scale image stitching, capable of theoretically handling unlimited data size. As illustrated in Fig 5a, ZarrStitcher involves three primary steps: data format conversion, cross-correlation registration, and stitching (fusion). As detailed in Sec 2.2, the Tiff format is not optimal for large-scale parallel processing. Thus, we convert tiles into the Zarr format if they are in Tiff format. This conversion step can accommodate user-defined

preprocessing functions applied to each tile, such as flat-field correction, data cropping, and other necessary adjustments. Due to sample drift and movement, stage drift, and stage motion errors, the recorded tile coordinates might not be accurate. To refine their placement, we employ a registration process using the cross-correlation method, which calculates the normalized cross-correlation between adjacent tiles to ensure accurate alignment [24]. Once the pairwise optimal shifts are computed, we apply a global optimization to infer the overall optimal shifts (Fig 5b, middle). This method better manages the potential discrepancies in relative shifts between neighboring tiles compared to the typical approach with a greedy local method (refer to Fig 5b, left). Moreover, when data is collected batch by batch with varying regular grids across batches, we implement a two-step optimization framework, where the global optimization is first applied to each batch, followed by an optimization across batches (Fig 5b, right). After registration, the subsequent step involves stitching the tiles together. We have developed a distributed framework for stitching, with individual tasks allocated for small regions to distribute among workers. Additionally, we have incorporated various methods to address overlapping regions, including direct merging (none), mean, median, or feather blending. Feather blending, a type of weighted averaging with weights determined by distances to the border, has proven to be particularly effective for merging overlapping regions.

Compared to existing stitching software such as BigStitcher [9] and stitch-spark [4], our stitching method stands out with its exceptional speed and stitching precision (Fig S6 and Table S1). BigStitcher struggles with large-scale registration involving hundreds or thousands of tiles and slow stitching speed (Fig S6). On the other hand, stitch-spark is limited to Tiff file inputs smaller than 4GB, and it falls short in terms of registration accuracy and stitching speed (Fig S6 i-m). In contrast, our method excels by stitching a massive 120 TB dataset in just 3.5 hours with 20 computing nodes (a total of 480 CPU cores, Table S1).

By integrating fast readers and writers, combined deskew and rotation, and ZarrStitcher, we implemented a real-time feedback framework for microscopy acquisition. This framework adeptly oversees newly acquired images, swiftly applies (downsampled) deskew and rotation processes, and promptly stitches together either the 3D volumes or the 2D maximum projections in real-time. This enables immediate feedback for ongoing experiments, facilitating rapid analysis and decision-making. This tool serves multiple purposes, such as identifying acquisition errors (Fig S7 a) and establishing the cessation time for the acquisition (Fig S7 b). Our framework also facilitates real-time processing of full-resolution data and following real-time visualization. In Fig 5c-f and Movie S1, we demonstrate the processing of a large field of view (mm^2) of 2D phase contrast images. This enables us to quickly identify specific cells in this large field view for interesting events such as cell division (Fig S8 a) or cell merging (Fig S8 b). In Fig 5g and Movie S2, we illustrate real-time processing of a long-term large field of view live cell imaging for the endoplasmic reticulum (ER)

and nuclei dynamics. To facilitate real-time visualization, we employed NVidia IndeX to visualize the data in full resolution [25]. This allows us to simultaneously inspect hundreds to thousands of cells at the same time. Notably, we were able to pinpoint numerous instances of cell division (Fig 5g), albeit these instances were transient for individual cells. Furthermore, this technology enables rapid detection of rare events, such as the division of a cell into three daughter cells (Fig 5h). Moreover, the whole pipeline can also be used to process other image modalities, such as two-photon and so on. We utilized the framework to analyze data for the live imaging of the mouse brain using the two-photon modality with a large field of view (Fig. S8 c-f and Movie S3), which allowed us to effectively track axons and dendrites over long distances.

2.6 Large-scale processing

Deskewing, rotation, deconvolution, and stitching are typically coupled preprocessing steps, and the sequence in which they are applied can vary. The order for the conventional approach is deskewing, deconvolution, rotation (if required), and stitching (if required). As previously highlighted, deskewing large datasets is time-consuming, and deconvolving deskewed data can be further slowed down by the large bounding regions that are largely empty. We have shown that it is possible to perform deconvolution directly on the raw skewed data, followed by deskewing and rotation (Fig S3a). For more efficiency with large datasets, our approach involves stitching the data in the skewed space before deconvolution (Fig 6a). This process eliminates the requirement to deconvolve additional areas in the overlapping regions. Hence, the optimal processing sequence is stitching (if necessary), followed by deconvolution, and then a combined deskew and rotation (Fig 6a).

When handling large datasets that exceed memory capacity, certain data processing steps become challenging due to their sizes. ZarrStitcher already enables the stitching of data that exceeds memory limitations for smaller-sized tiles where intermediate steps can be fitted into the memory. For stitching with large tiles, we developed the MIP slab-based stitching technique (Fig 6a, left). This approach involves downsampling the tiles with varying factors across axes (i.e., 100 on the z-axis, 2 on the x and y axes) to generate MIP slabs that fit into memory. The MIP slabs are used to calculate registration information and estimate distance transforms for feather blending, ensuring accurate stitching while addressing memory constraints (Fig S6 b, d, and f).

For deskewing, rotation, and deconvolution, we developed distributed frameworks that split the data into smaller, manageable regions, distributing them among multiple workers for efficient processing, followed by merging as the final output (Fig 6a, middle and right). The Zarr format enables seamless splitting and merging virtually, allowing workers to directly access and process specific regions and save the outcomes (with cropping if there are border buffers). In many imaging scenarios, a substantial amount of space beyond the region of interest (ROI) is empty to ensure complete coverage. Even when the ROI spans 80% of all three axes, it encompasses only up to 51.2% of the

entire 3D space. Processing these empty regions significantly slows down the processing, particularly in deconvolution (Fig S9 a, b, and f). To expedite the processing of large datasets, we've devised a method to define ROI based on Maximal Intensity Projection (MIP) to skip these empty regions (Fig S9 c). We can segment the masks of the ROI with the MIPs across all three axes easily, and then use them to define the object boundaries for both input and output to skip empty subregions (Fig S9 d, f-h). With these techniques in place, processing large-scale data up to petabytes scale becomes feasible and efficient. Our benchmarks across data sizes ranging from 1 TB to 1 PB demonstrate the linear scalability of our processing methods (Fig 6b-d).

With the large-scale processing framework, we have processed the data of large-scale super-resolution imaging of the Drosophila adult ventral nerve cord (VNC) with 8x expansion, as illustrated in Fig 6e-i and Movie S4. The sample was labeled to visualize glutamatergic neurons, which include all motor neurons, in one color (VGlut, cyan) and an additional subset of VNC neurons that includes a small number of motor neurons in another color (purple). As expected, the VNC predominantly features motor neurons that control the legs, wings, halteres, neck, and abdomen: diverse bundles of motor neuron dendrites and axons are distributed across various regions of the VNC (Fig 6e-i). Notably, this methodology allows us to capture the complete morphologies of entire neuron populations (e.g., motor neurons) as well as genetically defined subsets (Fig 6f). Therefore, the increased processing efficiency afforded by the framework developed here significantly enhances the ability to compare the super-resolution anatomical properties of large neuron populations and individually identifiable neurons across individuals, sexes, and species.

3 Discussion

The work we present here embodies several innovative algorithms and implementations to enable petabyte-scale light sheet microscopy image processing. First, we designed a generic computing framework to allow distributed computing with multiple-node computing for large-scale image processing. Second, we adopt the Zarr data format for parallel processing, and optimized readers and writers for both Tiff and Zarr formats, significantly enhancing processing speed. Thirdly, our combined deskew, rotation and resampling techniques operate over ten times faster and consume ten times less memory. Moreover, our rapid RL deconvolution method produces high-fidelity LLSM image reconstructions, boasting a speed one order of magnitude faster than traditional methods like RL and surpassing BW methods in performance. Demonstrating the feasibility and advantages of skewed space deconvolution, especially when paired with combined deskew and rotation, has been a key achievement. Additionally, our Zarr-based distributed stitching pipeline seamlessly handles terabytes to petabytes of data. Furthermore, our software contains the comprehensive framework for large-scale single-image deconvolution, and deskew and rotation by splitting the data and distributing the tasks to workers.

In addition to the above-mentioned key features, our real-time process and feedback pipeline provide immediate insights during image acquisition, aiding in debugging and determining stopping points. Moreover, our user-friendly GUI, tailored for non-programmers, offers common tools for efficient data processing and management. Our software achieves near real-time processing of petabyte-scale light sheet microscopy data and, theoretically, can handle exabyte-scale image data. It efficiently manages long-term live cell imaging and centimeter-scale samples, such as whole mouse brain images, providing robust preprocessing and enabling follow-up analysis and visualization. Most of the processing tools within our software are versatile and can be applied to images obtained through other imaging modalities, such as stitching, deconvolution, real-time processing, and useful tools.

Our tools provide a significant decrease in required computing hardware for large-scale image processing, resulting in a minimum tenfold reduction in computing costs. This impact is particularly pronounced when handling hundreds of terabytes and beyond, a demand that has grown due to the advancements in microscopy techniques influencing current trends in the field [4, 26]. The real-time processing feature coupled with real-time visualization promises accelerated feedback for biological experiments. The software will be useful for many important biological studies, including high throughput high-resolution 3D screening [27], long-term fast dynamics of cells, tissues, organoids, and organisms [3, 28], super-resolution imaging of large samples, and potentially spatial transcriptomics [29]. Recently, deep learning methods have been extensively applied to microscopy image processing [30–32], including deconvolution [33], denoising [34], registration [35], and so on. We aim to maintain simplicity, generality, and robustness in our software algorithms, without relying on extensive training data and prolonged training processes in typical deep-learning approaches. However, we recognize the potential of combining these algorithms with deep learning methods to enhance training data quality or preprocess data for specific deep learning models, aiding in feature extraction or decision-making processes.

Our software, while robust for petabyte-scale light sheet microscopy image processing, does have limitations. First, it primarily focuses on preprocessing tasks, offering tools for geometric transformation, deconvolution, stitching, and other common processing methods. Although integrating point source detection and tracking [12], it has limited functionalities for post-processing tasks like object detection, segmentation, and biological feature extraction and interpretation, crucial steps following the initial preprocessing. Second, the stitching pipeline primarily supports rigid registration for translation, which performs well in most scenarios. However, it may not be suitable for image tiles with rotation, scaling, or object movement (such as live imaging), necessitating approaches like rigid registration with translation, rotation, and rescaling, or even non-rigid methods such as elastic registration [36]. Lastly, while the Zarr format with zstd compression provides a better compression ratio, the software does not address storage issues for petabyte-scale or larger data. Real-time

preprocessing of raw data followed by massive compression during acquisition may be necessary to tackle this challenge.

In summary, the advanced capabilities for large-scale processing, coupled with user-friendly tools and a graphical user interface (GUI), make LLSM5DTools a valuable asset for handling petabyte-scale light sheet images. This facilitates real-time biological discovery and enhances overall research efficiency.

Online methods

Generic computing framework

Currently, our generic computing framework works on both single machines and Slurm-based computing clusters with multiple nodes for both CPU and GPU configurations. In the implementation, the master job, typically integrated into the pipeline, receives a collection of function strings, input file names, output file names, and relevant parameters for job settings, such as required memory, the number of CPU cores, system environments, and more. The function string, either a MATLAB function call or a bash command, delineates the command that worker jobs should execute for each task. The master job will initially verify the presence of output files, bypassing those that already exist. For a single machine or when Slurm job submission is disabled, the master job will autonomously execute tasks sequentially. When job submission is enabled and the tasks run on a computing cluster, the master job will format the Slurm command with function strings based on job parameters, followed by submitting worker jobs to the cluster. If enabled, the master job will also participate in the computing starting from the last pending task. Simultaneously, the master job will oversee the status of worker jobs, ensuring the completion of corresponding tasks. In the event of a worker job failure, the master job will automatically resubmit the task with an increased memory allocation (also CPU cores), typically doubling the original amount if feasible. This process continues until all tasks are completed or a predefined maximum retry threshold for job failures is reached. A parameter can be provided to allow users to define their own configuration file, enabling customization of Slurm-related parameters for their specific computing clusters. Thus, this framework is highly adaptable, capable of accommodating any user-defined function strings, and compatible with any Slurm-based computing cluster.

Fast Tiff and Zarr readers and writers

The Tiff reader and writer leverage the capabilities of the libtiff library in C++ with the MATLAB MEX interface for MATLAB. In the reading process, a binary search is employed to determine the number of z slices by identifying the last valid z slice, as there is no direct way to obtain the number of z slices in libtiff. This involves finding an interval containing the last z slice. To do so, it initializes start and end indices $s = 0$ and $t = 1$, respectively, followed by checking if slice t exists. If it exists, it sets $s = t$ and $t = 8t$, until slice t no longer exists. After that, it will get the last z slice by binary search. It checks for the existence of the middle z slice between slices s and t, denoted as $m = (s+t-1)/2$. If present, the start index s is updated to m; otherwise, the end index t is updated to $m-1$ (or 0 if $m = 0$). This iterative process continues until the start and end indices converge to the same value. After obtaining the number of z slices, the OpenMP framework is employed to efficiently distribute the reading tasks across multiple threads, partitioning the z slices into evenly

sized batches (except for the last one) for optimal parallel processing. For large 2D images, the Tiff strips are partitioned to facilitate multi-threaded reading using the OpenMP framework. For the Tiff writer, LZW compression from libtiff is adapted to support compression on individual z slices. This approach enables parallel compression across z slices, leveraging the OpenMP framework for multi-threading. Following compression, the compressed data is written to the disk using a single thread since a Tiff file is a single container, making parallel writing of compressed data infeasible.

There is no native Zarr reader and writer in MATLAB. A custom code in C++ has been adhering to the Zarr specification with massive parallelization and calls the reader and writer through the MATLAB MEX interface. In the implementation, the OpenMP framework is utilized for both the reader and writer to efficiently distribute the tasks across multiple threads with each chunk as an independent task. To update partial chunks, the process involves loading the entire chunk, replacing the specific region, rewriting the data, and subsequently saving the modified chunk file back to the disk. The Blosc library which contains a set of compression algorithms [37] is directly used. Within each task, the compression/decompression algorithm also leverages multi-threading, resulting in a dual-layered multi-threading to use the system resources more efficiently. As stated in the main text, the zstd compression with a level 1 setting is optimal in achieving a balance of compression ratio, read time, and write time. The increased compression ratio offers a substantial reduction in the overall data size for reading or writing, alleviating network load, particularly in extensive high-throughput processing scenarios where the network often acts as a bottleneck.

Combined deskew, rotation, resampling, and cropping

The concept is to execute deskew, rotate, and resampling (if a resampling factor is provided) in a single step by combining the geometric transformations. The fundamental geometric transform involves:

$$I_t = F_T(I)$$

where I is the original image, I_t represents the transformed image, $F_T(\cdot)$ denotes the image warp function according to the geometric transformation matrix T. The deskew operator applies a shear transformation defined by the shear transformation matrix S_{ds}. In the rotation process, there are four sub-steps: translating the origin to the image center, resampling in the z-axis to achieve isotropic voxels, rotating along the y-axis, and translating the origin back to the starting index. Let the transformation matrices be denoted as T_1, S, R, and T_2, respectively. If resampling factors are provided (by default as 1), then there are three additional sub-steps in resampling: translating the origin to the image center, resampling based on the factors provided, and translating the origin back to the start index. Let the transformation matrices in these sub-steps be T_{R1}, S_R, and T_{R2}, respectively.

In traditional approaches, these three steps are executed independently, resulting in multiple geometric transformations. This approach incurs substantial overhead in runtime and memory usage, particularly during the deskew step. It turns out we can combine deskew, rotation, and resampling into one single step, resulting in a unified affine transformation matrix:

$$A = S_{ds}(T_1 S R T_2)(T_{R1} S_R T_{R2})$$

In practice, this affine transformation matrix can be directly applied to the raw image if the scan step size is small. A quantity, denoted as "skew factor", is defined to describe the relative step size as

$$f_{sk} = d_z \cos\theta / p_x$$

where $\theta \in (-\pi/2, \pi/2]$ is the skewed angle, d_z denotes the scan step size, and p_x is the pixel size in the xy plane. If $f_{sk} \leq 2$, the direct combined processing operates smoothly without noticeable artifacts. For $f_{sk} > 2$, interpolation on the raw data within the skewed space is performed, taking into account their relative positions. To interpolate a z slice, neighboring slices from the raw data above and below are utilized, with normalized z distances w_s and $w_t = 1 - w_s$ relative to the target z slice. The interpolation process involves shifting within the planes by a specified distance in the x direction ($w_s d_z \cos\theta$ and $w_t d_z \cos\theta$, respectively) and subsequently performing linear interpolation across these two interpolated slices along the z-axis (with weights $1 - w_s$ and $1 - w_t$). Subsequently, the interpolated image will serve as input for the integrated processes of deskewing, rotation, and resampling. Because the image warp function permits the specification of the output view, we have also incorporated a cropping feature by providing a bounding box, allowing for saving specific regions, i.e., skipping empty regions.

In the objective scan mode, where the stage moves along the z-axis, the deskew operation is not needed. Nevertheless, the above processing can still be applied, with the modification of substituting S_{ds} with an identity matrix.

Deconvolution

Richardson-Lucy (RL) deconvolution has the form of

$$x^{(k+1)} = \left(\frac{I}{f \circledast x^{(k)}} \circledast b^T\right) x^{(k)}$$

where I is the observation, f is the forward projector (the point spread function (PSF)), b is the backward projector, $\circledast$ denotes the convolution operator, and $x^{(k)}$ is the deconvolution result in k-th iteration. In conventional RL deconvolution, $b = f$. In the OTF masked Wiener (OMW) method, the backward projector is generated with these steps:

1. The optical transfer function (OTF) H of the PSF f is computed, $H = \mathscr{F}(f)$.
2. The OTF mask for the OTF support is segmented by applying a threshold to the amplitude H. The threshold value is determined by a specified percentile (90% by default) of the accumulated sum of sorted values in H from high to low.
3. After segmentation, the OTF mask undergoes a smoothing process, retaining only the central object, followed by convex hull filling. For deskewed space deconvolution, the three major components are kept after object smoothing and concatenated into a unified object along the z-axis, followed by convex hull filling.
4. Then, the relative distance matrix D is computed with the image center as 0, and the edge of the support as 1 with the star distance from the center to the border of the whole image.
5. The distance matrix D is used to calculate the weight matrix W with the Hann window function for apodization, as expressed by the following formula:
$$w(x) = \begin{cases} 1 & x \leq l \\ cos^2(\frac{\pi(x-l)}{2(u-l)}) & l < x \leq u \\ 0 & x > u \end{cases}$$
Where l and u are the lower and upper bounds for the relative distances. By default, $l = 0.8$ and $u = 1$ (edge of the support). For skewed space deconvolution, the weight matrix will be recovered as a single distance matrix by adding the distance matrix from the corresponding three components together.
6. Calculate Wiener filter $F = \frac{H^T}{\|H\|^2+\alpha}$, where α is the Wiener parameter.
7. The backward projection in the Fourier space is expressed as $B = W \odot F$, where $\odot$ denotes the Hadamard product operator (element-wise multiplication), and the backward projector in the real space is $b = \mathscr{F}^{-1}(B)$.

The Fourier Shell Correlation (FSC) method [23] is used to determine the optimal number of conventional RL iterations and the optimal Wiener parameter in the OMW method. Here, the central region of the volume (i.e., $202 \times 202 \times 202$) is employed to compute the relative resolution. By default, the FSC is calculated with a radius of 10 pixels and an angle interval of $\pi/12$. Cut-off frequencies for relative resolution are determined using one-bit thresholding [38]. The relative resolution across iterations (or different Wiener parameters) is plotted. The optimal number of RL iterations or the Wiener parameter is defined by the value closest to 1.01 times the minimum of the curve beyond the point where the curve reaches its minimum value.

Stitching

The stitching process requires a CSV meta-file documenting file names and corresponding coordinates. The pipeline consists of three major steps: Tiff to Zarr conversion (or preprocessing), cross-correlation registration, and parallel

block stitching (fusion). The overall stitching workflow is governed by a master job using the generic computing framework. If the input tiles are in Tiff format or need additional processing before stitching, the initial step involves converting them from Tiff to Zarr (or processing the existing Zarr file). In this step, the master job distributes tasks to individual worker jobs, assigning one worker for each tile. A worker reads the data using the Cpp-tiff reader (Cpp-Zarr reader if in Zarr format), performs optional processing such as flipping, cropping, flat field correction, edge erosion, or other user-defined operations, and then writes the processed data using the Cpp-Zarr writer.

Following conversion, stitching can be executed directly using the input tile coordinates. Alternatively, normalized cross-correlation registration [24] can be employed to refine and optimize the coordinates before stitching. The master job utilizes coordinate information and tile indices to establish tile grids and identify neighboring tiles with overlaps. Cross-correlation registration is performed for overlapping tiles that are direct neighbors, characterized by a one-axis difference in tile indices, where the other two axes remain constant (e.g., $[1, 2, 3]$ and $[1, 2, 4]$). Again, the master job will assign the tasks to worker jobs. To optimize computing time and memory usage, only the overlap region for a tile (template) is loaded, along with the overlap region of another tile (source) incorporating a buffer size defined by the maximum allowed shifts in the xy and z axes. We also have the option to downsample the regions to allow faster cross-correlation computing. The optimal shift between the two tiles is identified as the one exhibiting the maximum correlation within the allowable shift limits. After completing the cross-correlation computation for all pairs of overlapping tiles (direct neighbors), the next step is to obtain the absolute shifts. We have two main methods for deciding the shifts: the local method, and the global method. The local method is based on the minimum spanning tree, where edges between overlapping tiles are pruned to form a tree based on the correlation values from high to low, followed by registration with the pairwise optimal shifts. For the global method, the approach involves inferring the optimal absolute shift based on the relative shifts between pairs through a nonlinear constrained optimization process with the problem below:

$$\min_{x} \sum_{i,j} w_{ij} \|x_i - x_j - d_{ij}\|_2^2$$

$$s.t. \quad l < x_i - x_j < u$$

where $\mathbf{x}_{shift} = x_1, ..., x_n$ are the absolute shifts for the tiles. d_{ij} is the relative shift between tile i and j, w_{ij} is the weight between tile i and j (here we use the squares of max cross-correlation values). l and u are the lower and upper bounds for the maximum allowable shift. The goal is to position all tiles at optimal coordinates by minimizing the weighted sum of the squared differences between their distances and the relative shifts while ensuring that their distances adhere to the specified maximum allowable shifts.

For images collected by subregions (batches) of the same sample that have different tile grids, we employ the global method for tiles within each subregion. Subsequently, the subregions are treated as super nodes, and a nonlinear constrained optimization is applied to those nodes, by minimizing the sum of squared differences of the centroid distances to the averaged shift distances.

$$\min_{x} \sum_{i,j} \|x_{ri} - x_{rj} - d_{r,ij}\|_2^2$$

$$s.t. \quad l_r < x_{ri} - x_{rj} < u_r$$

where x_{ri} and x_{rj} are the centroid coordinates for subregions i and j, l_r and u_r are lower and upper bounds for the maximum allowable shifts across subregions. The averaged shift distance, denoted as $d_{r,ij}$, is determined through a weighted average calculation based on the absolute shifts across subregions, as expressed below

$$d_{r,ij} = \frac{\sum_{m \in S_i, n \in S_j} w_{mn} d_{mn}}{\sum_{m \in S_i, n \in S_j} w_{mn}}$$

where w_{mn} is the cross-correlation value at the optimal shift between tiles $m \in S_i$, and $n \in S_j$, and S_k denotes the set of tiles in subregion k. Once the optimal shifts for the subregions are obtained, the last step is to reconstruct the optimal shifts for the tiles within the subregions by applying the optimal shifts of the centroids of the subregions to the coordinates of the tiles. After acquiring the absolute shifts, we can apply these shifts to the coordinates to get the final positions for the tiles.

After registration, the master job determines the final stitched image size and the specific locations to place the tiles. To facilitate parallel stitching, the process is executed region by region in a non-overlapping manner (with the same size except for the borders). These regions are subsequently processed and saved directly as one or more distinct chunk files in Zarr format. For each region, the system stores information about the relevant tiles and their corresponding bounding boxes within those tiles. The master job will submit stitching tasks to worker jobs, and the worker will load the information and start the stitching for the region. If the chunk comes from one tile, it directly reads the region in that tile and saves it to the corresponding chunk in the stitched image. If the region spans multiple tiles, merging the overlapping regions into a single cohesive region is necessary. Several blending options are available for this purpose, including "none", "mean", "median", "max" and "feather" methods. For the "none" option, half of the overlap region is taken from each tile. For the "mean", "median", and "max" options, the voxel values in the stitched region are calculated as the mean, median, and maximum values from the corresponding voxels in the overlapping regions of the involved tiles, respectively. The feather blending involves calculating the weighted average across the tiles [39]. The weights are the power of the distance transform of

the tiles as follows:

$$w_{i,m} = d^{\alpha}_{i,m}/(d^{\alpha}_{i,m} + d^{\alpha}_{j,n}), \quad \text{and} \quad w_{j,n} = d^{\alpha}_{j,n}/(d^{\alpha}_{i,m} + d^{\alpha}_{j,n})$$

$$I_{s,l} = w_{i,m} I_{i,m} + w_{j,n} I_{j,n}$$

where $d_{i,m}$, $d_{j,n}$ are distance transform for the voxel m in tile i and voxel n j, α is the order (10 by default), I_i and I_j are the intensities from image I_i and I_j, and I_s is the stitched image. Here we assume voxel m in tile i and voxel n in tile j are fused to voxel l in the stitched image. For the distance transform, we utilize a weighted approach, conducting the distance transform across z slices and applying the Tukey window function to address the anisotropic properties during stitching. When all tiles share the same size, we compute a weight matrix for a single tile and apply it across all tiles in the stitching process to optimize computing time. The final stitched image is obtained once all the chunks are processed.

Large-scale processing

In the context of stitching, we previously discussed the approach of utilizing small tiles where intermediate steps can fit into memory. Importantly, the framework remains effective regardless of the total number of these tiles. For stitching involving large tiles where intermediate steps exceed memory capacities, such as those acquired during imaging or stitched subregions, challenges arise in registration and distance transformation for feather blending, requiring the loading of large regions or entire tiles into memory. To resolve the issue, we use the MIP slabs for the registration and distance transform. These MIP slabs are computed across all three axes with given downsampling factors. We define a downsampling factor as $[M_x, M_y, M_z, m_x, m_y, m_z]$, where M_x, M_y, M_z are downsampling factors for the major axis and m_x, m_y, m_z are for the minor axis. When generating the MIP slab for a specific axis, i.e., x-axis, the MIP slab is computed with the major downsampling factor for the x axis, and minor downsampling factor for the yz axes, specifically, $[M_x, m_y, m_z]$. The MIP slabs for the y and z axes follow the same rule with downsampling factors $[m_x, M_y, m_z]$ and $[m_x, m_y, M_z]$, respectively. To enrich the signal, especially for spare samples, we use max pooling for the downsampling, that is, taking the max value in the neighborhood for the downsampling. It also supports initial data resampling using linear interpolation to smooth the data. For the registration, normalized cross-correlation calculations involve all three MIP slabs, generating three sets of optimal shifts. The optimal shifts from the minor axes are then averaged to obtain the final optimal shifts with weights assigned based on the squares of the cross-correlation values. For the distance transform, only the MIP slab along the z-axis (major axis) is employed to calculate the distance transform, followed by computing the weights for feather blending. In the stitching process, for overlapping regions, the corresponding downsampled weight regions are loaded and upsampled using linear interpolation to match the size of the regions in the stitching. The upsampled weights

are then utilized for feather blending, following the same approach as employed for stitching with smaller tiles.

For large-scale deskew and rotation, tasks are split across the y-axis based on the size in the x and z axes, with an additional buffer of one or two pixels on both sides in the y-axis. These tasks are then allocated to worker jobs for processing, with the results saved as independent Zarr regions on disk. Furthermore, as mentioned, MIP masks can be used to define a tight boundary for the object to optimize efficiency in data reading, processing, and writing. For input purposes, the MIP masks delineate the object's boundaries (with an additional border buffer), specifying the actual region for deskewing and rotation. Moreover, we perform deskewing and rotation for the MIP along the y-axis to define the bounding box for the output in the xz axes. The geometric transformation function directly relies on this bounding box to determine the output view to minimize the empty regions, thereby further optimizing processing time, memory, and storage requirements.

For large-scale deconvolution, tasks are distributed across all three axes, ensuring that regions occupy entire chunk files. An additional buffer size, set to at least half of the PSF size (plus some extra size, 10 by default), is included to eliminate edge artifacts. Moreover, as previously stated, MIP masks can be utilized to define a tight boundary for the object during the deconvolution process. The three MIPs corresponding to the designated region in a task are retrieved and examined for any instances of empty MIPs. If an empty MIP is identified, deconvolution is skipped, yielding an output of all zeros for the region.

Image processing and simulations

The images were processed using LLSM5DTools, including several essential steps like stitching, deconvolution, combined deskew and rotation, and additional steps, such as resampling and cropping, all integrated into the LLSM5DTools software. Flat field correction was applied for 11 hr cell data, phase contrast data, and VNC data with either experimentally collected flat fields or estimated ones based on the data. For estimation, the BaSiC tool was employed [40].

The benchmark images for readers and writers, deskew/rotation, and deconvolution (timing) were generated by cropping or replicating frames from a uint16 image sized at $1,800 \times 512 \times 3,400$ to specific numbers of frames. The stripped line patterns used to compare deconvolution methods were simulated using the methodology outlined in [19]. Large-scale stitching benchmarks, ranging from 1 TB to 1 PB were conducted based on the VNC dataset with 1,071 tiles (one channel), each sized at $1,800 \times 320 \times 17,001$. Specifically, for the stitching task with sizes of 1 TB and 10 TB, parts of the dataset with 57 and 562 tiles were included, respectively. For larger sizes from 100 TB to 1 PB, the tiles were replicated across all three axes, resulting in 5,644 and 57,392 tiles, respectively. For large-scale deconvolution benchmarks, the stitched VNC dataset ($28,478 \times 15,612 \times 21,299$, uint16) was

either cropped or replicated in all three axes to produce datasets with size $11,027 \times 6,046 \times 8,248$, $23,757 \times 13,024 \times 17,769$, $51,183 \times 28,059 \times 38,280$, and $111,145 \times 60,931 \times 83,127$ for 1TB, 10TB, 100 TB and 1 PB, respectively. For large-scale deskew/rotation benchmarks, datasets (uint16) with sizes $34,367 \times 2,722 \times 5,879$, $74,041 \times 5,864 \times 12,665$, $159,516 \times 12,632 \times 27,284$, and $346,393 \times 27,431 \times 59,248$ were generated for 1TB, 10TB, 100 TB and 1 PB, respectively.

Computing infrastructures

In the lab, there is a computing cluster consisting of 38 CPU/GPU computing nodes with 27 CPU nodes (24 nodes with dual Intel Xeon gold 6146 CPUs, 4 nodes with dual Intel Xeon gold 6342 CPU), 8 GPU nodes (3 nodes with dual Intel Xeon Silver 4210R and 4 Titan V GPUs, 4 nodes with dual Intel Xeon Gold 6144 and 4 A100 GPUs, 1 NVidia DGX box with dual AMD Rome 7742 and 8 A100 GPUs). The CPU and GPU nodes have 512 GB RAM on each node and the DGX box has 2 TB RAM. We have several SSD data servers, including a 70 TB, two 300 TB, and a 400 TB parallel file system. We also have access to the Perlmutter supercomputer from NERSC, with both CPU and GPU nodes. For each CPU node, there are dual AMD EPYC 7713 CPUs with 512 GB RAM; for each GPU node, there is a single AMD EPYC 7713 CPU with 4 A100 GPUs and 512 GB RAM.

Microscopy setup

The imaging experiments were performed using homemade lattice light sheet microscopy systems with a comparable setup to a prior system [3]. Two lasers, 488 nm and 560 nm (500 mW, MPB Communications 2RU-VFL-P-500-488-B1R, and 2RU-VFL-P-1000-560-B1R), were employed as the light sources. The water-dipped excitation objective (EO, Thorlabs TL20X-MPL) and detection objective (DO, Zeiss, 20X, 1.0 NA, 1.8 mm FWD, 421452-9800-000) were used for the imaging.

Cell culture and imaging

Pig kidney epithelial cells (LLC-PK1, a gift from M. Davidson at Florida State University) cells and HeLa cells were cultured in DMEM with GlutaMAX (Gibco, 10566016) supplemented with 10% fetal bovine serum (FBS; Seradigm) in the incubator with 5% CO_2 at 37°C with 100% humidity. For LLC-PK1 cell imaging, cells stably expressing the ER marker mEmerald-Calnexin and the chromosome marker mCherry-H2B were grown on the coverslips (Thorlabs, CG15XH) coated with 200 nm diameter fluorescent beads (Invitrogen FluoSpheresTM Carboxylate-Modified Microspheres, 505/515 nm, F8811) in a 6-well plate. When the cells reached $30 - 80\%$ confluency, the coverslip was loaded onto the microscopy stage at 37°C in Leibovitz's L-15 Medium without Phenol Red (Gibco catalog # 21-083-027), with 5% fetal bovine serum

(ATCC SCRR-30- 2020™), and an antibiotic cocktail consisting of 0.1% Ampicillin (ThermoFisher 611770250), 0.1% Kanamycin (ThermoFisher, 11815024) and 0.1% Penicillin/Streptomycin (ThermoFisher, 15070063) for the imaging. For Hela cell imaging, the cells were cultivated on the 25 mm coverslip in a 6-well plate until approximately 50% confluency was achieved. Then the cells were imaged at 37°C in Leibovitz's L-15 Medium without Phenol Red (Gibco catalog # 21-083-027), with 5% fetal bovine serum (ATCC SCRR-30- 2020™).

Mouse brain sample preparation and imaging

All mice experiments conducted at Janelia Research Campus, Howard Hughes Medical Institute (HHMI), were performed in accordance with the US National Institutes of Health Guide for the Care and Use of Laboratory Animals. Procedures and protocols were approved by the Institutional Animal Care and Use Committee of the Janelia Research Campus, HHMI.

Transgenic Thy1-YFPH mice 8 weeks or older with cytosolic expression of yellow fluorescent protein (YFP) at high levels in motor, sensory, and subsets of central neurons were anesthetized with isoflurane ($1-2\%$ by volume in oxygen) and placed on a heated blanket. An incision was made on the scalp followed by removing of the exposed skull. A cranial window made of a single 170-um-thick coverslip (no.1) was embedded in the craniotomy. The cranial window and a headbar were sealed in place with dental cement for subsequent imaging.

Fly VNC sample preparation and imaging

The genetically modified strain of fruit flies (Drosophila melanogaster) was raised on a standard cornmeal-agar-based medium in a controlled environment of 25°C on a 12:12 light/dark cycle. On the day of eclosion, female flies were collected, and group housed for 4-6 days. The genotype was VGlut^{MI04979}-LexA:QFAD/MN-GAL4 (attp40); 13XLexAop-Syn21-mScarle [JK65C], 20XUAS-Syn21-GFP [attp2]/MN-GAL4 [attp2] [41, 42]. Dissection and immunohistochemistry of fly VNC were performed following the protocol in [43] with minor modifications. The primary antibodies were chicken anti-GFP (1:1000, Abcam, ab13970) and rabbit anti-dsRed (1:1000, Takara Bio, 632496). The secondary antibodies were goat anti-chicken IgY Alexa Fluor 488 (1:500, Invitrogen, A11039) and goat anti-rabbit IgG Alexa Fluor 568 (1:500, Invitrogen, A11011). Expanded VNC samples were prepared for 8X expansion exactly as described in [43]. The imaging protocol for the expanded VNC sample was nearly identical to the method described in [4].

Visualization and software

The movies were made with Imaris (Oxford Instruments), Fiji, Amira (Fisher Scientific), Nvidia IndeX (Nvidia), and MATLAB R2023a (MathWork) software. The image data were processed, and figures were made with MATLAB R2023a (MathWork). Python (3.8.8) with Zarr-Python (2.16.1) and tifffile

(2023.7.10) were used for the benchmarks for image readers and writers. Spark version of BigStitcher (BigStitcher-Spark) and stitching-spark were used for the stitching comparison.

Acknowledgments

We would like to thank Alexander Kuhn, Marc Nienhaus, and other members of the Nvidia IndeX team for sharing the Nvidia Index software and helping set up the visualization workflows. We would like to thank Allen Institute for making their exaSPIM datasets publicly available [26]. This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility located at Lawrence Berkeley National Laboratory, operated under Contract No. DE-AC02-05CH11231 using NERSC award DDR-ERCAP0025501.

Funding

X.R., G.L., F.G., and S.U. are partially funded by the Philomathia Foundation. X.R. and G.L. are partially funded by the Chan Zuckerberg Initiative. X.R. and S.U. are supported by Lawrence Berkeley National Lab's LDRD. M.M., T.F., D.M., J.L., and E.B. are funded by HHMI. F.G. is partially funded by the Feodor Lynen Research Fellowship, Humboldt Foundation. E.B. is an HHMI Investigator. S.U. is funded by the Chan Zuckerberg Initiative Imaging Scientist program. S.U. is a Chan Zuckerberg Biohub – San Francisco Investigator.

Conflict of interest

There is no conflict of interest.

Authors Contributions

E.B. and S.U. supervised the project and acquired the funding. X.R. wrote the manuscript with input from all coauthors. X.R. designed the algorithms and implemented the software. M.M. implemented the fast image readers and writers under X.R.'s guidance. M.M. designed and implemented the graphical user interfaces. J.L. prepared the VNC sample and G.L. performed the imaging experiments. A.K. prepared the cultured LLC-PK1 cells and F.G. performed the live cell imaging experiment. T.F. performed the imaging experiments for the live mouse brain imaging and phase imaging of HeLa cells. D.M. developed the microscopy acquisition software for the imaging experiments. X.R. performed all the image processing and analysis and made the figures. X.R. and M.M. made the movies with S.U.'s input.

Data availability

The datasets used in this manuscript will be shared upon reasonable request.

Code availability

The source code of the software is publically available here: LLSM5DTools. The GUI for the software can be downloaded from here: LLSM Processing GUI.

References

[1] Stelzer, E.H., Strobl, F., Chang, B.-J., Preusser, F., Preibisch, S., McDole, K., Fiolka, R.: Light sheet fluorescence microscopy. Nature Reviews Methods Primers **1**(1), 73 (2021)

[2] Chen, B.-C., Legant, W.R., Wang, K., Shao, L., Milkie, D.E., Davidson, M.W., Janetopoulos, C., Wu, X.S., Hammer III, J.A., Liu, Z., *et al.*: Lattice light-sheet microscopy: imaging molecules to embryos at high spatiotemporal resolution. Science **346**(6208), 1257998 (2014)

[3] Liu, T.-L., Upadhyayula, S., Milkie, D.E., Singh, V., Wang, K., Swinburne, I.A., Mosaliganti, K.R., Collins, Z.M., Hiscock, T.W., Shea, J., *et al.*: Observing the cell in its native state: Imaging subcellular dynamics in multicellular organisms. Science **360**(6386), 1392 (2018)

[4] Gao, R., Asano, S.M., Upadhyayula, S., Pisarev, I., Milkie, D.E., Liu, T.-L., Singh, V., Graves, A., Huynh, G.H., Zhao, Y., *et al.*: Cortical column and whole-brain imaging with molecular contrast and nanoscale resolution. Science **363**(6424), 8302 (2019)

[5] Aguet, F., Antonescu, C.N., Mettlen, M., Schmid, S.L., Danuser, G.: Advances in analysis of low signal-to-noise images link dynamin and ap2 to the functions of an endocytic checkpoint. Developmental cell **26**(3), 279–291 (2013)

[6] Lamb, J.R., Ward, E.N., Kaminski, C.F.: Open-source software package for on-the-fly deskewing and live viewing of volumetric lightsheet microscopy data. Biomedical Optics Express **14**(2), 834–845 (2023)

[7] Schmid, B., Huisken, J.: Real-time multi-view deconvolution. Bioinformatics **31**(20), 3398–3400 (2015)

[8] Guo, M., Li, Y., Su, Y., Lambert, T., Nogare, D.D., Moyle, M.W., Duncan, L.H., Ikegami, R., Santella, A., Rey-Suarez, I., *et al.*: Rapid image deconvolution and multiview fusion for optical microscopy. Nature biotechnology **38**(11), 1337–1346 (2020)

[9] Hörl, D., Rojas Rusak, F., Preusser, F., Tillberg, P., Randel, N., Chhetri, R.K., Cardona, A., Keller, P.J., Harz, H., Leonhardt, H., *et al.*: Bigstitcher: reconstructing high-resolution image datasets of cleared and

expanded samples. Nature methods **16**(9), 870–874 (2019)

[10] Schmid, B., Schindelin, J., Cardona, A., Longair, M., Heisenberg, M.: A high-level 3d visualization api for java and imagej. BMC bioinformatics **11**(1), 1–7 (2010)

[11] Campagnola, L., Klein, A., Larson, E., Rossant, C., Rougier, N.P.: Vispy: harnessing the gpu for fast, high-level visualization. In: Proceedings of the 14th Python in Science Conference (2015)

[12] Aguet, F., Upadhyayula, S., Gaudin, R., Chou, Y.-y., Cocucci, E., He, K., Chen, B.-C., Mosaliganti, K., Pasham, M., Skillern, W., *et al.*: Membrane dynamics of dividing cells imaged by lattice light-sheet microscopy. Molecular biology of the cell **27**(22), 3418–3435 (2016)

[13] Miles, A., jakirkham, Bussonnier, M., Moore, J., Orfanos, D.P., Bourbeau, J., Fulton, A., Bennett, D., Lee, G., Patel, Z., Abernathey, R., Kristensen, M.R.B., Rocklin, M., AWA, A.B., Chopra, S., de Andrade, E.S., Hamman, J., Durant, M., Schut, V., raphael dussin, Verma, S., Chaudhary, S., Barnes, C., Nunez-Iglesias, J., Sansal, A., Williams, B., Mohar, B., Noyes, C.: Zarr-developers/zarr-python: V2.16.1. https://doi.org/10.5281/zenodo.8263439. https://doi.org/10.5281/zenodo.8263439

[14] Zaharia, M., Xin, R.S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M.J., *et al.*: Apache spark: a unified engine for big data processing. Communications of the ACM **59**(11), 56–65 (2016)

[15] Rocklin, M., *et al.*: Dask: Parallel computation with blocked algorithms and task scheduling. (2015)

[16] Saalfeld, S., Pisarev, I., Hanslovsky, P., Champion, A., Rueden, C., Bogovic, J., Kittisopikul, M., jakirkham: Saalfeldlab/n5: N5-2.5.1. https://doi.org/10.5281/zenodo.6578232. https://doi.org/10.5281/zenodo.6578232

[17] Moore, J., Allan, C., Besson, S., Burel, J.-M., Diel, E., Gault, D., Kozlowski, K., Lindner, D., Linkert, M., Manz, T., *et al.*: Ome-ngff: a next-generation file format for expanding bioimaging data-access strategies. Nature methods **18**(12), 1496–1498 (2021)

[18] TensorStore developers: TensorStore: Library for Reading and Writing Large Multi-dimensional Arrays. https://github.com/google/tensorstore

[19] Liu, G., Ruan, X., Milkie, D.E., Görlitz, F., Mueller, M., Hercule, W., Killilea, A., Betzig, E., Upadhyayula, S.: Characterization, comparison, and optimization of lattice light sheets. Science Advances **9**(13), 6623

(2023) https://www.science.org/doi/pdf/10.1126/sciadv.ade6623. https://doi.org/10.1126/sciadv.ade6623

[20] Richardson, W.H.: Bayesian-based iterative method of image restoration. JoSA **62**(1), 55–59 (1972)

[21] Lucy, L.B.: An iterative technique for the rectification of observed distributions. The astronomical journal **79**, 745 (1974)

[22] Zeng, G.L., Gullberg, G.T.: Unmatched projector/backprojector pairs in an iterative reconstruction algorithm. IEEE transactions on medical imaging **19**(5), 548–555 (2000)

[23] Koho, S., Tortarolo, G., Castello, M., Deguchi, T., Diaspro, A., Vicidomini, G.: Fourier ring correlation simplifies image restoration in fluorescence microscopy. Nature communications **10**(1), 3103 (2019)

[24] Briechle, K., Hanebeck, U.D.: Template matching using fast normalized cross correlation. In: Optical Pattern Recognition XII, vol. 4387, pp. 95–102 (2001). SPIE

[25] NVIDIA IndeX developers: NVIDIA IndeX: 3D Scientific Data Visualization. https://developer.nvidia.com/index

[26] Glaser, A., Chandrashekar, J., Vasquez, J., Arshadi, C., Ouellette, N., Jiang, X., Baka, J., Kovacs, G., Woodard, M., Seshamani, S., Cao, K., Clack, N., Recknagel, A., Grim, A., Balaram, P., Turschak, E., Liddell, A., Rohde, J., Hellevik, A., Takasaki, K., Barner, L.E., Logsdon, M., Chronopoulos, C., de Vries, S., Ting, J., Perlmutter, S., Kalmbach, B., Dembrow, N., Reid, R.C., Feng, D., Svoboda, K.: Expansion-assisted selective plane illumination microscopy for nanoscale imaging of centimeter-scale tissues (2023). https://doi.org/10.7554/elife.91979.1

[27] Wang, Y., Jeon, H.: 3d cell cultures toward quantitative high-throughput drug screening. Trends in Pharmacological Sciences (2022)

[28] Wan, Y., McDole, K., Keller, P.J.: Light-sheet microscopy and its potential for understanding developmental processes. Annual review of cell and developmental biology **35**, 655–681 (2019)

[29] Ståhl, P.L., Salmén, F., Vickovic, S., Lundmark, A., Navarro, J.F., Magnusson, J., Giacomello, S., Asp, M., Westholm, J.O., Huss, M., *et al.*: Visualization and analysis of gene expression in tissue sections by spatial transcriptomics. Science **353**(6294), 78–82 (2016)

[30] Liu, Z., Jin, L., Chen, J., Fang, Q., Ablameyko, S., Yin, Z., Xu, Y.: A survey on applications of deep learning in microscopy image analysis.

Computers in biology and medicine **134**, 104523 (2021)

[31] Melanthota, S.K., Gopal, D., Chakrabarti, S., Kashyap, A.A., Radhakrishnan, R., Mazumder, N.: Deep learning-based image processing in optical microscopy. Biophysical Reviews **14**(2), 463–481 (2022)

[32] Volpe, G., Wählby, C., Tian, L., Hecht, M., Yakimovich, A., Monakhova, K., Waller, L., Sbalzarini, I.F., Metzler, C.A., Xie, M., et al.: Roadmap on deep learning for microscopy. ArXiv (2023)

[33] Li, Y., Su, Y., Guo, M., Han, X., Liu, J., Vishwasrao, H.D., Li, X., Christensen, R., Sengupta, T., Moyle, M.W., *et al.*: Incorporating the image formation process into deep learning improves network performance. Nature Methods **19**(11), 1427–1437 (2022)

[34] Laine, R.F., Jacquemet, G., Krull, A.: Imaging in focus: an introduction to denoising bioimages in the era of deep learning. The international journal of biochemistry & cell biology **140**, 106077 (2021)

[35] Fu, Y., Lei, Y., Wang, T., Curran, W.J., Liu, T., Yang, X.: Deep learning in medical image registration: a review. Physics in Medicine & Biology **65**(20), 20–01 (2020)

[36] Li, J., Wang, Z., Lai, S., Zhai, Y., Zhang, M.: Parallax-tolerant image stitching based on robust elastic warping. IEEE Transactions on multimedia **20**(7), 1672–1687 (2017)

[37] Team, B.D.: Blosc: A blocking, shuffling and lossless compression library. https://github.com/Blosc/c-blosc

[38] van Heel, M., Schatz, M.: Fourier shell correlation threshold criteria. Journal of Structural Biology **151**(3), 250–262 (2005). https://doi.org/10.1016/j.jsb.2005.05.009

[39] Uyttendaele, M., Eden, A., Skeliski, R.: Eliminating ghosting and exposure artifacts in image mosaics. In: Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001, vol. 2, p. (2001). IEEE

[40] Peng, T., Thorn, K., Schroeder, T., Wang, L., Theis, F.J., Marr, C., Navab, N.: A basic tool for background and shading correction of optical microscopy images. Nature communications **8**(1), 14836 (2017)

[41] Diao, F., Ironfield, H., Luan, H., Diao, F., Shropshire, W.C., Ewer, J., Marr, E., Potter, C.J., Landgraf, M., White, B.H.: Plug-and-play genetic access to drosophila cell types using exchangeable exon cassettes. Cell reports **10**(8), 1410–1421 (2015)

[42] Pfeiffer, B.D., Truman, J.W., Rubin, G.M.: Using translational enhancers to increase transgene expression in drosophila. Proceedings of the National Academy of Sciences **109**(17), 6626–6631 (2012)

[43] Lillvis, J.L., Otsuna, H., Ding, X., Pisarev, I., Kawase, T., Colonell, J., Rokicki, K., Goina, C., Gao, R., Hu, A., *et al.*: Rapid reconstruction of neural circuits using tissue expansion and light sheet microscopy. Elife **11**, 81248 (2022)

Figure legends

Fig. 1: Overall design of the image processing frameworks. **a**, image acquisition and processing workflows. **b**, illustration of distributed generic computing framework. **c**, illustration of distributed processing of many independent files that can be processed by a single worker. **d**, illustration of distributed processing of the split-process-merge mechanism for the distributed processing of a large image file. **e**, overall functionalities and features in LLSM5DTools.

Fig. 2: Fast Tiff and Zarr readers and writers. **a**, acceleration folds of Cpp-tiff reader versus the conventional Tiff reader in MATLAB and tifffile reader in Python with different numbers of frames in the 3D stack. **b**, acceleration folds of Cpp-tiff writer versus the conventional tiff writer in MATLAB and tifffile writer in Python with different numbers of frames in the 3D stack. **a**, acceleration folds of Cpp-Zarr reader versus the conventional Zarr reader (MATLAB interface of Zarr) and native Zarr in Python with different numbers of frames in the 3D stack. **d**, acceleration folds of Cpp-Zarr writers versus the conventional Zarr writer (MATLAB interface of Zarr) and native Zarr in Python with different numbers of frames in the 3D stack. The frame sizes in the x and y axes are 512 and 1800, respectively. The benchmarks were run independently ten times on a 24-core CPU computing node.

Fig. 3: Combined deskew and rotation. **a**, scheme of conventional separate deskew and rotation processes. **b**, orthogonal views of raw (skewed), deskewed (DS), and deskew/rotated (DSR) images for cultured cells with xy pixel size $0.108\,\mu m$ and scan step size $0.3\,\mu m$. **c**, the plot for the proposed data size for raw, deskewed, deskew/rotated data across different numbers of frames per stack. x and y are fixed as 512 and 1800 pixels, respectively. The dashed line indicates the memory limit (500 GB) in a computing node. **d**, scheme of combined deskew and rotation for small (top) and large (bottom) z scan step sizes. **e**, the difference between separate and combined deskew/rotation for the same image in **b**. **f**, acceleration folds for separate, combined direct, and combined interpolated deskew/rotation across different numbers of frames per stack, from 500 to 5,000. This comparison only contains deskew and rotation processes, excluding the reading and writing time of the images. **g**, memory efficiencies for separate, combined direct, and combined interpolated deskew/rotation across different numbers of z slices, from 500 to 5,000. **h**, acceleration folds for separate, combined direct, and combined interpolated deskew/rotation across different numbers of z slices, from 500 to 2,000. This comparison contains deskew and rotation processes with the reading and writing of the images using the conventional and Cpp-tiff reader and writer.

Fig. 4: Fast RL deconvolution. **a**, Left: theoretical xy and xz PSFs (top, intensity $\gamma = 0.5$, scale bar: $1\,\mu m$) and OTFs (bottom, log-scale, scale bar: $2\,\mu m^{-1}$) for the multi-Bessel (MB) light sheet with NA 0.43 and NA annulus 0.40 - 0.47. Blue: theoretical support; orange and yellow: theoretical maximum (orange)

and experimental (yellow) envelopes for the BW method; magenta: experimental envelope for the OMW method. Right: illustration of deconvolution with the simulated strip pattern. The raw and deconvolved images with traditional, BW, and OMW methods are on the right, and the corresponding line cuts are on the left. The orange lines indicate the theoretical line locations and the blue lines are for the actual intensities along the line cuts. **b**, same illustration of theoretical PSFs and OTFs and deconvolution of simulated strip pattern for Sinc light sheet (NA 0.32, $\sigma_{NA} = 5.0$). **c-d**, illustration of backward projectors (top, scale bar: $1\,\mu m$), the Fourier spectrums of backward projectors (middle, intensity $\gamma = 0.5$), and the products with forward projectors in Fourier spaces (bottom, intensity $\gamma = 0.5$, scale bar: $1\,\mu m^{-1}$) for the MB light sheet (**c**) and Sinc light sheet (**d**). **e-f**, orthogonal views of cell images for raw, conventional RL decon, BW, and OMW methods for the MB light sheet (**e**) and Sinc light sheet (**f**), the iteration numbers are in the titles (scale bar: $2\,\mu m$). The Fourier spectrums are at the bottom of the corresponding images (intensity $\gamma = 0.5$, scale bar: $1\,\mu m^{-1}$). **g-h**, deconvolution time comparison between traditional and OMW methods on CPU (**g**) and GPU (**h**) (only the deconvolution step is timed).

Fig. 5: Zarr-based distributed stitching framework. **a**, scheme illustration of the stitching steps. **b**, scheme illustration of registration methods: local, grid, and grouped. Boxes indicate tiles and green/blue lines indicate pairs of tiles involved in registration. **c**, illustration of raw 2D phase tiles before processing. **d**, illustration of processed phase contrast image with stitching and deconvolution. Boxes with S8a and S8b are for the cells illustrated in Fig S8a and S8b, respectively. **e**, illustration of the fine features of a cell in zoomed views from **d**. **f**, overall visualization of the long-term large field of view imaging of cultured LLC-PK1 cells at time 0. Blue: H2B-Cherry; glow: Connexin-Emerald. Intensity $\gamma = 0.5$. **g**, illustration of cropped regions from **e**. **h**, illustration of the time-lapse of three-way division cell from **e**.

Fig. 6: Large-scale processing. **a**, scheme illustration of large-scale processing for stitching, deconvolution, and deskew/rotation. For deconvolution, the data is split over all three axes with a border size of slightly over half of the PSF size. For deskew and rotation, the data is split along the y-axis with a buffer size of one slice. **b**, total run times for large-scale stitching of a single volume with size ranging from 1 TB to 1 PB. **c**, total run times for large-scale deconvolution of a single volume with size ranging from 1 TB to 1 PB. **d**, total run times for large-scale deskew and rotation of a single volume with size ranging from 1 TB to 1 PB. **e**, visualization of whole fly VNC with 8× expansion. Cyan: VGlut^{MI04979}-LexA::QFAD, and purple: MN-GAL4. Intensity $\gamma = 0.5$. **f-i**, illustration of cropped regions from **e**. Intensity $\gamma = 0.75$ for all four regions.

Figures

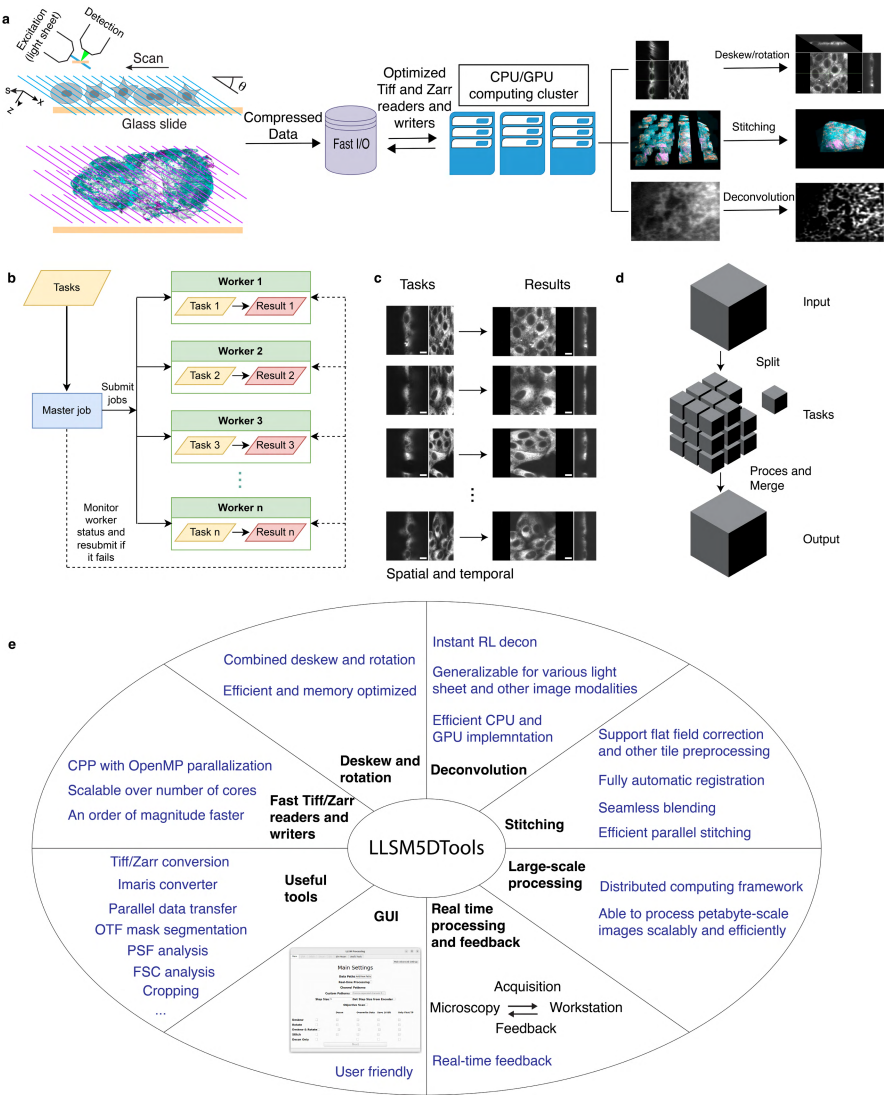

Fig. 1: Overall design of the image processing frameworks. **a**, image acquisition and processing workflows. **b**, illustration of distributed generic computing framework. **c**, illustration of distributed processing of many independent files that can be processed by a single worker. **d**, illustration of distributed processing of the split-process-merge mechanism for the distributed processing of a large image file. **e**, overall functionalities and features in LLSM5DTools.

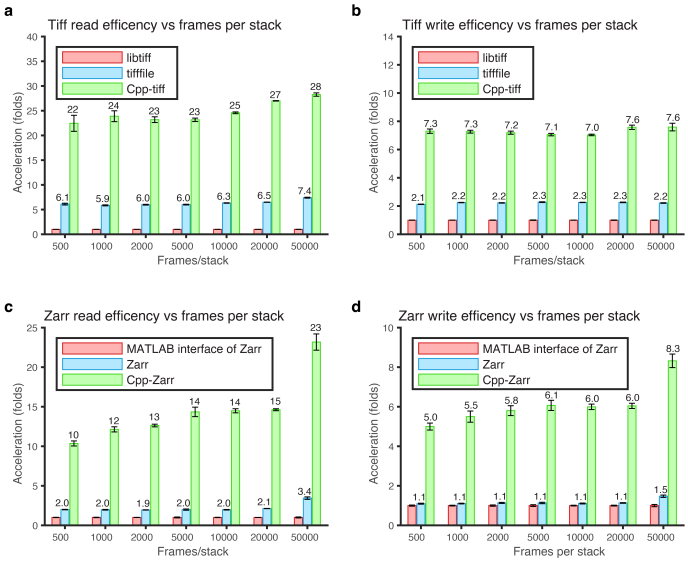

Fig. 2: Fast Tiff and Zarr readers and writers. **a**, acceleration folds of Cpp-tiff reader versus the conventional Tiff reader in MATLAB and tifffile reader in Python with different numbers of frames in the 3D stack. **b**, acceleration folds of Cpp-tiff writer versus the conventional tiff writer in MATLAB and tifffile writer in Python with different numbers of frames in the 3D stack. **a**, acceleration folds of Cpp-Zarr reader versus the conventional Zarr reader (MATLAB interface of Zarr) and native Zarr in Python with different numbers of frames in the 3D stack. **d**, acceleration folds of Cpp-Zarr writers versus the conventional Zarr writer (MATLAB interface of Zarr) and native Zarr in Python with different numbers of frames in the 3D stack. The frame sizes in the x and y axes are 512 and 1800, respectively. The benchmarks were run independently ten times on a 24-core CPU computing node.

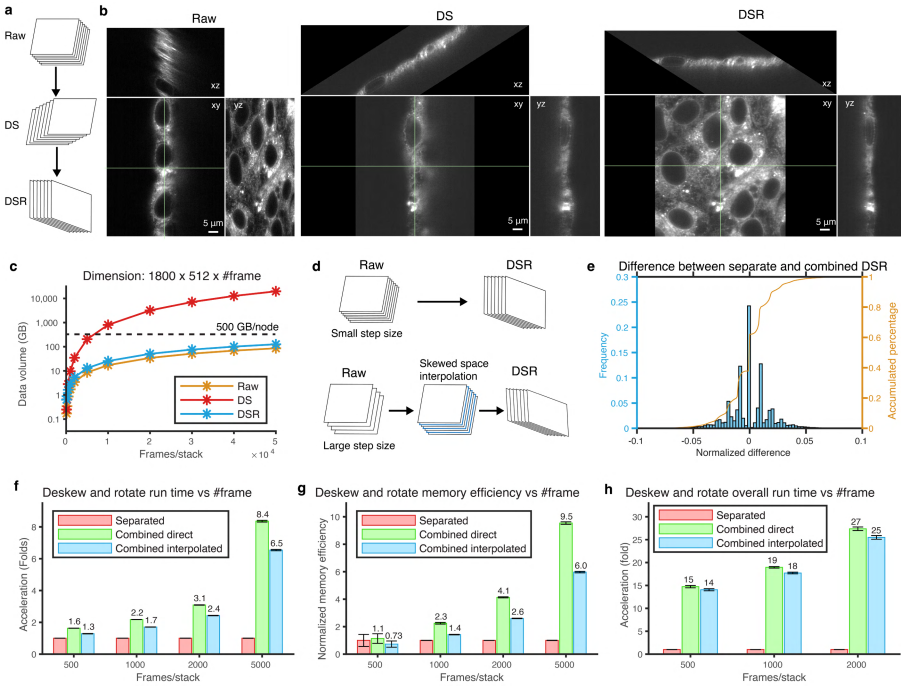

Fig. 3: Combined deskew and rotation. **a**, scheme of conventional separate deskew and rotation processes. **b**, orthogonal views of raw (skewed), deskewed (DS), and deskew/rotated (DSR) images for cultured cells with xy pixel size $0.108\,\mu m$ and scan step size $0.3\,\mu m$. **c**, the plot for the proposed data size for raw, deskewed, deskew/rotated data across different numbers of frames per stack. x and y are fixed as 512 and 1800 pixels, respectively. The dashed line indicates the memory limit (500 GB) in a computing node. **d**, scheme of combined deskew and rotation for small (top) and large (bottom) z scan step sizes. **e**, the difference between separate and combined deskew/rotation for the same image in **b**. **f**, acceleration folds for separate, combined direct, and combined interpolated deskew/rotation across different numbers of frames per stack, from 500 to 5,000. This comparison only contains deskew and rotation processes, excluding the reading and writing time of the images. **g**, memory efficiencies for separate, combined direct, and combined interpolated deskew/rotation across different numbers of z slices, from 500 to 5,000. **h**, acceleration folds for separate, combined direct, and combined interpolated deskew/rotation across different numbers of z slices, from 500 to 2,000. This comparison contains deskew and rotation processes with the reading and writing of the images using the conventional and Cpp-tiff reader and writer.

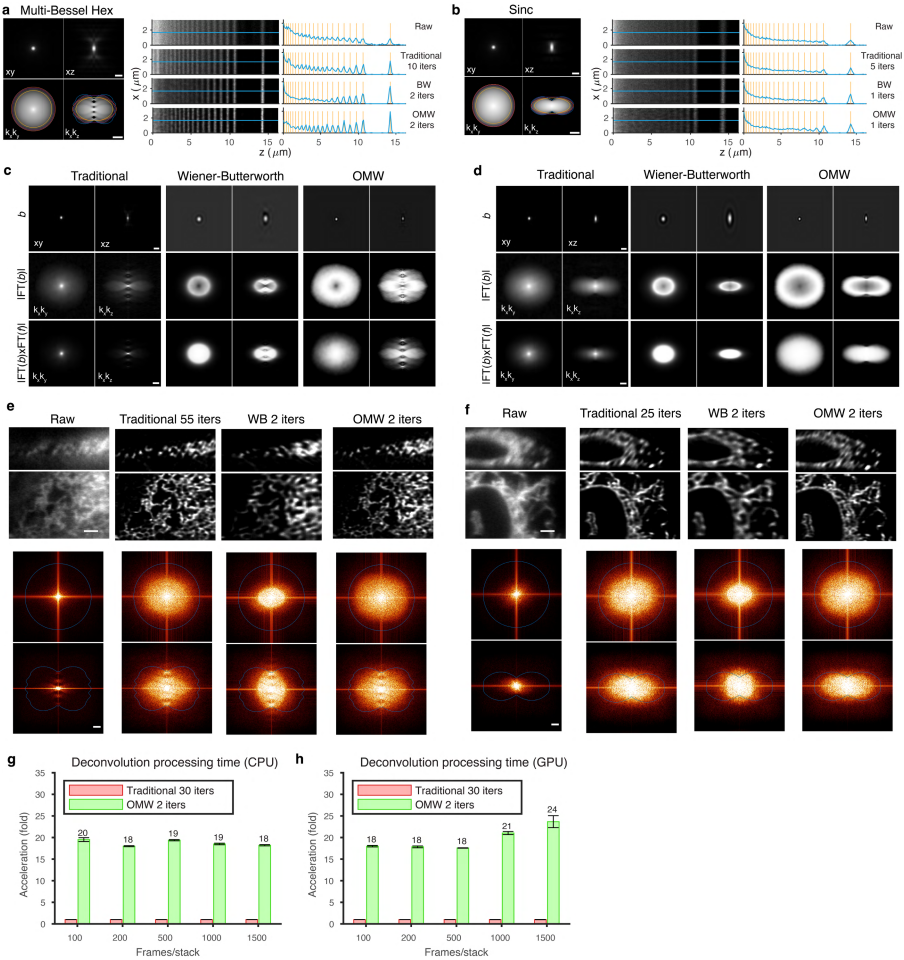

Fig. 4: Fast RL deconvolution. **a**, Left: theoretical xy and xz PSFs (top, intensity $\gamma = 0.5$, scale bar: $1\,\mu m$) and OTFs (bottom, log-scale, scale bar: $2\,\mu m^{-1}$) for the multi-Bessel (MB) light sheet with NA 0.43 and NA annulus 0.40 - 0.47. Blue: theoretical support; orange and yellow: theoretical maximum (orange) and experimental (yellow) envelopes for the BW method; magenta: experimental envelope for the OMW method. Right: illustration of deconvolution with the simulated strip pattern. The raw and deconvolved images with traditional, BW, and OMW methods are on the right, and the corresponding line cuts are on the left. The orange lines indicate the theoretical line locations, and the blue lines are for the actual intensities along the line cuts. **b**, same illustration of theoretical PSFs and OTFs and deconvolution of simulated strip pattern for Sinc light sheet (NA 0.32, $\sigma_{NA} = 5.0$). **c-d**, illustration of backward projectors (top, scale bar: $1\,\mu m$), the Fourier spectrums of backward projectors (middle, intensity $\gamma = 0.5$), and the products with forward projectors in Fourier spaces (bottom, intensity $\gamma = 0.5$, scale bar: $1\,\mu m^{-1}$) for the MB light sheet (**c**) and Sinc light sheet (**d**). **e-f**, orthogonal views of cell images for raw, conventional RL decon, BW, and OMW methods for the MB light sheet (**e**) and Sinc light sheet (**f**), the iteration numbers are in the titles (scale bar: $2\,\mu m$). The Fourier spectrums are at the bottom of the corresponding images (intensity $\gamma = 0.5$, scale bar: $1\,\mu m^{-1}$). **g-h**, deconvolution time comparison between traditional and OMW methods on CPU (**g**) and GPU (**h**) (only the deconvolution step is timed).

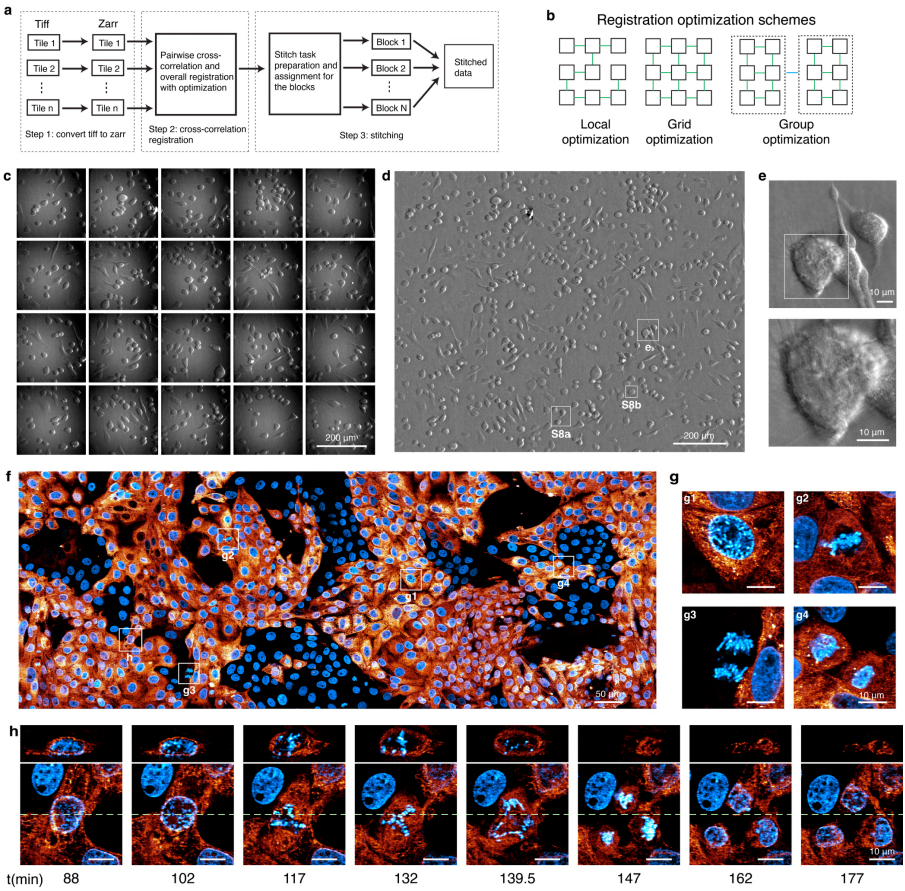

Fig. 5: Zarr-based distributed stitching framework. **a**, scheme illustration of the stitching steps. **b**, scheme illustration of registration methods: local, grid, and grouped. Boxes indicate tiles and green/blue lines indicate pairs of tiles involved in registration. **c**, illustration of raw 2D phase tiles before processing. **d**, illustration of processed phase contrast image with stitching and deconvolution. Boxes with S8a and S8b are for the cells illustrated in Fig S8a and S8b, respectively. **e**, illustration of the fine features of a cell in zoomed views from **d**. **f**, overall visualization of the long-term large field of view imaging of cultured LLC-PK1 cells at time 0. Blue: H2B-Cherry; glow: Connexin-Emerald. Intensity $\gamma = 0.5$. **g**, illustration of cropped regions from **e**. **h**, illustration of the time-lapse of three-way division cell from **e**.

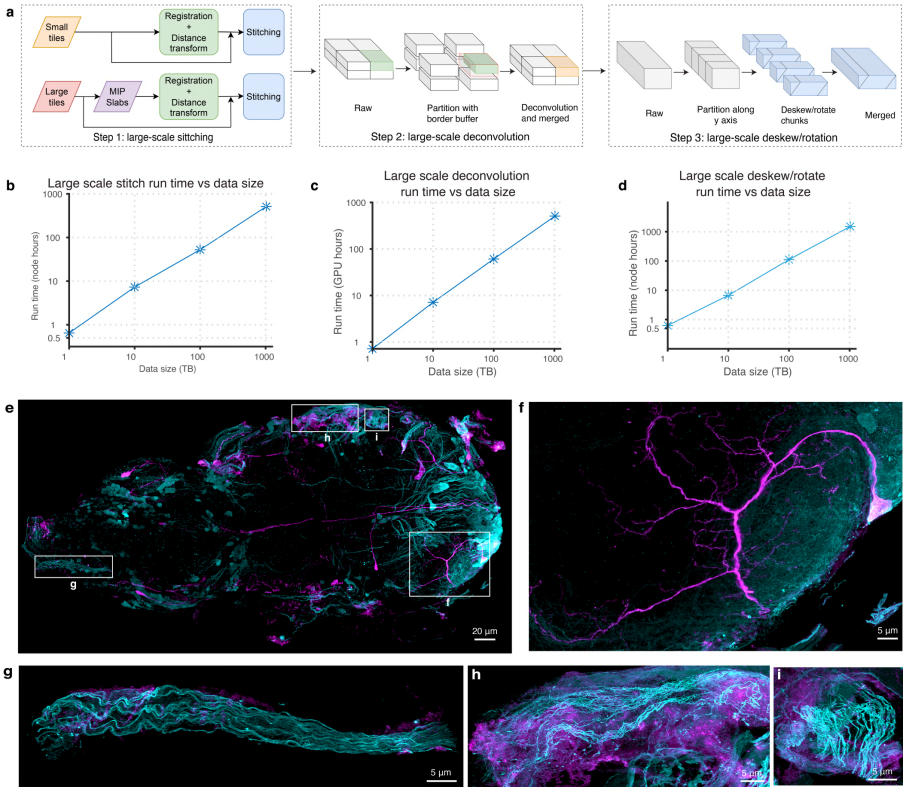

Fig. 6: Large-scale processing. **a**, scheme illustration of large-scale processing for stitching, deconvolution, and deskew/rotation. For deconvolution, the data is split over all three axes with a border size of slightly over half of the PSF size. For deskew and rotation, the data is split along the y-axis with a buffer size of one slice. **b**, total run times for large-scale stitching of a single volume with size ranging from 1 TB to 1 PB. **c**, total run times for large-scale deconvolution of a single volume with size ranging from 1 TB to 1 PB. **d**, total run times for large-scale deskew and rotation of a single volume with size ranging from 1 TB to 1 PB. **e**, visualization of whole fly VNC with 8× expansion. Cyan: VGlut$^{\text{MI04979}}$-LexA::QFAD, and purple: MN-GAL4. Intensity $\gamma = 0.5$. **f-i**, illustration of cropped regions from **e**. Intensity $\gamma = 0.75$ for all four regions.