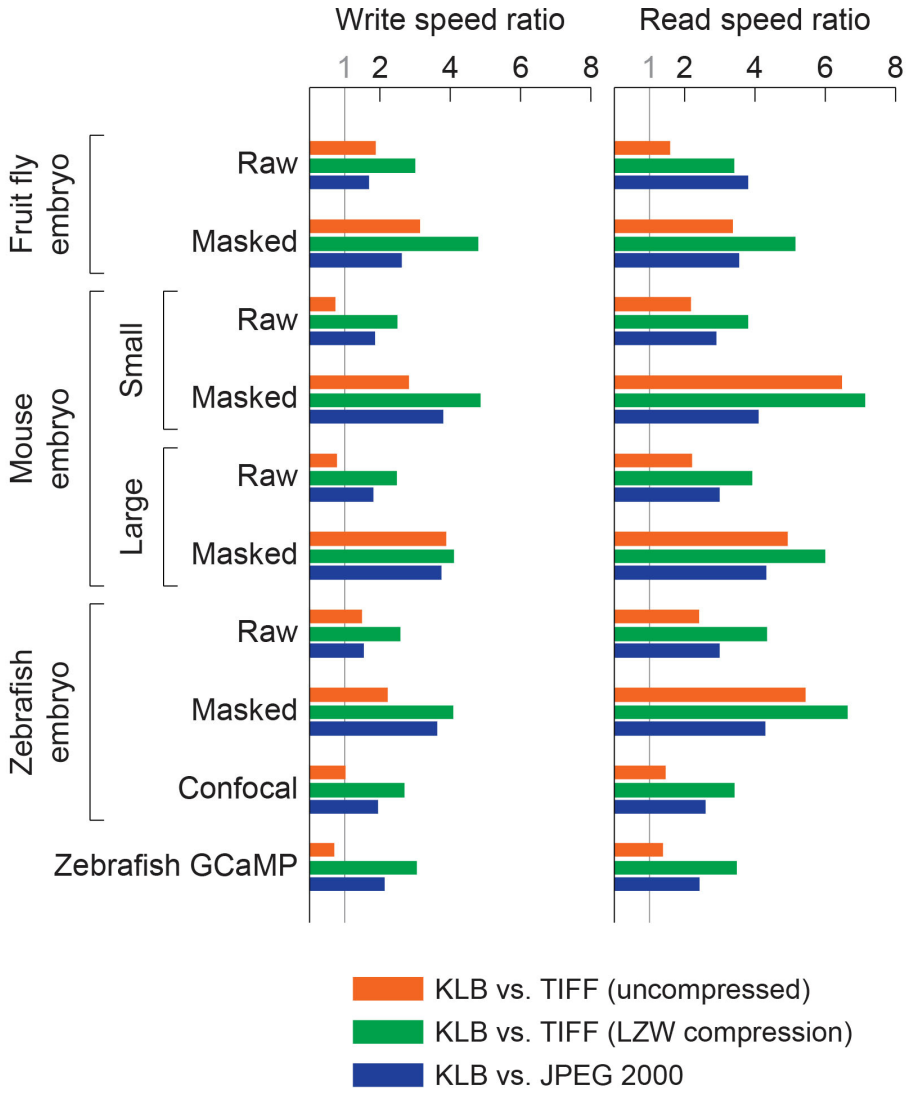
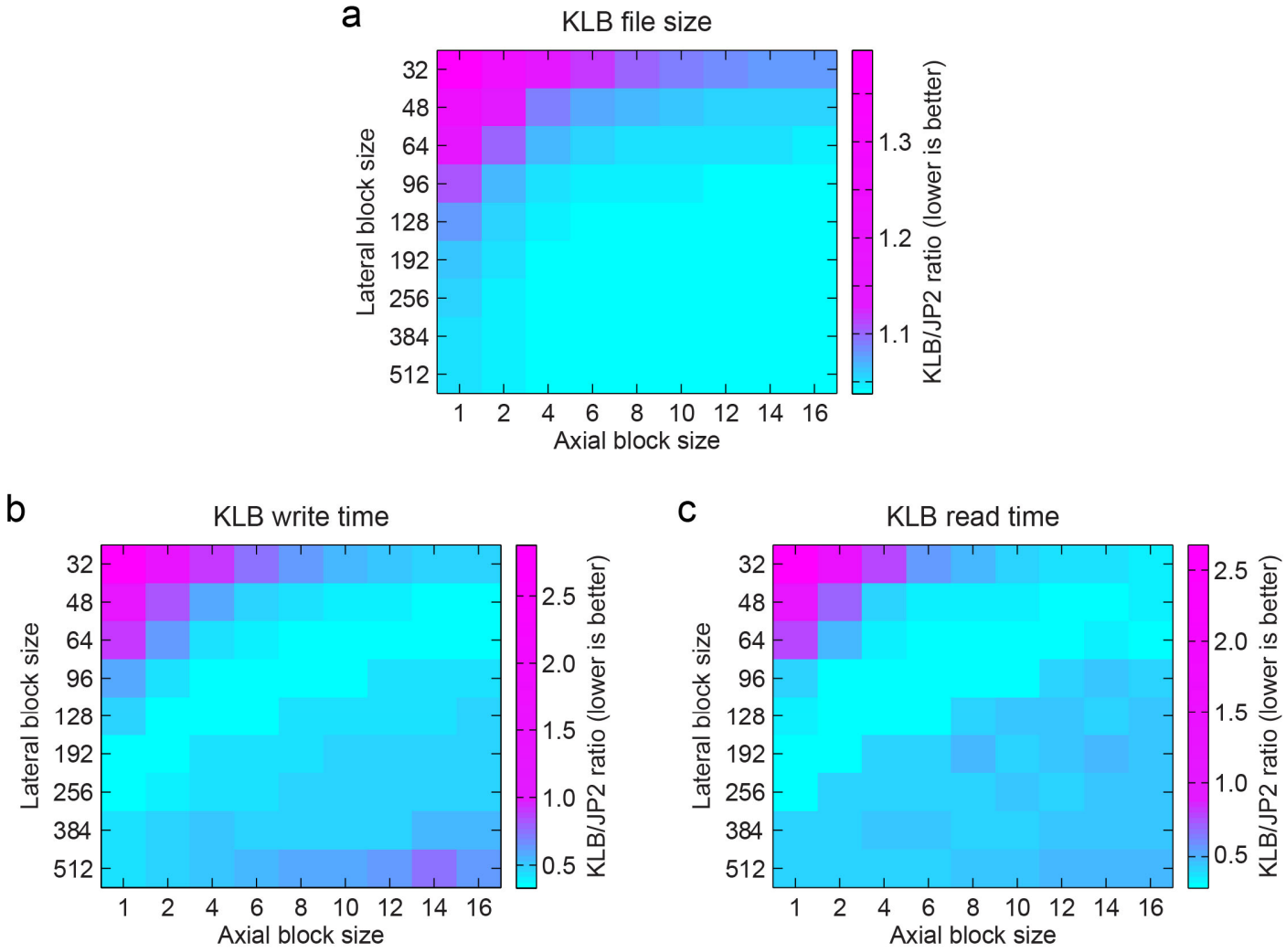


Supplementary Figure 1

Local performance of lossless compression image file formats

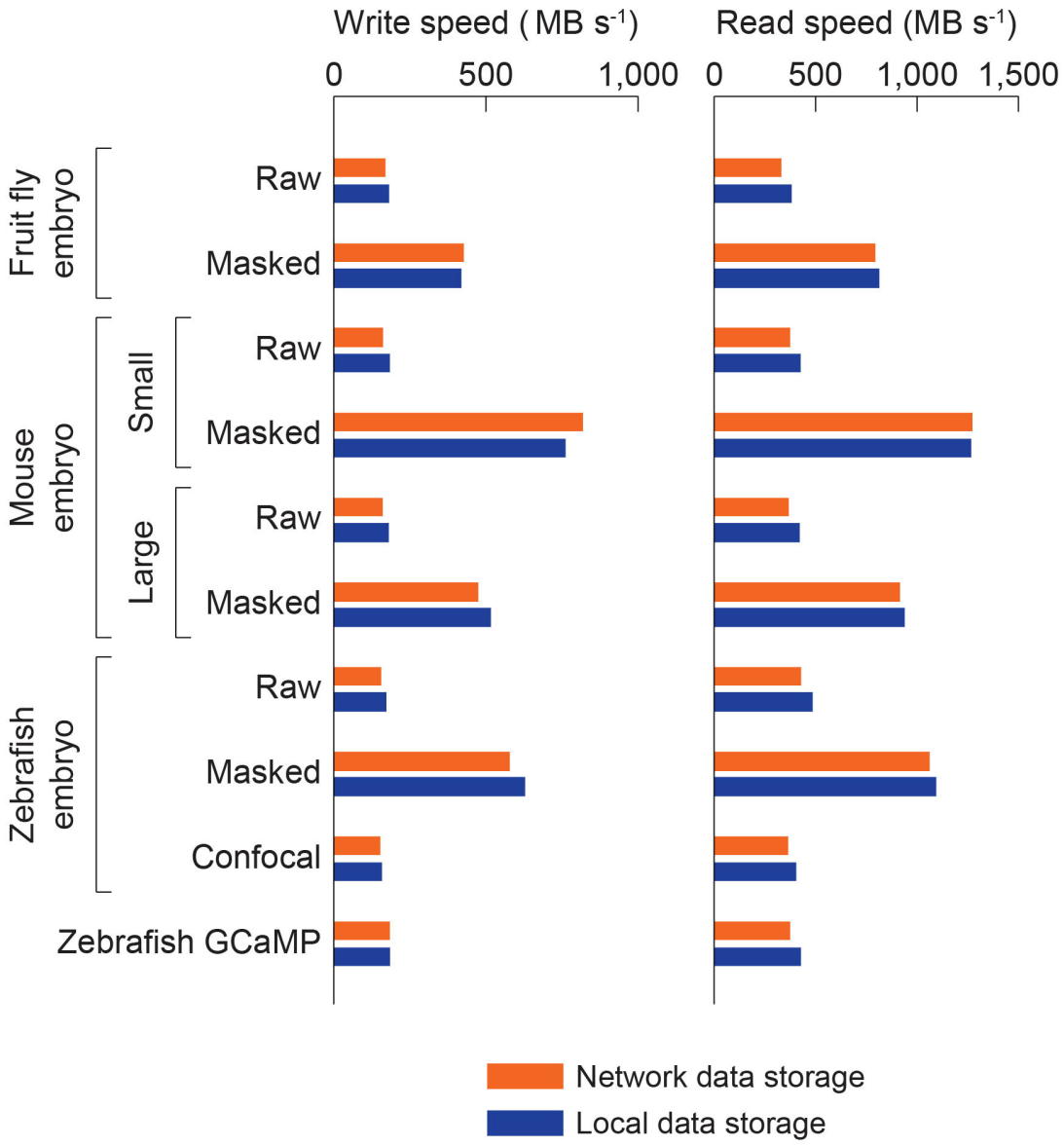
Performance of the KLB lossless compression format vs. LZW-TIFF (green) and JPEG 2000 (blue) lossless compression formats with respect to write speed (first column) and read speed (second column). The JPEG 2000 benchmark utilizes the multi-threaded commercial library PICTools Medical SDK (Accusoft). A performance comparison of KLB and uncompressed TIFF formats is included as well (orange). LZW-TIFF and uncompressed TIFF benchmarks utilize the *imread* and *imwrite* functions provided by the Image Processing Toolbox in Matlab. All performance data are provided as ratios with KLB performance in the numerator, i.e. ratios larger than one (grey lines) indicate superior performance of the KLB format. The comparison was performed using a variety of fluorescence microscopy image data sets stored locally on a high-performance RAID array built from solid-state drives (SSDs) and thus complements the network-based analysis shown in **Fig. 3** (note that performance with respect to compression ratios is identical to the data shown in **Fig. 3**). Benchmark data sets include SiMView light-sheet microscopy recordings of fruit fly, mouse and zebrafish embryonic development (data sets 1-8), confocal microscopy data of a zebrafish embryo (data set 9) and SiMView functional image data of brain activity in a larval zebrafish (data set 10). Developmental data sets (data sets 1-8) were analyzed as raw and masked versions in order to illustrate the importance of background masking for maximizing data storage and access efficiency. Please see Steps I-III in **Fig. 2** for a description of the concepts underlying background masking.



Supplementary Figure 2

Block-size dependency of KLB file size and read/write speeds

Performance comparison for KLB versus JPEG 2000 (JP2) with respect to file size (a), write time (b) and read time (c), as a function of KLB block size (in pixels). The results represent average performance across five data sets, including developmental image data from a fruit fly embryo, a zebrafish embryo and early-/late-stage mouse embryos as well as functional image data from a zebrafish larva. The larger the block size, the better the KLB compression ratio; however, this ratio reaches saturation already for relatively small block sizes. Read and write times are not optimal for extreme block sizes, i.e. both for very small and for very large blocks. If blocks are too small, communication overhead in processing threads becomes an issue. If blocks are too large, computations cannot be parallelized to the maximum extent (in the most extreme scenario, a single thread has to handle the entire image). The figure shows a diagonal band, where all three metrics are optimal or near optimal at the same time. Based on these benchmarks, we chose the default block size as 96 x 96 x 8 pixels. The JPEG 2000 benchmark utilizes the multi-threaded commercial library PICTools Medical SDK (Accusoft). Lateral size refers to the X and Y axes of the image volume. Axial size refers to the Z axis of the image volume, which is typically smaller than the lateral size in light microscopy due to anisotropic spatial resolution in the microscope and anisotropic spatial sampling of the specimen volume.



Supplementary Figure 3

KLB performance comparison for local vs. network data storage

Comparison of KLB read and write speeds on a local data drive versus a data drive mounted over the network (using a 10 Gb/s glass fiber connection). Speeds are comparable since most of the time is spent on data compression and decompression, respectively, and physical disk access introduces relatively little overhead. Moreover, most modern operating systems and RAID hardware improve I/O performance by caching and by using dedicated processors that avoid load on primary CPUs. Thus, while some blocks are compressed or decompressed others are written or read, respectively, masking I/O costs. All data points are averages based on $n = 5$ iterations of the benchmark.

Efficient processing and analysis of large-scale light-sheet microscopy data

Supplementary Information

Fernando Amat[*], Burkhard Höckendorf, Yinan Wan, William C. Lemon, Katie McDole and Philipp J. Keller[*]

Howard Hughes Medical Institute, Janelia Research Campus, Ashburn, Virginia, USA

[*]*Correspondence should be addressed to P.J.K. (kellerp@janelia.hhmi.org) or F.A. (amatf@janelia.hhmmi.org).*

Supplementary Notes

Supplementary Note 1	Header of the KLB lossless compression file format

Supplementary Tables

Supplementary Table 1	Software and hardware requirements of pipeline modules

Supplementary Software

Supplementary Software 1	KLB lossless compression file format
Supplementary Software 2	KLB Java Native Interface library and SCIFIO implementation
Supplementary Software 3	Image processing pipeline for light-sheet microscopy
Supplementary Software 4	TGMM software for segmentation and cell tracking
Supplementary Software 5	CATMAID branch for 5D image visualization and lineage editing
Supplementary Software 6	Matlab import/export scripts for TGMM, CATMAID and Imaris

Nature Protocols: doi:10.1038/nprot.2015.111

Supplementary Note 1 | Header of the KLB lossless compression file format

The KLB header contains the following items stored in binary format:

- uint8 *headerVersion*: KLB header version
- uint32 *xyzct* [5]: image dimensions (x, y, z, channels, time points)
- float32 *pixelSize* [5]: sampling of each dimension (in units of µm, index count, seconds)
- uint8 *dataType*: look-up-table for data type (uint8, uint16, etc.)
- uint8 *compressionType*: look-up-table for compression type (none, pbzip2, etc.)
- char *metadata* [256]: character block providing space for user-defined metadata
- uint32 *blockSize* [5]: block size used to partition the data in each dimension
- uint64 *blockOffset* [*Nb*]: offset (in bytes) of each block in the file
  Note: offset information stored here enables efficiently retrieving individual blocks.

The KLB file format is implemented for up to five-dimensional data (three spatial dimensions, one channel dimension, one time dimension). All items listed above have a fixed length in bytes, except for the last item (*blockOffset*), which is data-dependent. The number of blocks, *Nb*, is defined as follows:

$$Nb = \prod_{i=1}^{5} \text{ceil}\left(\frac{xyzct[i]}{blockSize[i]}\right) \qquad \text{(Eq. 1)}$$

The API provides a function call to retrieve this number.

The KLB header also allows future extensions, such as new compression formats. At the time of writing of this paper, *compressionType* supports two file formats, 'no compression' (value 0) and 'pbzip2' (value 1). Thus, if the user wishes to use other types of compression algorithms for different data sets, it is straight-forward to include support for additional algorithms. Such a modification simply requires a corresponding line in the source code, indicating the location of the code for the new compression library.

Nature Protocols: doi:10.1038/nprot.2015.111

Supplementary Table 1 | Software and hardware requirements of pipeline modules

Pipeline module(s)	Operating system(s)	Software requirements	Hardware requirements
KLB file format **Supplementary Software 1**	Windows 7 or 8 64-bit, Linux, Mac OS	None	• CPU multi-core support
KLB JNI library and SCIFIO implementation **Supplementary Software 2**	Windows 7 or 8 64-bit	None	• CPU multi-core support
Image processing pipeline for light-sheet microscopy **Supplementary Software 3**	Windows 7 or 8 64-bit, Linux, Mac OS MEX files compiled for Windows 7 or 8 64-bit	Matlab R2013b Matlab toolboxes[1)]: Curve Fitting, Image Processing, Statistics, Optimization, Signal Processing, Parallel Computing	• CPU multi-core support
TGMM **Supplementary Software 4**	Windows 7 or 8 64-bit, Linux, Mac OS	nVidia CUDA drivers	• GPU with CUDA compute capability of 2.0 or higher • CPU multi-core support
CATMAID **Supplementary Software 5**	Linux recommended	None	• SSD hard-drives for optimal performance
Import/export scripts **Supplementary Software 6**	Windows 7 or 8 64-bit, Linux, Mac OS	Matlab R2013b	• CPU multi-core support

This table provides an overview of supported operating systems and software/hardware requirements of all software packages accompanying this paper. Most software packages are compatible with multiple operating systems and can take full advantage of modern multi-core processors. Our Matlab code has been tested using Matlab version R2013b; however, these modules should in principle be compatible with any version above R2011a, without a need for code modifications.

[1)] Note that this is a comprehensive list of Matlab toolbox requirements based on the full functionality provided by our processing pipeline. However, only a subset of these toolboxes is required to run the pipeline using typical parameter settings.

Nature Protocols: doi:10.1038/nprot.2015.111

Supplementary Software 1 | KLB lossless compression file format

This software package contains the C++11 source code for the KLB file format implementation as well as wrappers for Matlab and Java. The folder *bin* contains the precompiled static and shared (DLL) libraries for Windows 7 64-bit as well as a simple executable *test_KLBIO.exe* for testing read/write operations. The source code of this executable represents a good example of how to use the API for the KLB file format. For Windows 7 64-bit, we also provide precompiled MEX files in the folder *matlabWrapper*.

Linux and Mac OS users need to compile both the source code and the Matlab wrappers to obtain libraries and executables. For the first part, a CMake file is available in the folder *src*. For the second part, the folder *matlabWrapper* contains the script *compileMex.m* for generating MEX files. The C++ libraries need to be compiled in release mode before compiling the MEX files.

In order to keep track of possible software updates, the user can also clone all files from the primary public software repository using the following git command:

git clone https://fernandoamat@bitbucket.org/fernandoamat/keller-lab-block-filetype.git

Supplementary Software 2 | KLB Java Native Interface library and SCIFIO implementation

This software package exposes the C++ API on the Java side and includes a functional implementation of a SCIFIO format that provides KLB support to image processing frameworks such as ImageJ and Knime. Precompiled native libraries for Windows and Linux (64-bit) are bundled inside the JAR file included in this software package.

For convenience, ImageJ users can follow the update site at http://sites.imagej.net/SiMView (for instructions, see http://wiki.imagej.net/How_to_follow_a_3rd_party_update_site).

Supplementary Software 3 | Image processing pipeline for light-sheet microscopy

This software package contains our Matlab code for image processing of light-sheet microscopy data sets, including (1) sCMOS image correction, background masking and KLB lossless image compression (using script *clusterPT.m*), (2) content-based multi-view image registration and fusion (using scripts *clusterMF.m*, *localAP.m* and *clusterTF.m*), (3) spatial drift correction and intensity normalization (using scripts *localEC.m* and *clusterCS.m*) and (4) adaptive local

Nature Protocols: doi:10.1038/nprot.2015.111

background correction (using script *clusterFR.m*). Please see the README file for detailed information about these software modules.

Supplementary Software 4 | TGMM software for segmentation and cell tracking

This software package contains the C++ and CUDA source code for the Tracking with Gaussian Mixture Models (TGMM) software for automated segmentation and cell tracking in light microscopy time-lapse data sets. The software package includes the following folders:

- *src*: contains all source code files. This folder also includes the file *CMakeList.txt* that can be used to compile the source code.
- *doc*: contains the documentation of the TGMM software.
- *bin*: contains Windows 7 64bit executables for running the TGMM software. When compiling the source code, the executables for the release version will be placed here. This folder also contains all necessary DLLs (CUDA and MSVC runtime) as well as the text files containing machine learning classifiers for cell division detection.

Please see the README file for detailed information on how to run and compile the TGMM software. In order to keep track of possible software updates, the user can also clone all files from the primary public software repository using the following git command:
git clone git://git.code.sf.net/p/tgmm/code tgmm-code

Supplementary Software 5 | CATMAID branch for 5D image visualization and lineage editing

This software package contains our branch of the open source software CATMAID. The software can also be cloned using the following git command:
git clone -b 5Dvisualization --single-branch https://fernandoamat@bitbucket.org/fernandoamat/catmaid_5d_visualization_annotation.git
The PDF file *UserGuide.pdf* in the root folder of this software package and the website http://catmaid.org/ provide detailed instructions for setting up a CATMAID server.

Nature Protocols: doi:10.1038/nprot.2015.111

Supplementary Software 6 | Matlab import/export scripts for TGMM, CATMAID and Imaris

This software package contains Matlab code for transferring results between TGMM, CATMAID and Imaris. In order to optimize read speed, the code for reading XML files generated by TGMM needs to be compiled into MEX files. The folder *readTGMM_XMLoutput* contains the script *compileMex.m* for this purpose. The README file contains further details on this topic and a description of the main Matlab functions included in this software package. Briefly, these Matlab functions facilitate: (1) import of TGMM tracking and segmentation results into Matlab, (2) export of image data and tracking results from Matlab to CATMAID, (3) import of cell lineage information from CATMAID into Matlab, (4) export of cell lineage information from Matlab to Imaris.

Nature Protocols: doi:10.1038/nprot.2015.111