Acceleration of 3D Feature-Enhancing Noise Filtering in Hybrid CPU/GPU Systems

V. González-Ruiz[1*], J.J. Moreno[1] and J.J. Fernández[2*]

[1]Department of Informatics, University of Almería, Agrifood Campus of International Excellence (ceiA3), Carretera de Sacramento s/n, Almería, 04120, Spain.

[2]Spanish National Research Council (CINN-CSIC), Health Research Institute of Asturias (ISPA), Av Hospital Universitario s/n, Oviedo, 33011, Spain.

*Corresponding author(s). E-mail(s): vruiz@ual.es; JJ.Fernandez@csic.es;

Abstract

FlowDenoising is a new approach to noise reduction in biological volumes obtained with three-dimensional electron microscopy (3DEM). Its abilities to enhance the structural features stem from the fact that an anisotropic Gaussian filtering is steered according to the local structures. To this end, the Optical Flow (OF) among consecutive slices is estimated, which is the most computationally expensive step in this approach. In this article, a hybrid CPU/GPU implementation of FlowDenoising is introduced and evaluated. It exploits parallel computing by distributing the workload among multiple cores and takes advantage of the massive processing in GPUs to accelerate the OF estimation. The hybrid implementation provides remarkable speed-up factors and an important reduction of the processing time, which is particularly relevant for the denoising of huge volumes typically found in 3DEM.

Keywords: High performance computing, GPU, CPU, Heterogeneous computing, Noise filtering, Optical flow, 3D electron microscopy.

1 Introduction

High-Performance Computing (HPC) refers to the efficient use of computing resources to address complex computational problems that can be expressed as algorithms that,

in general, may exceed the power of standard computing systems. Independently of the underlying hardware, HPC relies on the following main parallel programming paradigms [1, 2]. First, Parallel Computing (PC), when the problem is divided into smaller sub-problems that can be executed simultaneously typically as processes or threads, which not necessarily perform the same computation. This programming model is also referred to as Multiple Instruction Multiple Data (MIMD) [1] or thread-level parallelism [2], and it is usually implemented on multicore systems, both, with shared and distributed memory architectures. Second, Vector Processing (VP), when the Processing Units (PUs) perform the same computation over different chunks (vectors) of data. This model is also called Sing le Instruction Multiple Data (SIMD) [1] and basically exploits the data parallelism, hence also referred to as data-level parallelism [2]. Usually, Graphics Processing Units (GPUs) and Field-Programmable Gate Arrays (FPGAs) are efficient SIMD accelerators, though lately acceleration systems based on Tensor Processing Units (TPUs) or Neural Processing Units (NPUs) optimized for deep learning tasks are also appearing [2].

Nowadays, most HPC systems, desktop computers, and even laptops include GPUs that can be used to perform vector processing, even in parallel computing contexts (e.g., running multiple GPU processes –streams– on the same GPU). In fact, according to top500 (https://www.top500.org/), two of the three most powerful HPC computers, Frontier (2023) and LUMI (2022), mount thousands of AMD Instinct MI250X (a high-performance GPU). In the case of the new desktops and laptops, almost all of them include at least an in-chip GPU (such as in the case of Intel Core processors) or a discrete GPU (typically from Nvidia or AMD).

The growing availability of systems composed of different types of computing resources or with different computing capabilities brought new challenges to the HPC field. And the concept of heterogeneous computing or hybrid computing then emerged, where the different resources (CPUs, GPUs, FPGAs) available in the system are combined and jointly collaborate in tackling a problem [3, 4]. To make the most of these heterogeneous systems, the problem is decomposed into tasks that are specifically distributed to the resources that are best suited for them and/or load balancing mechanisms are used for the task distribution [3, 4].

Three-Dimensional Electron Microscopy (3DEM) is a set of 3D imaging techniques based on electron microscopy that play an important role in biology and biomedicine to visualize and analyze the inner architecture of cells, their compartments and their molecular organization [5, 6]. In all these 3DEM techniques, image processing is an indispensable tool involved in the preprocessing of the acquired 2D images, their mutual alignment, and their combination to derive the 3D tomographic volume [7]. Afterwards, image processing and analysis are essential to facilitate volume visualization and its objective interpretation (e.g., denoising, segmentation, morphological analysis of structural features) [7].

Denoising is an essential step to enable the visualization and interpretation of the 3DEM volumes due to their high noise levels [8]. Gaussian filtering is one of the most common methods used in the field because of its speed and ease of use, despite its tendency to blur the features. There have been several proposals of sophisticated denoising methods that reduce noise with abilities to preserve and enhance features [8],

including anisotropic non-linear diffusion [9–11] and recent deep-learning approaches based on the Noise2Noise or Noise2Void strategies [12–14]. However, these methods are characterized by their tricky parameter tuning or their computational burden, which is an important limitation due to the huge size of 3DEM volumes (in the order of GigaBytes). There have been a number of HPC strategies to accelerate these methods based mainly on pure CPUs or GPUs [8, 11, 15–20] while deep-learning solutions are known to require GPUs. However, these HPC efforts have not completely paved the way for the extensive application of these methods and, as a consequence, Gaussian filtering still remains a common choice in the 3DEM field [21–23].

Recently, we proposed a new approach to denoising volumes with abilities to feature enhancement [24]. The associated software, FlowDenoising[1], implements an anisotropic Gaussian-like filter guided by the estimation of the Optical Flow (OF) between adjacent slices in the 3D volume [25]. The method uses OF to estimate the local variations of the features across the volume and locally adapt the Gaussian filtering accordingly, thereby enhancing the biological structures, avoiding blurring, and overcoming the limitation of the standard Gaussian filtering. FlowDenoising has been shown to be competitive with state-of-the-art structure-preserving denoising methods, including deep-learning-based, with the important advantage that it maintains the ease of use of Gaussian filtering [24]. However, compared to standard Gaussian filtering, the use of the OF in FlowDenoising significantly increases the computational needs of the filtering algorithm.

In this article we present a hybrid FlowDenoising strategy to get the most out of standard computers, combining CPU and GPU. More specifically, we have used the GPU to estimate the OF, a computation that takes advantage of vector processing, while keeping the parallel computing capability at the same time. As a result of the joint exploitation of the CPU (leveraging the availability of multiple processing units) and the GPU (which can be used by several streams in parallel), the performance of FlowDenoising is significantly increased with regard to a pure CPU-based solution.

The rest of the article is organized as follows. First, the approach to 3D noise filtering in FlowDenoising is presented and its relevance in the context of 3DEM is highlighted. Next, our hybrid CPU/GPU implementation of FlowDenoising is introduced. After this, the evaluation tests and results are presented and analyzed. And finally, the key conclusions are outlined.

2 3D Noise filtering with feature enhancement

FlowDenoising estimates the local displacements (in general, deformations) of the structures between neighboring 2D slices of the 3D volume using an OF estimator, and then compensates for these changes by nonrigid alignment (i.e. warping) of the 2D slices before applying a Gaussian filtering [24]. This is equivalent to steering the Gaussian filtering to the local features present in the volume, thus enhancing the structures and avoiding blurring. This operation is performed along the three axes Z, Y and X consecutively by exploiting the separability property of the Gaussian function. Fig. 1 illustrates the filtering strategy in FlowDenoising along the Z-dimension, with

[1] Available at https://github.com/microscopy-processing/FlowDenoising.

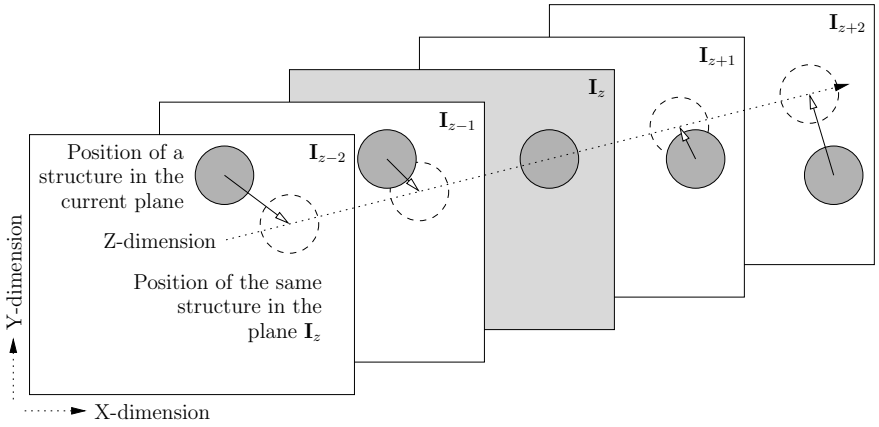

Fig. 1 Filtering strategy in FlowDenoising.

five consecutive XY slices involved in the filtering of the slice $\mathbf{I}_z$ (highlighted in gray). Four displacement fields (represented by the arrows) determined by an OF estimator are used for the alignment of the structures found in the slices $\mathbf{I}_{z-2}$, $\mathbf{I}_{z-1}$, $\mathbf{I}_{z+1}$, and $\mathbf{I}_{z+2}$ (solid gray circles) with respect to those present in $\mathbf{I}_z$ (dashed hollow circles), hence producing OF-compensated planes. The Gaussian filter along the Z-dimension (dotted straight line) is then applied to the OF-compensated slices so as to yield the filtered version of the slice $\mathbf{I}_z$.

The computation involved in FlowDenoising is described by Eq. 1

$$\mathbf{I} \overset{\rightarrow}{*} \mathbf{H} = R^{\mathrm{X}}\Big(R^{\mathrm{Y}}\big(R^{\mathrm{Z}}(\mathbf{I}) * \mathbf{h}^{\mathrm{Z}}\big) * \mathbf{h}^{\mathrm{Y}}\Big) * \mathbf{h}^{\mathrm{X}}, \tag{1}$$

where a volume $\mathbf{I}$ is convolved with a 3D Gaussian kernel $\mathbf{H}$, being[2]

$$\begin{aligned}
R^{\mathrm{Z}}(\mathbf{I}) &= \big\{\{\overset{z'\rightarrow z}{\mathbf{d}}(\mathbf{I}[z',:,:]) \approx \mathbf{I}[z,:,:] \\
&\quad \text{for } z' = z-k,\cdots,z+k\big\} \text{ for } z = 0,1,\cdots,N-1\big\}, \\
R^{\mathrm{Y}}(\mathbf{I}) &= \big\{\{\overset{y'\rightarrow y}{\mathbf{d}}(\mathbf{I}[:,y',:]) \approx \mathbf{I}[:,y,:] \\
&\quad \text{for } y' = y-k,\cdots,y+k\big\} \text{ for } y = 0,1,\cdots,N-1\big\}, \text{and} \\
R^{\mathrm{X}}(\mathbf{I}) &= \big\{\{\overset{x'\rightarrow x}{\mathbf{d}}(\mathbf{I}[:,:,x']) \approx \mathbf{I}[:,:,x] \\
&\quad \text{for } x' = x-k,\cdots,x+k\big\} \text{ for } x = 0,1,\cdots,N-1\big\},
\end{aligned}$$

and $\mathbf{h}^{\mathrm{Z}}, \mathbf{h}^{\mathrm{Y}}, \mathbf{h}^{\mathrm{X}}$ three 1D Gaussian kernels. In this formulation, $\overset{i\rightarrow j}{\mathbf{d}}()$ denotes a displacement field that expresses the local changes in the slice $\mathbf{I}_i$ to be as similar as possible to $\mathbf{I}_j$. As shown in the previous equations, for each slice to filter, there will be $2k$ slices that will be warped (k slices on each side of the reference slice, for each axis). Thus, the size of the Gaussian kernel is given by $2k+1$. In FlowDenoising, $k = 4\sigma$

[2]Notice that we have used the NumPy indexing notation.

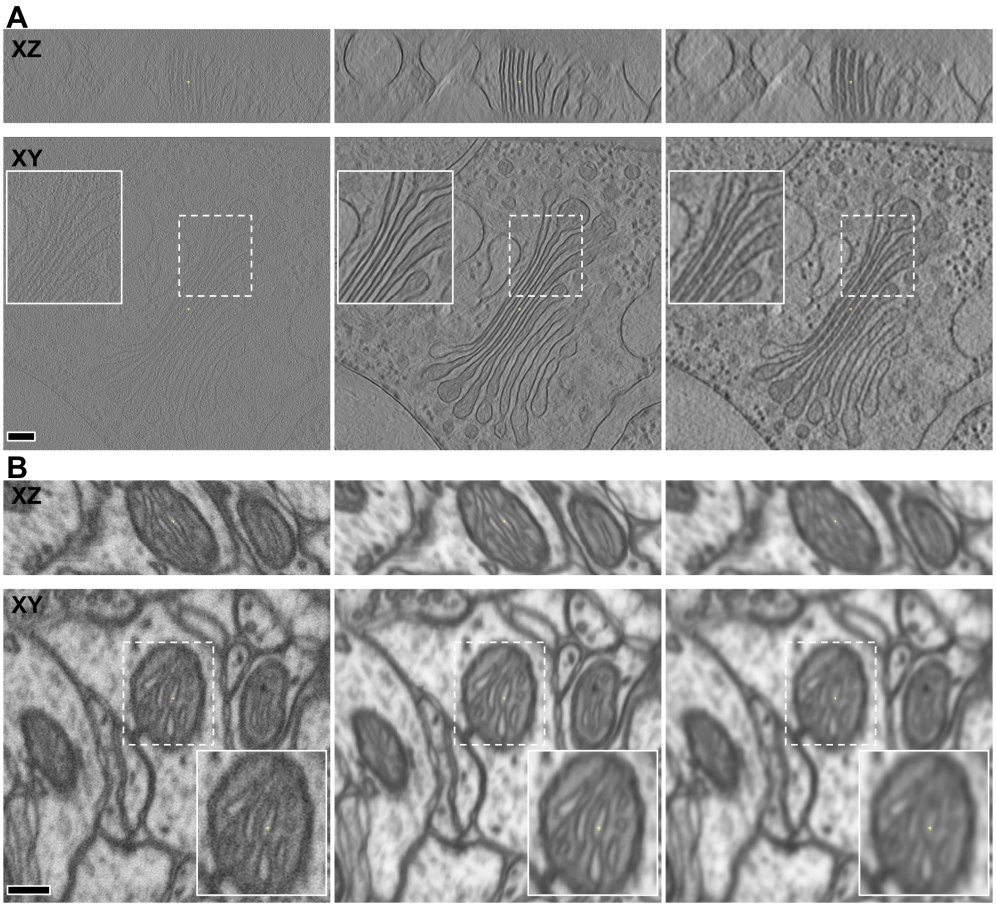

Fig. 2 FlowDenoising on 3DEM datasets

.

rounded to the nearest integer, with σ being the standard deviation of the Gaussian filtering.

FlowDenoising uses the Farnebäck OF estimator to compute the displacement fields between neighbor planes [26]. This algorithm has two main parameters: the window size w and the number of levels l. Small values of w and l prioritize fine structural accuracy in the OF estimate needed for our denoising purposes. By default, FlowDenoising uses $w = 5$ and $l = 3$, which in general provide good results. Thus, from a practical point of view, FlowDenoising has only one parameter, the standard deviation of the Gaussian filtering, σ [24].

Fig. 2 shows the effects of the noise reduction achieved by FlowDenoising on representative 3DEM datasets, (A) Golgi apparatus from an unicellular green algae visualized with cryo-electron tomography [27] and (B) rat brain tissue visualized with

FIB-SEM tomography [28, 29]. The figure shows, from left to right, the original volume, denoised with FlowDenoising, and denoised with standard Gaussian filtering. XY and XZ planes of the volumes are shown, with a little white crosshair denoting their correspondence. A sigma value of 1.5 was used in the standard Gaussian filtering and FlowDenoising in both datasets. The dashed boxes denote areas that are shown magnified in the insets. The bars correspond to 100 nm (A) and 250 nm (B). As can be seen in this figure, standard Gaussian filtering reduces the noise at the expense of blurring the structures (which is especially evident around the membranes), whereas FlowDenoising preserves the sharpness of the membranous structures while substantially attenuating the noise at the same time. The effects are particularly striking in the dataset in Fig. 2(A), which comes from cryo-electron tomography, a 3DEM modality with particularly high noise levels and low contrast.

3 Hybrid CPU/GPU implementation

Most modern computers are equipped with multicore processors and GPUs, which provide them with enormous computational power. Strategies for hybrid computing, however, are needed to take full advantage of this power.

This section describes the main contribution of this work, the hybrid CPU/GPU implementation of the filtering approach in FlowDenoising. We combine parallel computing in shared-memory multicore computers with GPU computing. For CPU-based parallel computing, we leverage Multiprocessing available in the Standard Python Library. Here, it is important to highlight that the processes use exclusive memory spaces and therefore some intercommunication mechanism is required between processes that solve a "shared" problem (in our case, the filtering of an input volume $\mathbf{I}$, generating an output volume $\mathbf{O}$). The Multiprocessing package solves this problem by using shared memory. For GPU computing, we make use of a CUDA (Compute Unified Device Architecture) version of the Farnebäck algorithm for OF estimation, as described below. In the CUDA programming model, the CPU performs kernel invocations to accelerate the corresponding computation on the GPU. The input/output data of the GPU kernels are communicated between the CPU and the GPU memories. Modern Nvidia GPUs support asynchronous, concurrent kernels/streams executions.

Figs. 3 and 4 describe our hybrid CPU/GPU implementation of FlowDenoising in schematic and pseudo-code forms, respectively. P parallel CPU processes are spawned, each of them in charge of a subset of contiguous slices of the volume along each dimension, Z, Y, and X (Fig. 3 and Code blocks 3, 5, and 7 in Fig. 4). They work in parallel and independently, by sweeping across their own slices one by one. Each process uses its own exclusive memory space, replicating all data structures except input and output volumes, which are allocated in shared memory (Fig. 3). For each slice to be processed, there is a number of k $(= 4\sigma + 1)$ adjacent slices (see Steps *.1.1) in Fig. 4) that are involved in the filtering along the corresponding direction Z, Y, and X. The OF between the current slice acting as a reference and the neighbor slices is then estimated (Steps *.1.1.1). Next, these neighbor slices are warped according to the OF (Steps *.1.1.2). Finally, Gaussian filtering is applied (Steps *.1.1.3). Note that

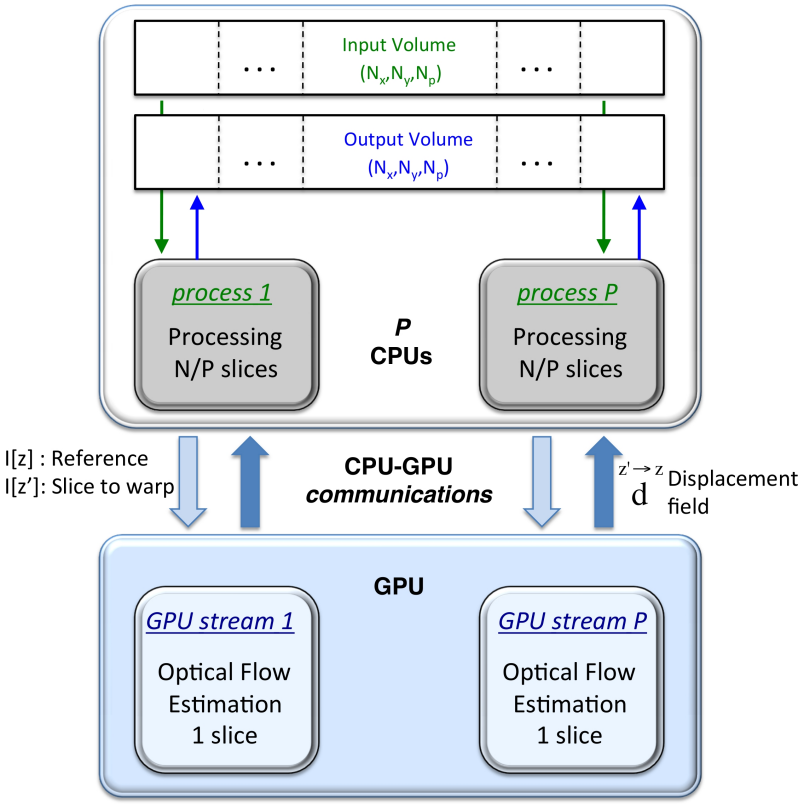

Fig. 3 Hybrid CPU/GPU implementation of FlowDenoising.

the execution of Steps *.1.1.1 for OF estimation in the CPU results in a pure CPU parallel implementation of FlowDenoising.

An analysis of the running times spent in Steps *.1.1.1, *.1.1.2 and *.1.1.3 of FlowDenoising (for a volume of $100 \times 1024 \times 1024$ voxels, on an Intel Core i7-12700H, and parameters $l = 3$, $w = 5$, $\sigma_Z = \sigma_Y = \sigma_X = 2.0$) revealed that the estimation of the OF (*.1.1.1) requires 16 and 6 more time than the convolution (*.1.1.3) and the warping (*.1.1.2), respectively. For this reason, we decided to estimate the OF in a GPU. To do this, we took advantage of the fact that there is an accelerated CUDA-based version for Nvidia GPUs of the Farnebäck method[3] in OpenCV. Thus, each CPU process launches the OF estimation of a slice to the GPU at a time, thus making P GPU streams running concurrently in the device (Fig. 3). This is denoted by "GPU" in Steps *.1.1.1 within the pseudo-code in Fig. 4. Note, however, that to achieve this, all CPU processes (in each iteration of Steps *.1.1) must transmit the corresponding slices to the GPU memory (the slice acting as a reference and the slice to be warped), and the resulting displacement fields must be sent from the GPU memory back to the CPU memory (Fig. 3). These data transfers might limit the GPU acceleration, as shown later in Section Results, when we compare with CPU-only filtering.

[3] https://docs.opencv.org/3.4/d9/d30/classcv_1_1cuda_1_1FarnebackOpticalFlow.html

FlowDenoising($\mathbf{I}, \mathbf{H}, P$): $\rightarrow \mathbf{O}$

1. $\mathbf{O} \leftarrow$ zeros_like($\mathbf{I}$)
2. $c \leftarrow \mathbf{I}$.shape$[0]/P$ /* Chunk size in the Z domain */
3. for p in range(P), in parallel run: /* Filtering in the Z domain */
   1. for z in range(c):
      1. for k in range($\mathbf{h}^{\mathrm{Z}}$.size):
         1. $\overset{z' \rightarrow z}{\mathbf{d}} \leftarrow$ flow$(\mathbf{I}[z+k+cp,:,:], \mathbf{I}[z,:,:])$ /* GPU */
         2. $R^{\mathrm{Z}}(\mathbf{I}) \leftarrow$ warp$(\mathbf{I}[z+k+cp,:,:], \overset{z' \rightarrow z}{\mathbf{d}})$
         3. $\mathbf{O}[z+cp,:,:] \leftarrow \mathbf{O}[z+cp,:,:] + R^{\mathrm{Z}}(\mathbf{I})\mathbf{h}^{\mathrm{Z}}[k]$
4. $c \leftarrow \mathbf{I}$.shape$[1]/P$ /* Chunk size in the Y domain */
5. for p in range(P), in parallel run: /* Filtering in the Y domain */
   1. for y in range(c):
      1. for k in range($\mathbf{h}^{\mathrm{Y}}$.size):
         1. $\overset{y' \rightarrow y}{\mathbf{d}} \leftarrow$ flow$(\mathbf{I}[:,y+k+cp,:], \mathbf{I}[:,y,:])$ /* GPU */
         2. $R^{\mathrm{Y}}(\mathbf{I}) \leftarrow$ warp$(\mathbf{I}[:,y+k+cp,:], \overset{y' \rightarrow y}{\mathbf{d}})$
         3. $\mathbf{O}[:,y+cp,:] \leftarrow \mathbf{O}[:,y+cp,:] + R^{\mathrm{Y}}(\mathbf{I})\mathbf{h}^{\mathrm{Y}}[k]$
6. $c \leftarrow \mathbf{I}$.shape$[2]/P$ /* Chunk size in the X domain */
7. for p in range(P), in parallel run: /* Filtering in the X domain */
   1. for x in range(c):
      1. for k in range($\mathbf{h}^{\mathrm{X}}$.size):
         1. $\overset{x' \rightarrow x}{\mathbf{d}} \leftarrow$ flow$(\mathbf{I}[:,:,x+k+cp], \mathbf{I}[:,:,x])$ /* GPU */
         2. $R^{\mathrm{X}}(\mathbf{I}) \leftarrow$ warp$(\mathbf{I}[:,:,x+k+cp], \overset{x' \rightarrow x}{\mathbf{d}})$
         3. $\mathbf{O}[:,:,x+cp] \leftarrow \mathbf{O}[:,:,x+cp] + R^{\mathrm{X}}(\mathbf{I})\mathbf{h}^{\mathrm{X}}[k]$

Fig. 4 Pseudo-code Python description of FlowDenoising.

4 Results

To evaluate the performance of the hybrid implementation of FlowDenoising, we have used two representative 3DEM datasets [27, 30]. These datasets are publicly available in EMDataResource (http://www.emdataresource.org), with id 3977 and 10780, respectively. They have the typical volume sizes found in 3DEM. They have $100 \times 1024 \times 1024$ and $1000 \times 928 \times 960$ voxels (Z,Y,X), in single precision floating point, that is, around 400 MB and 1.7 GB in size, respectively. In the following, these datasets are denoted by 3977 and 10780, respectively.

Three representative modern multicore computers have been used for the evaluation. First, a Bull Sequana X410-A5 server equipped with 2 AMD EPYC 7302 processors and a Nvidia Volta V100 GPU with 5120 CUDA cores. Second, a Fujitsu Celsius server that mounts an Intel Xeon E5-2697v4 processor and a Nvidia GTX 1080

Table 1 Processing times (in seconds) and speed-ups vs. the number of cores and the use of the GPU, for the 3977 volume ($100 \times 1024 \times 1024$, float32, 401 MB, $l = 3$, $w = 5$, $\sigma_Z = \sigma_Y = \sigma_X = 2.0$).

Number of cores	AMD EPYC 7302 + Nvidia Tesla V100			
	Time (CPU)	Speed-up (CPU)	Time (CPU/GPU)	Speed-up (CPU/GPU)
1	1112.72	1.00	220.84	5.04
2	564.35	1.97	127.18	8.75
4	286.62	3.88	87.34	12.74
8	148.79	7.48	78.97	14.09
16	77.97	14.27	89.42	12.44
Number of cores	Intel Xeon E5-2697v4 + Nvidia GTX 1080			
	Time (CPU)	Speed-up (CPU)	Time (CPU/GPU)	Speed-up (CPU/GPU)
1	1383.29	1.00	321.95	4.30
2	744.90	1.86	210.39	6.57
4	473.23	2.92	146.73	9.43
8	292.97	4.72	128.19	10.79
16	171.01	8.09	131.76	10.50
Number of cores	Intel Core i7-12700H + Nvidia RTX 3060			
	Time (CPU)	Speed-up (CPU)	Time (CPU/GPU)	Speed-up (CPU/GPU)
1	814.70	1.00	186.34	4.37
2	573.18	1.42	119.52	6.82
4	284.93	2.86	81.40	10.01
8	174.99	4.66	70.15	11.61
16	138.18	5.90	72.82	11.19

GPU with 2560 CUDA cores. Third, a Dell G15 laptop with an Intel Core i7-12700H and a Nvidia RTX 3060 with 3584 CUDA cores.

For evaluation, we have considered a pure CPU-based parallel version (that is, with the OF estimation carried out in the CPU) and the hybrid CPU/GPU implementation. The results presented in the following paragraphs are denoted by CPU and CPU/GPU, respectively.

Tables 1 and 2 present the processing times and speed-ups obtained for the two test volumes and the three machines. All speed-up factors are with respect to the pure CPU version of the code with a single core. These results clearly show that the hybrid CPU/GPU implementation of FlowDenoising provides significant speed-ups compared to the CPU-only version, and this is true for all the tested computers. Another important aspect to highlight is that these speed-ups depend on the problem size: the larger the volume dimensions, the higher the speed-ups. This is particularly true for the CPU/GPU implementation, where a significant increase is observed in Table 2 (obtained with the largest volume) with regard to Table 1. A remarkable speed-up factor approaching $30\times$ is obtained with the first machine using 8 or 16 CPU processes and GPU streams (Table 2). Apart from the speed-up factors, an important detail from the practical and user's point of view is the fact that the hybrid solutions may reduce the processing time required by huge volumes from hours to the range of 5-10 minutes in systems with 4-8 cores (Table 2).

Figs. 5 and 6 graphically show the speed-up obtained by the CPU and hybrid CPU/GPU versions. Clearly, GPU usage significantly improves the speed-ups, and

Table 2 Processing times (in seconds) vs. the number of cores and the use of the GPU, for the 10780 volume ($1000 \times 928 \times 960$, float32, 1.7 GB, $l = 3$, $w = 5$, $\sigma_Z = \sigma_Y = \sigma_X = 2.0$).

Number of cores	AMD EPYC 7302 + Nvidia Volta V100			
	Time (CPU)	Speed-up (CPU)	Time (CPU/GPU)	Speed-up (CPU/GPU)
1	9959.94	1.00	1440.45	6.91
2	5113.35	1.95	839.85	11.86
4	2565.50	3.88	490.90	20.29
8	1289.82	7.72	355.15	28.04
16	654.80	15.21	333.54	29.86
Number of cores	Intel Xeon E5-2697v4 + Nvidia GTX 1080			
	Time (CPU)	Speed-up (CPU)	Time (CPU/GPU)	Speed-up (CPU/GPU)
1	11743.69	1.00	2421.88	4.85
2	6241.71	1.88	1625.37	7.23
4	3593.58	3.27	1074.67	10.93
8	2293.43	5.12	867.26	13.54
16	1249.35	9.40	763.40	15.38
Number of cores	Intel Core i7-12700H + Nvidia RTX 3060			
	Time (CPU)	Speed-up (CPU)	Time (CPU/GPU)	Speed-up (CPU/GPU)
1	7158.56	1.00	1352.01	5.29
2	4361.95	1.64	717.52	9.98
4	2279.76	3.14	492.98	14.52
8	1467.92	4.88	402.43	17.79
16	1093.86	6.54	367.17	19.50

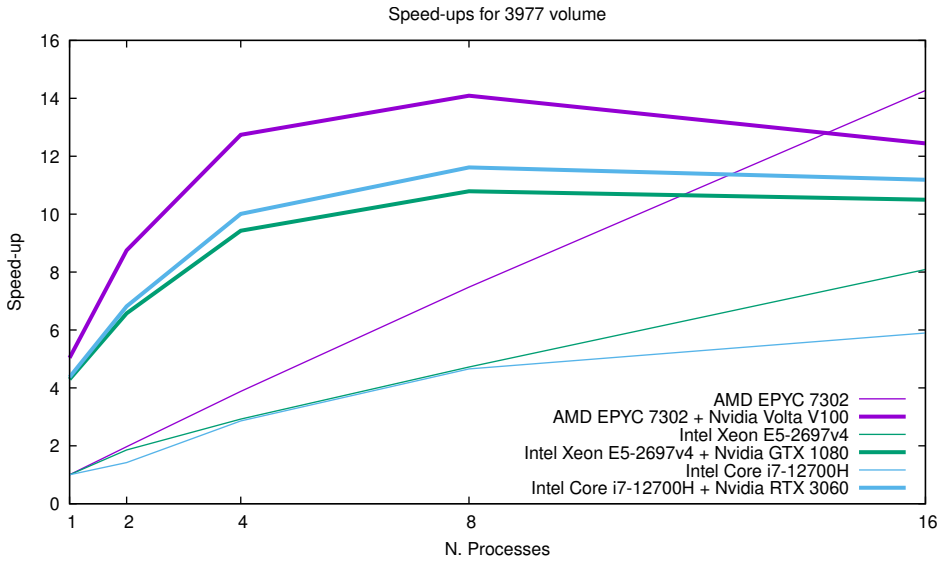

Fig. 5 Speed-ups for 3977 volume (see Table 1).

these improvements are higher if the volume to filter is big enough (Fig. 6). The curves also reflect the computational power of the GPUs, in terms of CUDA cores, with the

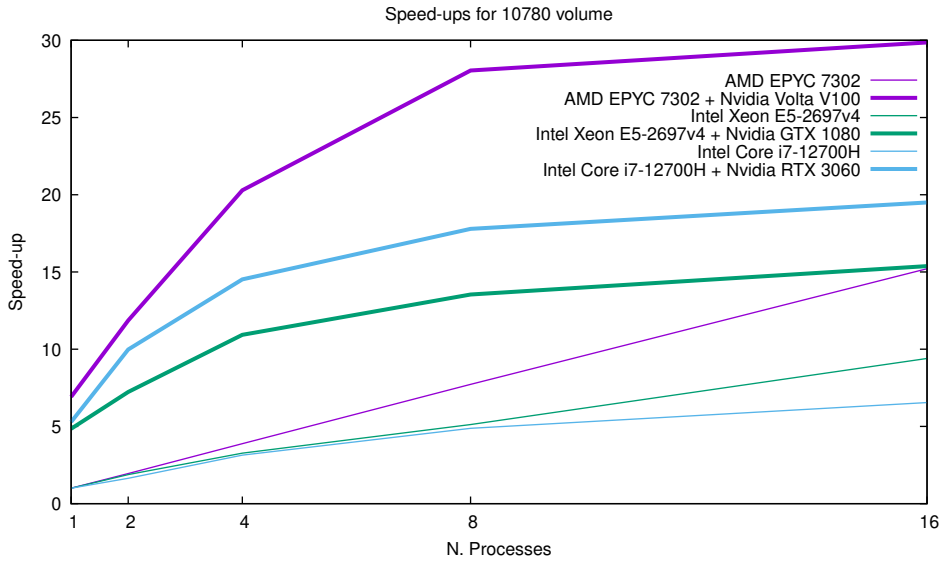

Fig. 6 Speed-ups for for 10780 volume (see Table 2).

Table 3 Time analysis for 3977 volume in the Intel Core i7-12700H + Nvidia RTX 3060 (see Fig. 7). Times are in seconds.

Number of cores	OFE (Steps *.1.1.1)	Warping (Steps *.1.1.2)	Convolution (Steps *.1.1.3)	Transfer
1	30.43	75.87	193.50	56.04
2	23.91	45.00	58.20	54.97
4	20.72	27.60	19.18	67.15
8	24.83	17.90	8.21	99.79
16	32.11	12.78	4.38	183.59

Volta V100 showing the highest curve and the GTX 1080 the lowest. It can also be observed that the pure CPU version shows a rather linear speed-up curve, with the slope being dependent on the platform (with better scalability in the first machine). However, the slope of the speed-ups in the hybrid CPU/GPU version is decreasing as a function of the number of processes. This is caused by the overhead from data transmission between the CPU and GPU in relation to the workload in the GPU and, therefore, the effective speed-up provided by the GPU, which depends on the chunk size (see Steps 2, 4 and 6 in the Fig. 4). Thus, for smaller volumes the penalty is larger, and even a decay in the speed-up factor is obtained with a number of CPU processes and GPU streams higher than 8 (Fig. 5).

In order to further explore the decreasing slope of the speed-ups in the hybrid CPU/GPU version, we have measured the time spent in the most demanding steps of FlowDenoising (OFE, warping, and convolution) and CPU/GPU transfers. Figs. 7 and 8 as well as Tables 3 and 4 show the results for the Dell G15 laptop, but a similar behavior has been obtained for the other machines. It can be seen that

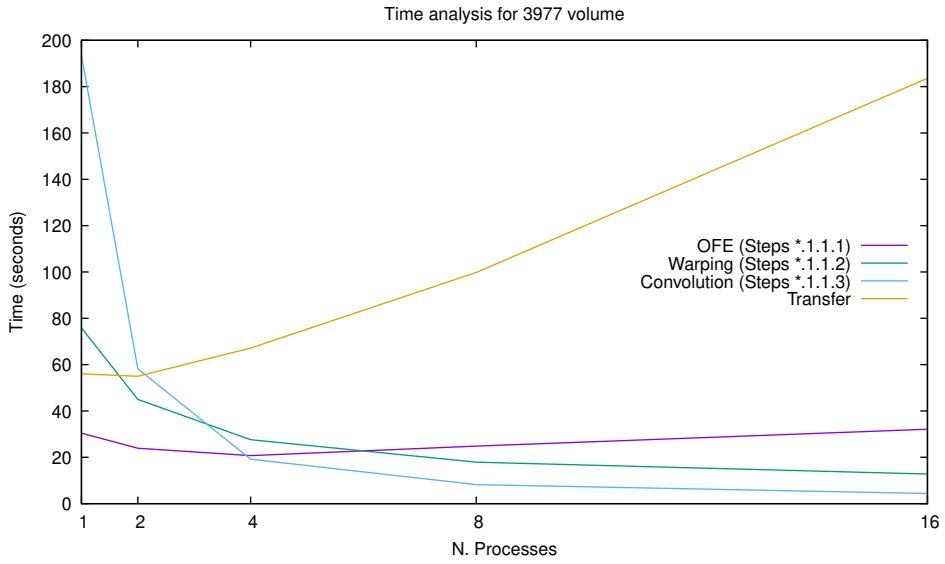

Fig. 7 Time analysis for 3977 volume in the Intel Core i7-12700H + Nvidia RTX 3060 (see Table 3). OF estimation (OFE), warping and convolution times (see Fig. 4) are the average time among all the processes. The transfer time is the total time.

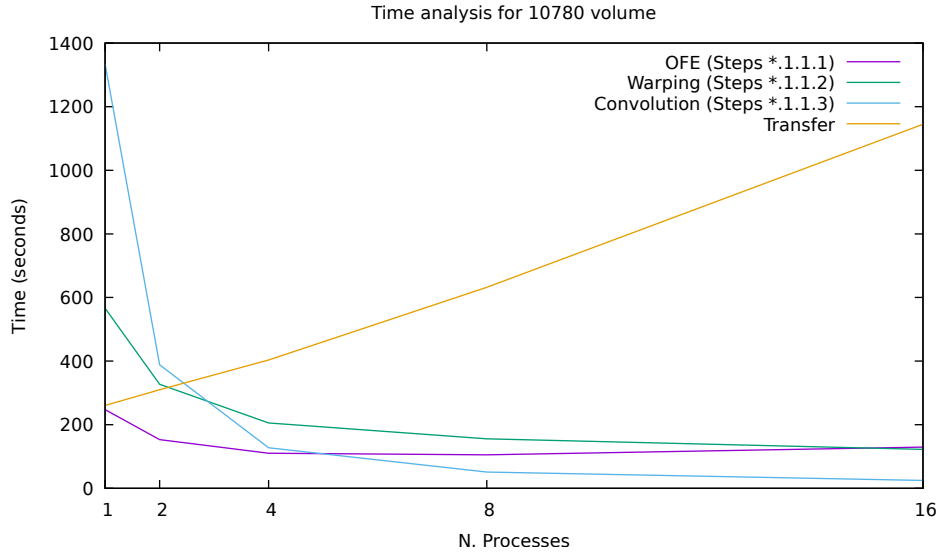

Fig. 8 Time analysis for 10780 volume in the Intel Core i7-12700H + Nvidia RTX 3060 (see Table 4). OF estimation (OFE), warping and convolution times (see Fig. 4) are the average time among all the processes. The transfer time is the total time.

the computational part of FlowDenoising scales reasonably well with the number of

Table 4 Time analysis for 10780 volume in the Intel Core i7-12700H + Nvidia RTX 3060 (see Fig. 8). Times are in seconds.

Number of cores	OFE (Steps *.1.1.1)	Warping (Steps *.1.1.2)	Convolution (Steps *.1.1.3)	Transfer
1	247.25	566.05	1334.43	260.64
2	153.21	327.16	388.38	309.97
4	110.23	205.70	127.42	403.54
8	105.22	155.46	51.21	631.99
16	129.71	122.37	24.54	1144.67

processes, which is especially true for warping and convolution, while OFE exhibits a limited gain for higher number of processes. However, the transfer time of the data between the CPU memory and the GPU memory increases almost linearly with the number of processes, which might turn into a cause of the drop in efficiency in terms of speed-up. Nonetheless, note that these transfer times are total accumulated times. In practice, the execution through GPU streams indeed overlaps computation with these communications, thereby hiding part of the latency associated to these transfers. Fig. 9 (a snapshot of the command nvtop) presents the percentage of GPU power used during an execution of FlowDenoising on the Dell G15 with the 3977 dataset using 16 GPU streams. It shows around 90% of use during the first stage (filtering in Z domain, where the slices are 1024x1024 in size), and then approach 100% for the second and third stages (Y and X domains, with slices of 1024x100). Therefore, despite the transfers, the GPU remains nearly at the full computation power thanks to the latency hiding achieved by the use of GPU streams, thus suggesting that the implementation is not latency-bound.

To gain a deeper understanding, we also conducted a comparative analysis between the aforementioned implementation of FlowDenoising and an alternative approach with reduced transfers. In this alternative implementation, the slices required by each stream are cached in the GPU memory, resulting in a reduction of CPU/GPU data traffic as they are uploaded to the GPU only once per filtering direction. The results showed that, as a consequence of the temporal overlapping of the computation and the transmission of the slices in the described version, this alternative version yielded only a slight improvement in wall-times, approximately 1% shorter. However, it came at the expense of a significant increase in GPU memory usage. Given our specific interest in denoising large volumes using the maximum number of PUs, we ultimately chose to use the described version of FlowDenoising to showcase the execution times.

5 Conclusions

In this article, we have presented and evaluated a hybrid CPU/GPU implementation of FlowDenoising, a feature-enhancing noise filtering approach for volumes in 3DEM. Our hybrid strategy consists of exploiting the parallel capabilities of modern multicore computers by distributing the set of slices among the CPUs, and taking advantage of the higher computational capabilities of GPUs to launch the most expensive step in FlowDenoising, namely the estimation of the optical flow, to these accelerators. The

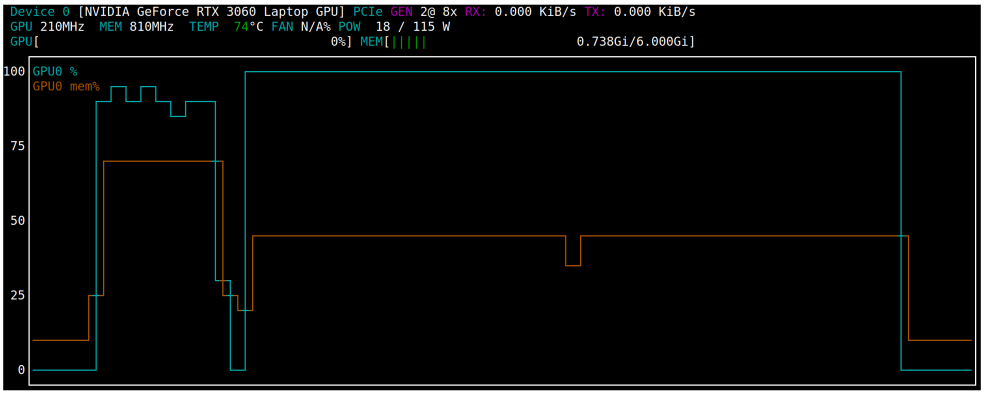

Fig. 9 Percentage of GPU computing power and memory used during a FlowDenoising execution.

results we obtained clearly show that this hybrid strategy outperforms a pure CPU-based parallel implementation. From the practical point of view, it is remarkable the reduction of processing time achieved by the hybrid strategy, which can denoise huge volumes typically found in 3DEM in just few minutes in rather standard computers equipped with 4-8 cores and a GPU. Our future plans include exploration of more elaborate solutions to improve the ratio computation vs. transfer times in the GPU. Furthermore, we are considering the development of multi-GPU implementations. These future improvements are expected to further accelerate FlowDenoising.

Acknowledgments. Work supported by MCIN/AEI/10.13039/501100011033, "ERDF A way of making Europe" and by the "European Union NextGenerationEU/PRTR" through grants PID2021-123278OB-I00, TED2021-132020B-I00, PID2022-139071NB-I00 and PDC2022-133370-I00.

Declarations

Ethical Approval

Not applicable.

Competing interests

The authors have no competing interests to declare that are relevant to the content of this article.

Authors' contributions

VGR and JJF: Conceptualization, Investigation, Software, Validation, Writing, Funding Acquisition. JJM: Investigation, Software.

Funding

This work was supported by Spanish Ministry of Science and Innovation MCIN/AEI/10.13039/501100011033, "ERDF A way of making Europe" and by the "European Union NextGenerationEU/PRTR" through grants PID2021-123278OB-I00, TED2021-132020B-I00, PID2022-139071NB-I00 and PDC2022-133370-I00.

Availability of data and materials

The data of this study will be available from the corresponding authors on reasonable request. Code will be available through the github of the authors (https://github.com/microscopy-processing/FlowDenoising).

References

[1] Flynn, M.: Some computer organizations and their effectiveness. IEEE Trans. Computers **C-21**, 948–960 (1972)

[2] Hennessy, J.L., Patterson, D.A.: Computer Architecture. A Quantitative Approach. 6th Ed. Morgan Kauffman, USA (2019)

[3] Brooks, D.: CPUs, GPUs, and Hybrid Computing. IEEE Micro **31**, 4–6 (2011)

[4] Agulleiro, J.I., Vazquez, F., Garzón, E.M., Fernandez, J.J.: Hybrid computing: CPU+GPU co-processing and its application to tomographic reconstruction. Ultramicroscopy **115**, 109–114 (2012) https://doi.org/10.1016/j.ultramic.2012.02.003

[5] Eisenstein, M.: Seven technologies to watch in 2023. Nature **613**, 794–797 (2023)

[6] Peddie, C.J., *et al.*: Volume electron microscopy. Nature Reviews Methods Primers **2**, 51 (2022) https://doi.org/10.1038/s43586-022-00131-9

[7] Fernandez, J.J., Martinez-Sanchez, A.: Computational methods for three-dimensional electron microscopy (3DEM). Computer Methods and Programs in Biomedicine **225**, 107039 (2022) https://doi.org/10.1016/j.cmpb.2022.107039

[8] Fernandez, J.J.: Computational methods for electron tomography. Micron **43**, 1010–1030 (2012) https://doi.org/10.1016/j.micron.2012.05.003

[9] Fernandez, J.J., Li, S.: An improved algorithm for anisotropic nonlinear diffusion for denoising cryo-tomograms. J. Struct. Biol. **144**, 152–161 (2003) https://doi.org/10.1016/j.jsb.2003.09.010

[10] Fernandez, J.J., Li, S.: Anisotropic nonlinear filtering of cellular structures in cryo-electron tomography. Comput. Sci. Eng. **7**(5), 54–61 (2005) https://doi.org/10.1109/MCSE.2005.89

[11] Moreno, J.J., Martinez-Sanchez, A., Martinez, J.A., Garzon, E.M., Fernandez, J.J.: TomoEED: fast edge-enhancing denoising of tomographic volumes. Bioinformatics **34**, 3776–3778 (2018) https://doi.org/10.1093/bioinformatics/bty435

[12] Lehtinen, J., Munkberg, J., Hasselgren, J., Laine, S., Karras, T., Aittala, M., Aila, T.: Noise2Noise: Learning image restoration without clean data. In: Dy, J., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning, vol. 80, pp. 2965–2974 (2018)

[13] Krull, A., Buchholz, T.-O., Jug, F.: Noise2void-learning denoising from single noisy images. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 2129–2137 (2019). https://doi.org/10.1109/CVPR.2019.00223

[14] Buchholz, T.-O., Krull, A., Shahidi, R., Pigino, G., Jekely, G., Jug, F.: Content-aware image restoration for electron microscopy. In: Muller-Reichert, T., Pigino, G. (eds.) Three-Dimensional Electron Microscopy. Methods in Cell Biology, vol. 152, pp. 277–289. Academic Press, USA (2019). https://doi.org/10.1016/bs.mcb.2019.05.001

[15] Tabik, S., Garzón, E.M., García, I., Fernandez, J.J.: High performance noise reduction for biomedical multidimensional data. Digital Signal Processing **17**, 724–736 (2007) https://doi.org/10.1016/j.dsp.2006.11.004

[16] Fernandez, J.J.: High performance computing in structural determination by electron cryomicroscopy. J. Struct. Biol. **164**, 1–6 (2008) https://doi.org/10.1016/j.jsb.2008.07.005

[17] Cuomo, S., Michele, P.D., Piccialli, F.: 3D data denoising via Nonlocal Means filter by using parallel GPU strategies. Comput Math Methods Med **164**, 523862 (2014)

[18] Tabik, S., Murarasu, A., Romero, L.F.: Anisotropic nonlinear diffusion for filtering 3d images on gpus. In: 2014 IEEE International Conference on Cluster Computing (CLUSTER) (2014). https://doi.org/10.1109/CLUSTER.2014.6968786

[19] Kwon, K., Kim, M.S., Shin, B.S.: A fast 3D adaptive bilateral filter for ultrasound volume visualization. Comput Methods Programs Biomed **133**, 25–34 (2016) https://doi.org/10.1016/j.cmpb.2016.05.008

[20] Yano, K., Sugimoto, K., Kamata, S.-i.: GPU-friendly Approximate Bilateral Filter for 3D Volume Data. In: 2018 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC), pp. 2054–2058 (2018). https://doi.org/10.23919/APSIPA.2018.8659773

[21] Steyer, A.M., Ruhwedel, T., Nardis, C., Werner, H.B., Nave, K.A., Möbius, W.: Pathology of myelinated axons in the PLP-deficient mouse model of spastic paraplegia type 2 revealed by volume imaging using focused ion beam-scanning electron microscopy. J Struct Biol **210**, 107492 (2020) https://doi.org/10.1016/j.jsb.2020.107492

[22] Hennies, J., Lleti, J.M.S., Schieber, N.L., Templin, R.M., Steyer, A.M., Schwab, Y.: AMST: Alignment to Median Smoothed Template for Focused Ion Beam Scanning Electron Microscopy Image Stacks. Scientific Reports **10**, 2004 (2020) https://doi.org/10.1038/s41598-020-58736-7

[23] Uwizeye, C., Decelle, J., Jouneau, P.-H., Flori, S., Gallet, B., Keck, J.-B., Bo, D.D., Moriscot, C., Seydoux, C., Chevalier, F., Schieber, N.L., Templin, R., Allorent, G., Courtois, F., Curien, G., Schwab, Y., Schoehn, G., Zeeman, S.C., Falconet, D., Finazzi, G.: Morphological bases of phytoplankton energy management and physiological responses unveiled by 3D subcellular imaging. Nature

Communications **12**, 1049 (2021) https://doi.org/10.1038/s41467-021-21314-0

[24] González-Ruiz, V., Fernández-Fernández, M.R., Fernandez, J.J.: Structure-preserving gaussian denoising of FIB-SEM volumes. Ultramicroscopy **246**, 113674 (2023) https://doi.org/10.1016/j.ultramic.2022.113674

[25] González-Ruiz, V., Fernandez, J.-J.: Flowdenoising: Structure-preserving denoising in 3d electron microscopy (3dem). SoftwareX **23**, 101413 (2023) https://doi.org/10.1016/j.softx.2023.101413

[26] Farnebäck, G.: Two-Frame Motion Estimation Based on Polynomial Expansion. In: Scandinavian Conference on Image Analysis, pp. 363–370 (2003). https://doi.org/10.1007/3-540-45103-X_50

[27] Bykov, Y.S., *et al.*: The structure of the COPI coat determined within the cell. eLife **6**, 32493 (2017) https://doi.org/10.7554/eLife.32493

[28] Knott, G., Rosset, S., Cantoni, M.: Focussed Ion Beam Milling and Scanning Electron Microscopy of Brain Tissue. JoVE (Journal of Visualized Experiments) **53**, 2588 (2011) https://doi.org/10.3791/2588

[29] Lucchi, A., Li, Y., Fua, P.: Learning for structured prediction using approximate subgradient descent with working sets. In: 2013 IEEE Conference on Computer Vision and Pattern Recognition, pp. 1987–1994 (2013). https://doi.org/10.1109/CVPR.2013.259

[30] Wietrzynski, W., *et al.*: Charting the native architecture of chlamydomonas thylakoid membranes with single-molecule precision. eLife **9**, 53740 (2020) https://doi.org/10.7554/eLife.53740