Parallelization and validation of algorithms for zebrafish cell lineage tree reconstruction from big 4D image data

Robert Spir, Karol Mikula, Nadine Peyrieras

Department of Mathematics and Descriptive Geometry, Faculty of Civil Engineering, Slovak University of Technology, Radlinskeho 11, 810 05 Bratislava, Slovakia, spir@math.sk, mikula@math.sk, nadine.peyrieras@inaf.cnrs-gif.fr.

Abstract: The paper presents numerical algorithms, postprocessing and validation steps for an automated cell tracking and cell lineage tree reconstruction from large-scale 3D+time two-photon laser scanning microscopy images of early stages of zebrafish (Danio rerio) embryo development. The cell trajectories are extracted as centered paths inside segmented spatio-temporal tree structures representing cell movements and divisions. Such paths are found by using a suitably designed and computed constrained distance functions and by a backtracking in the steepest descent direction of a potential field based on a combination of these distance functions combination. Since the calculations are performed on big data, parallelization is required to speed up the processing. By careful choice and tuning of algorithm parameters we can adapt the calculations to the microscope images of vertebrae species. Then we can compare the results with ground truth data obtained by manual checking of cell links by biologists and measure the accuracy of our algorithm. Using an automatic validation process and visualisation tool that can display ground truth data and our result simultaneously, along with the original 3D data, we can easily verify the correctness of the tracking.

Keywords: cell tracking; validation; big data; parallel computation

1 Introduction

With the recent research in biology and medicine the in vivo imaging of various organisms at cell level at very early stages of development without corrupting the cell integrity and normal evolution of the embryo is becoming possible thanks to the advancement of the modern microscopy techniques. Using two-photon laser scanning microscopy we can obtain long periods of the 3D+time images of an embryonic development with relatively short time step beginning just a few hours after the fertilization. By expression of the fluorescence protein trough the RNA injection at the one-cell stage we can obtain the labelling of cell nuclei and membranes

(Fig. 1). Thanks to the transparency for the laser scanning and thanks to the similarity with human cells in many aspects, this can be used to perform various analyses of the embryogenesis and the growth of organisms and the Zebrafish (Danio reio) embryogenesis is studied extensively. The results are used both in basic and applied biology and medicine research, e.g. in the anticancer drug design. To process such a large amount of data there is a need for efficient and parallel algorithms that will allow us to quickly process this data and obtain satisfying results.

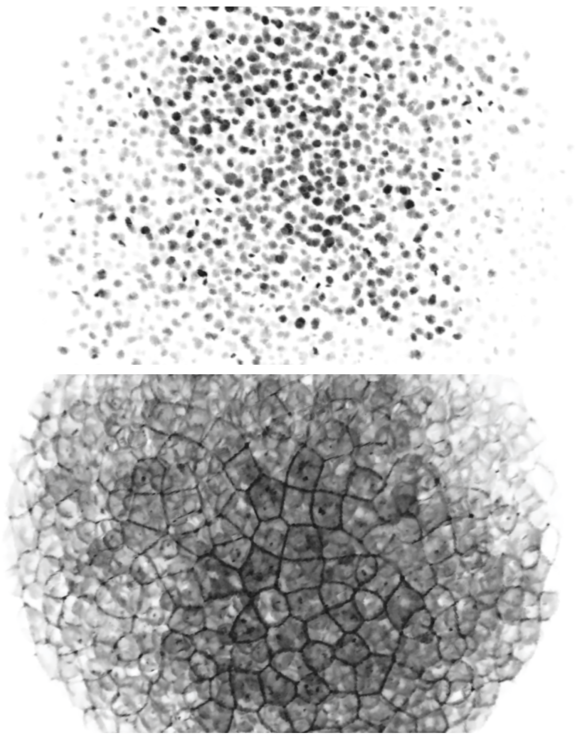

Figure 1
Volume rendering of cell nuclei (top) and cell membrane (bottom) data from confocal microscope.

In this paper we focused on the cell tracking part of our algorithm, mainly on the efficient OpenMP parallel implementation of 4D Rouy-Tourin scheme with fixing [11] and on the detailed description of various postprocessing steps and result validation by comparison with ground truth data. These topics are not discussed in [7, 9].

The paper is organized as follows. In the section 2 we briefly describe the individual steps of our image processing workflow. In the next section, section 3, we present our approach to cell trajectories and lineage tree extraction and discuss the numerical approaches used in the tracking method and parameters that can be used to tune and improve the tracking results. In section 4 we discuss the paralelization of the distance function calculation. Then we discuss extraction of the cell trajectories and postprocessing of the results in section 5. In the last section, section 6, we discuss the numerical experiments on real 3D+time image sequences of the early zebrafish embryogenesis and perform the automatic validation of the results by comparison with the ground truth data containing correct cell links in time verified manually by biologists and manual visual checking using our visualization tool.

2 Image processing workflow

Developing algorithms for the analysis of the embryogenesis images, we first start with the image filtering to reduce the noise in the original 3D images obtained from a two-photon laser microscope. For the filtering we are using a geodesic mean curvature flow method (GMCF) based on the nonlinear diffusion equation that was developed by Chen et al. in [1] and efficiently discretized by Kriva et al. in [2, 3]. We apply the filtering to the cell nuclei and membrane images and in this step we eliminate various small structures and noise which do not represent the cell nuclei or membrane structures. For the computation we are using a parallel MPI implementation of the algorithm and Red-Black SOR parallel solver.

In the next step we detect the cell nuclei identifiers using a level-set center detection method (LSCD) developed by Frolkovic et al. in [3, 4]. This model is based on the nonlinear advection-diffusion equation which is applied to the result of GMCF filtering. The basic assumption is that cell nuclei are represented by humps of relatively higher image intensity where the diameter of cell nuclei is rather large compared to the diameter of spurious inner cell structures. Thus applying a sufficient number of evolutionary steps of LSCD, the spurious structures are shrunk and smoothed but larger humps are still significant and we can look for the remaining local maxima in the volume representing the coordinates of the cell nuclei. The method is parallelized with MPI using Red-Black SOR parallel solver to solve the linear system. These cell nuclei coordinates will represent the basic input data for our segmentation and tracking algorithms.

Using the cell identifiers, we can move to the segmentations of inner cell structures, the cell nuclei. Here we use the generalized subjective surface equation (GSUBSURF) of advection-diffusion type developed by Sarti et al. in [3, 5, 6]. The cell identifiers are used as seeds for the segmentations.

With the cell nuclei coordinates from the LSCD and nuclei segmentations from GSUBSURF we came up with the idea of the construction of 4D segmentation that will sufficiently approximate the real cell shapes. The first approach was to use 3D ellipsoids with constant half-axes for all cells in all time steps to obtain some very rough approximation of cell nuclei and combine them to the 4D segmentation. Here we found that thanks to the imperfection of the input data a non negligible number of cell identifiers that should represent the cell nuclei were missing and the result of the tracking was many short disjointed trajectories. To improve the results, the next step was considering 4D ellipsoids constructed also in the time dimension. Using such time overlapping we were able to achieve connected trajectories (close gaps) even in cases when the cell nuclei are missing in single time step [7, 8]. We improved the approximation of cell nuclei by using the diameters of 4D ellipsoid from performed segmentations of each single cell in the whole dataset so each 4D ellipsoid is unique and is approximating the corresponding cell. By shrinking or enlarging this diameter one can further refine the quality of the tracking results. Calculating suitable designed distance functions inside the 4D segmentation and using their combination to construct a potential field, we can extract cell trajectories representing their spatio-temporal movement during the embryogenesis. Here we can use a special weighted combination of the distance functions during the potential field construction to further improve the tracking results [9].

There are also developed other approaches to the problem of cell tracking. In [10] Amat et al. are using the methods based on sequential Bayesian approach with Gaussian mixture models are used. In [3] there has been presented a method to build the cell lineage tree for the complex stages of Zebrafish embryo development based on stochastic simulated annealing minimization of a heuristic energy functional.

3 Cell tracking

Our method is composed of five basic steps:

- construction of a 4D segmentation yielding the 4D spatio-temporal tree structure,
- computation of constrained 4D distance functions inside this 4D segmentation and designing of a proper combination of them in order to built a potential field for tracking,
- extraction of cell trajectories using the steepest descent direction of the potential field,
- centering the extracted trajectories inside the 4D spatio-temporal trees in order to eliminate duplicates,
- construction of the cell lineage tree by detecting trajectories which merge together when going backward in time indicating mitosis and thus a branching node of the lineage tree.

3.1 4D segmentations

4D segmentation is a spatio-temporal structure which approximates the space-time movement of cell nuclei. According to Zanella et al. in [6, 12] the shape of cell nuclei during zebrafish embryogenesis is reasonably approximated by spheres or ellipsoids. Thus, in order to construct 4D segmentation we use cell identifiers detected in all time steps, $s^l_m, m = 1, \ldots, n^l_C, l = 1, \ldots, N_\theta$ (m denotes cell identifier index at time step l and N_θ is the number of time steps) by LSCD method from [4] and create 4D ellipsoids around all these points. To determine halfaxes of the ellipsoids we use real cell nuclei segmentations obtained using the generalized subjective surface (GSUBSURF) method [6, 12] paired with cell coordinates from the

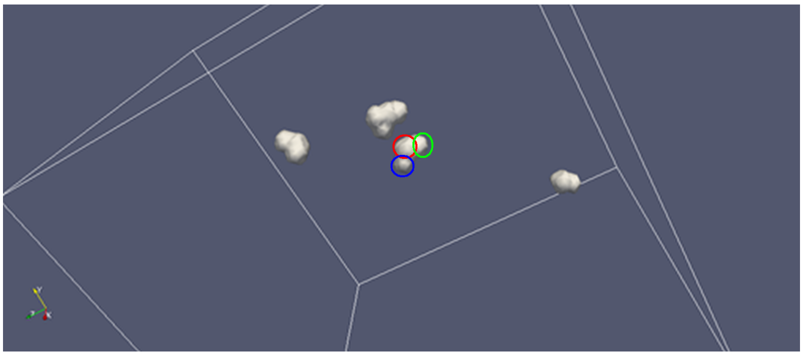

Figure 2
Projection of the 4D segmentation to one time step. For each cell identifier we can see the central sphere (inside the red circle) surrounded by the overlaps of the sphere from the previous time step (inside the green circle) and from the next time step (inside the blue circle). Three further parts of 4D segmentation are shown as well.

cell detection step. We calculate the volume of the real segmented nucleus and compute the radius of a sphere with the same volume. This radius is then used as spatial half-axes for the constructed ellipsoid. Here, we also introduce a parameter S representing shrinking of the half-axes (if $S < 0$) or expanding of them (if $S > 0$). A slight shrinking of the real radius is used later in the tracking algorithm since it helps to have a spatially non-overlapping tubular structure representing the cell movement. This parameter is tuned by comparison of the tracking with ground truth data and its optimal choice can improve the quality of tracking results. In temporal direction we are using halfaxis equal to $d\theta$ corresponding to the image acquisition interval (Fig. 2). The nonzero temporal halfaxis is important due to the time overlap which we create and thus we improve connectivity of 4D spatio-temporal tree structures. Thanks to the time overlap we interconnect branches of the 4D spatio-temporal tree where a cell center was not detected in one frame but it was detected in two neighboring frames and thus we correct false negative errors of the cell center detection algorithm.

3.2 Computing 4D distance functions and building the potential field

Our 4D segmentation containing 4D spatio-temporal tree structures can be represented by a 4D piecewise constant function, with some BIG, a sufficiently large number greater than maximum distance in the data (e.g. $BIG = 10^4$), value outside of the segmentation and with zero value inside it. Using this information, we compute two types of distance functions inside the 4D spatio-temporal tree structures. We call them constrained because all the calculations are constrained by the boundaries of the 4D segmentation. Due to that fact, the computed distances between 4D points of the 3D+time image sequence are not the standard Euclidean distances in R^4 but they represent minimal Euclidean paths between the points inside the 4D segmentation.

The first type of distance function will be denoted by $D_L(x_1,x_2,x_3,\theta)$. It is calculated gradually inside all simply connected regions, starting from cell centers in the lowest possible time step θ. After the calculation is completed in all regions reachable from these cell centers, we fix the computed values and continue the calculation from centers in the next time step but only in regions where the values are not yet fixed. Using this approach we calculate the distance function $D_L(x_1,x_2,x_3,\theta)$ inside the whole 4D segmentation. At the end all doxels inside the 4D segmentation contain the value of the distance to the farthest (backwardly in time) cell identifier to which it is continuously connected. The second type of distance function, $D_B(x_1,x_2,x_3,\theta)$, represents the distance of any inner point of the 4D segmentation to the boundary of the 4D segmentation. Again, $D_B(x_1,x_2,x_3,\theta) = 0$ for all (x_1,x_2,x_3,θ) outside of the 4D segmentation.

Based on these facts we build the new potential field

$$V(x_1,x_2,x_3,\theta) = D_L(x_1,x_2,x_3,\theta) - \alpha D_B(x_1,x_2,x_3,\theta), \tag{1}$$

which is used for the extraction of cell trajectories. The parameter $\alpha > 0$ is used to adjust the weight of D_B function to tune and improve tracking results.

Both distance functions $D_L(x_1,x_2,x_3,\theta)$ and $D_B(x_1,x_2,x_3,\theta)$ are computed numerically on the doxel structure of the 3D+time image sequence using the Rouy-Tourin scheme [13]. We numerically solve the time relaxed Eikonal equation, which has the following form

$$d_t + |\nabla d| = 1 \tag{2}$$

for the unknown function $d(x_1,x_2,x_3,\theta,t)$. Here we solve a spatially 4D problem, so ∇d is the 4D gradient of the function d, i.e. the vector of partial derivatives with respect to x_1,x_2,x_3 and θ variables. For discretization of the equation (2) we use the spatially 4D Rouy-Tourin scheme.

We identify here the 4D doxels with finite volumes V_{ijkl} having four indices. Without losing generality, we rescale the time step $d\theta$ to be equal to $dx_1 = dx_2 = dx_3$ and denote their common value by h_D (standardly we set $h_D = 1$). Let d^n_{ijkl} denote the approximate value of solution d in the barycenter of V_{ijkl} in a discrete step $t^n = n\tau_D$ where τ_D is the length of step discretizing t variable. Then, for every V_{ijkl} we define the index set N_{ijkl} of all (p,q,r,s) such that $p,q,r,s \in \{-1,0,1\}$, $|p|+|q|+|r|+|s| = 1$. In order to build the scheme for any $(p,q,r,s) \in N_{ijkl}$, we define

$$D^{pqrs}_{ijkl} = \left(\min\left(d^{n-1}_{i+p,j+q,k+r,l+s} - d^{n-1}_{ijkl}, 0\right)\right)^2 \tag{3}$$

and then also

$$M^{pqrs}_{ijkl} = \max\left(D^{-p,-q,-r,-s}_{ijkl}, D^{p,q,r,s}_{ijkl}\right). \tag{4}$$

Using this notations, the 4D Rouy-Tourin scheme for solving equation (2) has the following form

$$d^n_{ijkl} = d^{n-1}_{ijkl} + \tau_D - \frac{\tau_D}{h_D}\sqrt{M^{1000}_{ijkl} + M^{0100}_{ijkl} + M^{0010}_{ijkl} + M^{0001}_{ijkl}}\,. \tag{5}$$

Since the scheme produces monotonically increasing updates that are gradually approaching a steady state, we can implement (5) in a computationally efficient way [11]. Let us consider the index set I of all indices (i,j,k,l) and the set $\mathscr{F}^n$ that contains the indices $(i,j,k,l) \in I$ of the finite volumes where the steady state has been already approximately reached, i.e. $|d^n_{ijkl} - d^{n-1}_{ijkl}| < \varepsilon$, where ε is some chosen small threshold value. The basic principle is that we perform all computations only in the finite volumes that have not yet reached the steady state. The number of these finite volumes and the computational time needed to complete one time step of the procedure gradually decrease until the values in all cells are fixed. The method is given by *Algorithm 1*:

```
✓ Do while $\mathscr{F}^n \neq I$
    ✓ Do for all $(i,j,k,l) \in I$
        ✓ if $(i,j,k,l) \in \mathscr{F}^n$ then continue
        ✓ else
            ✓ update $d^n_{ijkl}$ using (5)
            ✓ if $|d^n_{ijkl} - d^{n-1}_{ijkl}| < \varepsilon$ then $\mathscr{F}^n = \mathscr{F}^n \cup \{(i,j,k,l)\}$
✓ $n = n+1$
```

4 OpenMP parallel implementation of 4D Rouy-Tourin scheme

The two main standards for parallel programing are MPI - Message Passing Interface [14] and OpenMP - Open Multi-Processing [15]. With the MPI, the parallel calculation runs in multiple independent processes, so this standard is suitable for systems with distributed memory. Here the processes communicate using MPI subroutines to exchange data using some communication media (e.g. local network or high speed interconnect). The disadvantage is that this communication always includes some overhead. On the other hand, OpenMP runs the calculation in single process with multiple parallel threads. Each thread has access to shared portion of the memory, so the communication tends to be fast with minimal overhead. The disadvantage is that this approach cannot be used on systems with distributed memory. When dealing with 4D data and 4D computations by using MPI parallelization, we would need to send large amount of data during interprocess communication. E.g., we would need around 120MB of data to send and receive between each neighboring parallel processes during each time step of distance function calculation. So, we decided to use OpenMP for parallelization, which is faster in this case. The drawback of this approach is that we need to run the calculation on single shared memory computer and we are limited by the amount of the available memory of such a server. For example, for a 320 time step dataset with $512 \times 512 \times 120$ voxels 3D volumes we need roughly 128GB of shared memory. Since current generation HPC servers have 256GB or more of shared memory at disposal, OpenMP approach is applicable. In comparison, with MPI approach in standard cluster configuration, where all internode communication is going through 1Gbit network, it spent 2 seconds on the communication part, when running on two servers. The total time of calculation of a single Rouy-Tourin time step took 5-7 seconds, depending on the number of already fixed points. We can see that the communication part is significant, contributing 30% of the single time step execution. When running on a single server, so the communication does not need to go through network, it took 1 second, which is still a measurable difference. In OpenMP version there is no communication needed.

Using OpenMP directives we were able to parallelize most parts of the whole calculation by dividing the data in the time dimension and doing the calculations on the data in parallel. Since the distance function is calculated in an explicit manner using only the data form previous time step, no communication is needed. The only parts that remain serial are reading of the input data and writing of the results, since these parts are limited by the speed of disk drives and there is no sense on parallelizing them.

On standard multiprocessor servers, the global memory space is divided into separate units, so-called NUMA nodes, where each processor has fast access to a local NUMA node, but slow access to other NUMA nodes. E.g. each NUMA node could contain 32GB of memory, which equals 256GB of total memory with 8 processors. Since in our calculations we need 128GB of memory, even a serial code can use the memory from multiple nodes. In this case, it is more efficient to use *numactl* utility to set node memory interleaving policy, when the memory is allocated from all nodes in a round-robin fashion. Since the other CPUs are not under load and the memory access is evenly distributed between nodes, the performance hit is lower than if the program uses memory only from certain nodes. In Fig. 3 we can see memory distribution when running serial calculation using default settings and when using node memory interleaving set with *numactl* utility. Using the memory interleaving, the memory usage is equal across all nodes, while with default settings, only memory from nodes 1,2,3,5 and 7 is used. In this test, the second distance function D_B calculation took 752 seconds with default settings and 558 seconds with memory interleaving, which is more than 25% faster.

Using the parallel processing, it is important to bind an individual parallel process or in our case, thread, to a single NUMA node, or processor core. Thus, when the thread allocates some local memory, it does not happen that the process scheduler switches the thread to a different CPU where the thread does not have any data in local memory. When the thread is bound to a certain CPU, it will run on this CPU for its whole lifetime. For CPU binding with OpenMP programming we can use the environment variable OMP_PROC_BIND. If it is set to TRUE, then the individual threads will be bound to single CPU cores in a sequential manner. This is not optimal, because when we want to run e.g. only 4 parallel threads, they will be bound to the first four cores of the first CPU, so they will all run on a single NUMA node and have to share a single memory access. The optimal would be a binding to NUMA nodes in round-robin fashion. To use this type of binding, we can use GOMP_CPU_AFFINITY environment variable that is not in the OpenMP standard, but is an extension of GCC compiler. Setting this variable to "0 4 8 12" will bind the

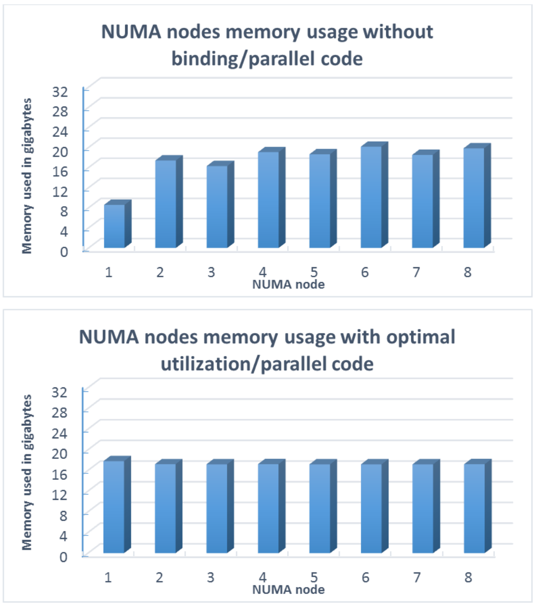

Figure 3
on top, node memory usage with default settings, memory is used only on certain nodes, on bottom, node memory is used evenly with memory interleaving.

first thread to CPU core 0, second to 4 etc. and each thread will run on single NUMA node with dedicated memory controller and the combined memory bandwidth will be 4-times wider than in the previous case. In our test, with calculation running in parallel on the first four CPU cores on a single NUMA node, the calculation of D_B took 362 seconds, when running on first cores of four NUMA nodes the calculation took only 143 seconds, which is more than twice as fast.

When running the calculation in 32 parallel threads without any binding, it took 57 seconds and with binding it took 32 seconds which is again nearly twice as fast. Without the binding, the parallel threads are switched between CPUs and NUMA nodes by the operating system process scheduler. The node memory usage of parallel calculation with and without binding can be seen in Fig. 4.

Since the OpenMP program runs as a single process with multiple threads in a single shared memory space, we can use a special programming technique which utilizes the so-called lazy allocation feature of Linux kernel where the physical memory is actually allocated only after writing in it. In the program for computation of the distance function we allocate one large array using single call of the malloc() function which allocates only virtual memory in Linux and no physical memory is used. Then we write zeroes into this array in a parallel `for` cycle and only then the real physical memory is allocated. Since we use static OpenMP scheduling, each parallel thread will allocate memory on node on which it is actually running and then in all subsequent parallel cycles, the thread will always get the same part of the calculation and will use only local memory allocated in the first parallel cycle.

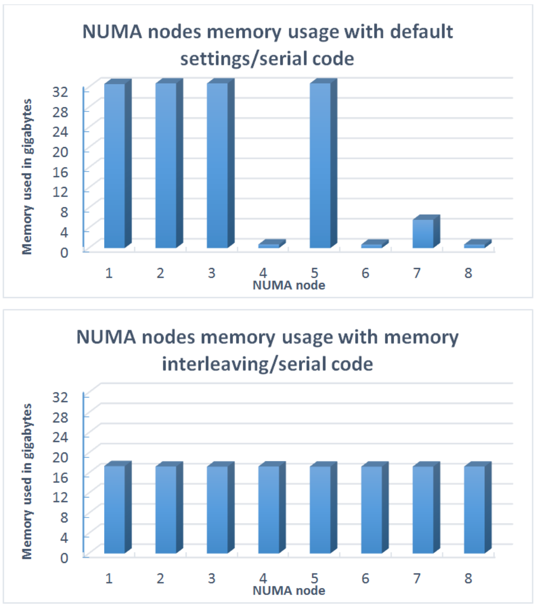

Figure 4
on top, node memory usage without thread binding is uneven, on bottom memory usage with thread binding and optimal allocation, each thread is accessing only the local memory and the calculation is faster.

With this approach we achieve optimal uniform NUMA node memory usage even without memory interleaving and combining with thread binding the threads are mainly using the local node memory and accesses to memory on other nodes are minimized.

The speed-up of the parallel calculation when running multiple threads can be seen in Table 1. Because with more threads we can use local memory optimally, the speed-up is more than double in some cases.

Table 1
Parallel speed-up of the calculation with increasing number of threads.

Threads	1	2	4	8	16	32
Time (s)	752	324	143	73	43	32
Speedup	–	2.32	5.26	10.30	17.49	23.5

The total speed-up of parallel computation running in 32 parallel threads compared to serial code is 23.5 times.

5 Extraction of the cell trajectories

The cell trajectory is represented by a series of points in space-time (discrete spatio-temporal curve) for which we prescribe the condition that there exists exactly one point in every time step $l = N_b, \dots, N_e$, $1 \le N_b < N_e \le N_\theta$. The extraction of cell trajectories is realized in two steps

- at first, we use backtracking in time by the steepest descent direction of the potential V built in (1) starting from all cell identifiers $s^l_m, m = 1, \dots, n^l_C$, detected in all time steps $l = 2, \dots, N_\theta$, recursively moving to the nearest points with strictly lower value of the potential,
- then we center all the extracted trajectories inside the 4D spatio-temporal trees by using the constrained distance function D_B in order to eliminate duplicates.

For the trajectories extraction we are again using the OpenMP parallelization and extracting each trajectory in parallel. In this case, we are using the *numactl* utility to set the memory interleaving policy, because during the backtracking we can move through the whole dataset and we cannot predict the parts of memory through which the trajectory will move. It is noteworthy that the longest part of this process is the writing of the results to disk which cannot be parallelized. The parallel backtracking and centering part lasts around 5 minutes and the writing part lasts 10 minutes.

The trajectories are stored in a text file with the following format:

- Each trajectory is defined by 1 line header containing a unique ID and the trajectory length
  ```
  --trajectory:  3632944 length:  28
  ```
- Then there are exactly *length* lines with coordinates in time and space representing the trajectory points
  ```
  184 393 90 452
  184 393 91 453
  184 393 90 454
  184 392 90 455
  184 392 90 456
  ...
  ```

5.1 Postprocessing of the extracted trajectories and cell lineage tree reconstruction

To fully reconstruct the cell lineage tree, we need to do further postprocessing on the trajectory extraction result. We have to eliminate duplicate points when two trajectories merge and continue backwards in time together and indicate mitosis on the merged trajectory. In the postprocessing step we also improve the results by reconnecting still disjointed trajectories and fix mitoses of more than two cells joining in the same time. Our postprocessing software is written in C# language and it can be easily run in multiplatform environment using .NET Framework in Windows family of operating systems or mono in unix systems without recompilation.

Let us consider a mother cell which is going to divide at time step l, i.e. at time step $l+1$ it has two descendants and in later times maybe more due to further divisions. Without losing generality, let the number of descendant of this cell in the whole image sequence be $N_d = 2^m$. Up to time l the life of the cell is represented by exactly N_d trajectories which are, however, until time l composed by the same spatio-temporal points. From the time $l+1$, the half of trajectories differs from the second half, but every half is again composed by the same points until the next division. The representation of cell life by the multiple trajectories which are partially the same does not cause any problem in visualization and/or reconstruction of a single cell or cell population movements and divisions. However, the reduction of the equal multiple parts of trajectories is necessary for the reconstruction of the cell lineage tree and is explained below.

The cell lineage tree is stored in a structure with a specific format. For each node of the cell lineage tree we store the spatio-temporal coordinates of the corresponding point, its unique global ID and the global ID of its mother in the extracted trajectory. In case of the first point of the trajectory, the mother ID is set to zero. To fill in the lineage tree structure, we subsequently take all points of the extracted trajectories, starting with the longest trajectories, and add subsequently the node representation of those points to the structure. For every trajectory we start by the first point and check whether the node corresponding to this point already exists in the structure. If it exists, it means that we have already added a trajectory which has some part equal to the current one. And also that there exists a later time after which the trajectories differ. We skip all equal points which are already represented as nodes in the lineage tree structure. Only the first different point of the current trajectory is added to the structure, together with the mother ID of the last equal point from the previously added trajectory. Using such approach we obtain the whole cell lineage tree where each node exists only once and the nodes are logically linked together in the same manner as it is in the real cell mother-daughter relation.

The lineage tree is then stored in a file with the following format

```
...
100007;8;44;-72;28;0;0;-1;-65536
100008;9;-83;19;32;0;0;-1;-1
100009;9;-36;2;39;1;0;-1;-16776961
100010;25;-79;18;33;1;9;-1;-1
...
```

where each point has global ID, local ID, three coordinates in space and one coordinate in time, local ID of the mother point and two optional parameters. In our case, the first -1 represents that the point was not validated and the last number represents the color of the point in ARGB format for visualization purposes. In this example we can see that the point with the global ID 100008 in time 0 is the mother point of the point with the global ID 100010 in time 1 since they are related using their local ID's. When the local ID of the mother equals 0, then the point has no mother and represents the starting point of the trajectory.

Now if we had perfect data, no further postprocessing would be needed. Since the real data are noisy and further errors can be introduced by wrong center detection, segmentation and near trajectory overlapping in space, we need to do additional steps to improve the results and try to fix errors. The most common errors are points where there are more than two trajectories merging and where the trajectories are disjointed because the corresponding connecting center in single time-step is missing and the time overlap in segmentation is insufficient.

For each point we calculate its movement direction using the central difference between the next and previous point of the trajectory, the average movement direction by averaging the movement direction using three points forward and backward and the average direction with respect to the points from the surrounding trajectories by averaging the direction with points in the near vicinity given by a fixed threshold. Now we can move to individual postprocessing steps.

- In the first step we disconnect all points where there are more than two trajectories merging and leave only the two nearest trajectories to merge.
- Next we try to reconnect the ending trajectories with the beginning trajectories in the next step searching for the nearest trajectory in the direction of the average cell movement gradually increasing the search radius from one to ten doxels.
- In the last step we again try to reconnect the ending trajectories with the beginning trajectories, but in the distance of two time steps, gradually increasing the search radius up to ten doxels.

Since the reconnection of trajectories in each time step is idependent, we can use parallelization. For the paralelization we are using Task Parallel Library [16] included in Microsoft .NET Framework 4.0 and higher and also supported by the mono framework. We are using the *Parallel.For* construct to process multiple time steps in parallel. Using

```
Parallel.For(0, nodes.Length - 1, i =>
                                   {
                                   ...
                                   }
```

the runtime will automatically split the calculation to all available processors. Various parallelization options can be set using the *ParallelOptions* class, which we are not using in this case. Unfortunately currently there is no way to control the NUMA node binding so the best we can do is to use the NUMA memory interleaving policy when running the postprocessing on a NUMA server. We tested the parallel speed-up on standard quad-core desktop CPU intel core i7-4770k. The serial code took 413 seconds and the parallel version took 144 seconds, which represents the speed-up of 2.87. After this postprocessing we have the tracking results ready for a validation.

6 Validation of the results of numerical experiments on real Zebrafish embryogenesis data with ground truth data

For the testing of the method and for the reconstruction of the cell lineage tree we use one of the dataset produced by BioEmergences platform [17] with the acquisition step $d\theta = 50$ seconds, N_θ, number of time steps, is equal to 480 and dimension of every 3D image is 512x512x104 voxels. The real voxel size is $dx_1 = dx_2 = dx_3 =$ 1.33 micrometer in every spatial direction. In the last time step $N_\theta = 480$ the biologist marked cell populations forming various organs of the embryo and we can use this information to visualize the formation of these organs in time, Fig. 5. Using developed approach we can track those cell populations. Since the size of a single 3D image is about 27MB, for 480 time steps we deal with about 13 GB of raw data which requires usage of high-performance parallel computing (HPC) facilities especially when designing spatially 4D numerical algorithms.

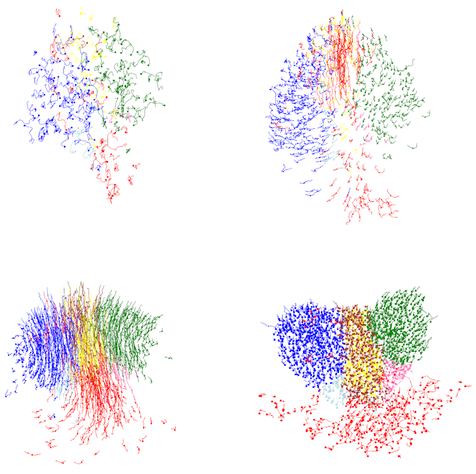

Figure 5
Visualization of the organ formation in time steps 1, 160, 320 and 480.

Before tracking, all 3D images of the processed data were filtered by 10 steps of geodesic mean curvature flow (GMCF) model [2, 3] and the cell nuclei identifiers were detected by 15 steps of level set center detection (LSCD) algorithm [4, 3]. The cell nuclei were segmented using the generalized subjective surface (GSUBSURF) method [3]. From several millions of cell identifiers we built the 4D segmentation and then the cell trajectories were extracted. The correctness of the mother-daughter cell links for the first dataset was validated using the ground truth data and the results are presented in Table 2 and Fig. 6.

The ground truth data (GTD) contains 38797 manually checked links between cells in time during all 480 time steps which we can use to validate the correctness of cell links in our results. It also contains 71 valid mitoses that can be used for the accuracy of mitosis detection. Since the mitoses in ground truth data and in our tracking can be shifted by a few time steps in time, we are using the following procedure for the validation.

- In the first step, we find the corresponding trajectory points in our results with points in the ground truth data by finding the points with the same coordinates or with the shortest distance to the ground truth data points.
- Then we can easily check for the correctness of the cell links by checking if the link between two points in ground truth data is the same as the link between the corresponding points in our results. If it is the same we have a correct mother-daughter link, if the link is missing or we have the link to a different corresponding cell, then we have a wrong mother-daughter link.
- For the mitosis detection, we have three different possibilities:
  - The mitosis is in the same time step in the ground truth data and our results. This is the most simple case and can be validated easily just by checking the links between points.
  - The mitosis occurs later in our dataset than in the ground truth data, but on the same trajectory. We call this case ”forward mitosis” and we move forward by the links between cells, checking the links in daughter cells after mitosis in the ground truth data and corresponding cells in our results looking for mitosis in the next five time steps.
  - In the third case, the mitosis occurs later in the ground truth data than in our dataset, but on the same trajectory. We call this case ”backward mitosis” and we move backward through corresponding points and search for mitosis in our results in the previous five time steps.

Using this approach we can validate the results of our tracking algorithm by comparing it with the ground truth data. We are interested in two parameters, correctness of the cell links and the number of correctly detected mitoses.

To tune the tracking results of the 4D segmentation approach (ellipsoids created from diameters of real cell nuclei segmentations) we adjust two parameters mentioned in section 3.1, S and α. First, we can expand or shrink the nuclei radii used for building the 4D segmentation. By comparison with ground truth data we concluded that the best results for these two datasets were obtained when we shrink the radii by $S = -0.5$, cf. Fig. 6. The second parameter is α, used in the construction of the potential V. We try to find the optimal α to obtain the best ratio of correct links and detected mitoses. We tested the tracking for $\alpha \in [0.5, 4]$ and obtained the best results around $\alpha = 1.6$, see Table 2, where we achieved 94% accuracy of cell link detection and 21% of mitosis detection. When comparing the number of detected mitoses (Table 3) we can see that it is decreasing very fast and around $\alpha = 3.2$, where we have 96.5% correct links, we have only 3 correct mitosis, which is only 4% of all 71 mitosis in the ground truth data. The problem is caused by the fact, that the cell nuclei right after the mitosis ”jump” away from each other, thus creating large gaps in the 4D segmentation. Improving the mitosis detection is difficult and

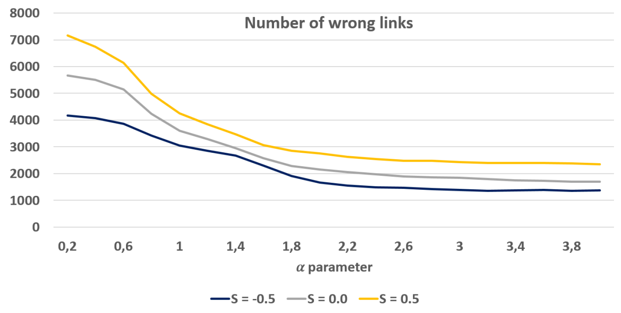

Figure 6
Number of wrong links in tracking compared to ground truth data, depending on α and S. With α increasing towards 2, the number of wrong links is decreasing, then it stabilizes and the best result is obtained for $S = -0.5$, cf. also Table 2.

still an open problem. Possible direction in this research would be to use membrane images for creating the 4D segmentation where no jumps during the cell split occur. The quality of membrane images by a confocal microscopy is still not sufficient for this puprose, but one could use simulated cell membranes by using approximate Voronoi shapes constructed around the cell identifiers, which gives promising results.

Table 2
Comparison of the tracking result obtained with segmentation from real nuclei segmentations with ground truth data depending on α, with $S = -0.5$.

α	**Correct mother links**	**Correct daughter links**	**Wrong mother links**	**Wrong daughter links**
0.4	34856	34719	3935	4072
0.8	35510	35371	3280	3419
1.2	36064	35946	2726	2844
1.6	36551	36493	2240	2298
2.0	37125	37126	1668	1667
2.4	37288	37309	1506	1485
2.8	37349	37386	1448	1411
3.2	37402	37437	1395	1360
3.6	37357	37415	1440	1382
4.0	37377	37434	1420	1363

For manual visual validation we created a tool to visualize the result alongside the ground truth data. One can display the tracking with the trajectories with optional length, highlight trajectory start and end, cell mitosis and the volume rendering of original data along with the 2D slices can also be displayed.

Here we present the Figures 7-9 showing our tool for visual validation. In Fig. 7 one can see the visualization of the volume rendering of the original cell along with the trajectories from the ground truth data (left) and our tracking result. Here the

Table 3
Comparison of the correctly detected mitosis with segmentation from real nuclei segmentations with ground truth data depending on α, with $S = -0.5$.

α	**Correctly detected mitosis**	**Correctly detected forward mitosis**	**Correctly detected backward mitosis**	**Total correctly detected mitosis**
0.4	10	0	6	16
0.8	6	0	10	16
1.2	5	0	14	19
1.6	3	0	12	15
2.0	2	0	4	6
2.4	2	0	1	3
2.8	2	0	1	3
3.2	2	0	1	3
3.6	2	0	1	3
4.0	2	0	2	4

trajectory in our result is the same as the trajectory manually validated by biologists even for a long time interval (20 time steps forward and backward in this case). In Fig. 8 and Fig. 9 the cells before mitosis are displayed. One can see that the mitosis will occur since the trajectory is divided. Correctly detected cell mitosis is displayed in Fig. 8. In Fig. 9 on the right, in our result, the mitosis is not detected and only one branch of the trajectory is registered.

Conclusions

In this paper we presented an algorithm for the cell tracking on large-scale 3D+time microscopy data. We created parallel implementation of the 4D Rouy-Tourin scheme to speed up the data processing and then we applied this method to complex stages of the zebrafish early embryogenesis microscopic images. Using our automatic post-processing and validation workflow we can easily compare the tracking with ground truth data. Using the visualization tool we can visually verify the correctness by displaying the ground truth data along with our result and volume rendering of the original images.

Acknowledgement

This work was supported by Grants APVV-15-0522 and VEGA 1/0608/15.

References

[1] Y. Chen, B. C. Vemuri, L. Wang, Image denoising and segmentation via non-linear diffusion, Comput. Math. Appl. 39 (2000) 131–149. `doi:10.1016/S0898-1221(00)00050-X`.

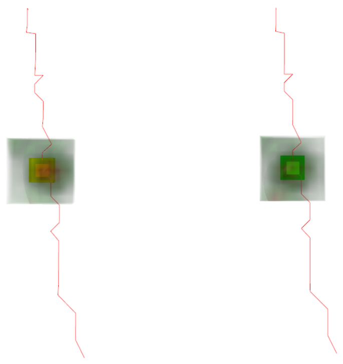

Figure 7
Visualization of a single cell trajectory from our manual validation tool. Data from the ground truth data is displayed on the left and our tracking result is on the right. Even a long trajectory is reconstructed correctly.

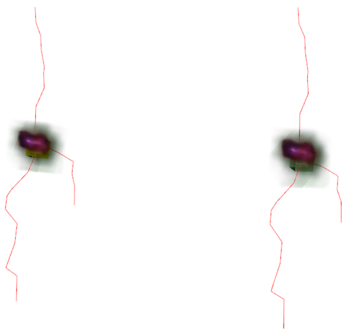

Figure 8
Visualization of a single cell trajectory from our manual validation tool. Data from the ground truth data is displayed on the left and our tracking result is on the right. The cell division is detected correctly.

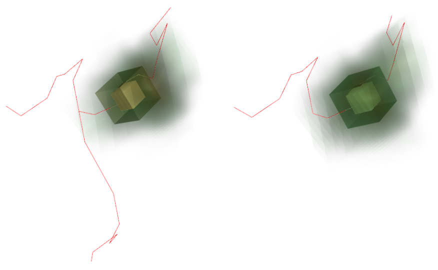

Figure 9
Visualization of a cell with dividing trajectory from our manual validation tool. Data from the ground truth data is displayed on the left and our tracking result is on the right. Only one branch of the divided cell is registered in our tracking result.

[2] Z. Krivá, K. Mikula, N. Peyriéras, B. Rizzi, A. Sarti, O. Stašová, 3d early embryogenesis image filtering by nonlinear partial differential equations, Medical Image Analysis 14 (4) (2010) 510–526. doi:10.1016/j.media.2010.03.003.

[3] E. Faure, T. Savy, B. Rizzi, C. Melani, M. Remešíková, R. Špir, O. Drblíková, R. Čunderlík, G. Recher, B. Lombardot, M. Hammons, D. Fabrèges, L. Duloquin, I. Colin, J. Kollár, S. Desnoulez, P. Affaticati, B. Maury, A. Boyreau, J. Y. Nief, P. Calvat, P. Vernier, M. Frain, G. Lutfalla, Y. Kergosien, P. Suret, R. Doursat, A. Sarti, K. Mikula, N. Peyriéras, P. Bourgine, An algorithmic workflow for the automated processing of 3d+time microscopy images of developing organisms and the reconstruction of their cell lineage, Nature Communications 7, Article number: 8674. doi:10.1038/ncomms9674.

[4] P. Frolkovič, K. Mikula, N. Peyriéras, A. Sarti, A counting number of cells and cell segmentation using advection-diffusion equations, Kybernetika 43 (6) (2007) 817–829.

[5] A. Sarti, R. Malladi, J. A. Sethian, Subjective surfaces: A method for completing missing boundaries, Proceedings of the National Academy of Sciences of the United States of America 97 (12) (2000) 6258—-6263. doi:10.1073/pnas.110135797.

[6] K.Mikula, Peyriéras, M. Remešíková, A.Sarti, 3d embryogenesis image segmentation by the generalized subjective surface method using the finite volume technique, Finite Volumes for Complex Applications V, Problems & Perspectives (Eds. R.Eymard, J.M.Herard), ISTE and WILEY, London (2008) 585–592.

[7] K. Mikula, N. Peyriéras, R. Špir, Numerical algorithm for tracking cell dynamics in 4d biomedical images, Discrete and Continuous Dynamical Systems - Series S 8 (5) (2015) 953–967. doi:10.3934/dcdss.2015.8.953.

[8] K. Mikula, R. Špir, M. Smíšek, E. Faure, N. Peyriéras, Nonlinear pde based numerical methods for cell tracking in zebrafish embryogenesis, Aplied Numerical Mathematics 95 (2015) 250–266. doi:10.1016/j.apnum.2014.09.002.

[9] K. Mikula, R. Spir, N. Peyrieras, Cell lineage tree reconstruction from time series of 3d images of zebrafish embryogenesis, Submitted to MCBMIIA2016.

[10] F. Amat, W. Lemon, D. P. Mossing, K. McDole, Y. Wan, K. Branson, E. W. Myers, P. J. Keller, Fast, accurate reconstruction of cell lineages from large-scale fluorescence microscopy data, Nature Methods 11 (2014) 951–958. doi:10.1038/nmeth.3036.

[11] P. Bourgine, P. Frolkovič, K. Mikula, N. Peyriéras, M. Remešíková, Extraction of the intercellular skeleton from 2d microscope images of early embryogenesis, Lecture Notes in Computer Science 5567 (Proceeding of the 2nd International Conference on Scale Space and Variational Methods

in Computer Vision, Voss, Norway, June 1-5,2009) , Springer (2009) 38–49doi:10.1007/978-3-642-02256-2_4.

[12] C. Zanella, M. Campana, B. Rizzi, C. Melani, G. Sanguinetti, P. Bourgine, K. Mikula, N. Peyrieras, A. Sarti, Cells segmentation from 3-d confocal images of early zebrafish embryogenesis, EEE Transactions on Image Processing 19 (3) (2011) 770–781. doi:10.1109/TIP.2009.2033629.

[13] E. Rouy, A. Tourin, Viscosity solutions approach to shape-from-shading, SIAM Journal on Numerical Analysis 29 (3) (1992) 867—-884. doi:10.1137/0729053.

[14] MPI Forum, http://mpi-forum.org/, accessed: 2017-03-29.

[15] OpenMP Home page, http://www.openmp.org/, accessed: 2017-03-29.

[16] Microsoft, Task parallel library, https://msdn.microsoft.com/en-us/library/dd460717, accessed: 2016-07-25.

[17] BioEmergences platform web site, http://bioemergences.iscpif.fr/bioemergences/index.php, accessed: 2017-03-29.