Computer Physics Communications 227 (2018) 8–16

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

A high-speed tracking algorithm for dense granular media

Mauricio Cerda [a,b,*], Cristóbal A. Navarro [c], Juan Silva [d,f], Scott R. Waitukaitis [e], Nicolás Mujica [f], Nancy Hitschfeld [d]

[a] Anatomy and Developmental Biology Program, Institute of Biomedical Sciences, Facultad de Medicina, Universidad de Chile, PO Box 70031, Santiago, Chile
[b] Biomedical Neuroscience Institute, Independencia 1027, Santiago, Chile
[c] Instituto de Informática, Facultad de Ciencias de la Ingeniería, Universidad Austral de Chile, General Lagos 2086, Valdivia, Chile
[d] Departamento de Ciencias de la Computación, Facultad de Ciencias Físicas y Matemáticas, Universidad de Chile, Avenida Beauchef 851, Santiago, Chile
[e] Leiden Institute of Physics, Leiden University, Niels Bohrweg 2, 2333 CA Leiden, Netherlands
[f] Departamento de Física, Facultad de Ciencias Físicas y Matemáticas, Universidad de Chile, Avenida Blanco Encalda 2008, Santiago, Chile

ARTICLE INFO

Article history:
Received 13 September 2017
Received in revised form 15 January 2018
Accepted 10 February 2018
Available online 16 February 2018

Keywords:
Particle tracking
Peak detection
GPU computing
Granular media

ABSTRACT

Many fields of study, including medical imaging, granular physics, colloidal physics, and active matter, require the precise identification and tracking of particle-like objects in images. While many algorithms exist to track particles in diffuse conditions, these often perform poorly when particles are densely packed together—as in, for example, solid-like systems of granular materials. Incorrect particle identification can have significant effects on the calculation of physical quantities, which makes the development of more precise and faster tracking algorithms a worthwhile endeavor. In this work, we present a new tracking algorithm to identify particles in dense systems that is both highly accurate and fast. We demonstrate the efficacy of our approach by analyzing images of dense, solid-state granular media, where we achieve an identification error of 5% in the worst evaluated cases. Going further, we propose a parallelization strategy for our algorithm using a GPU, which results in a speedup of up to 10× when compared to a sequential CPU implementation in C and up to 40× when compared to the reference MATLAB library widely used for particle tracking. Our results extend the capabilities of state-of-the-art particle tracking methods by allowing fast, high-fidelity detection in dense media at high resolutions.

1. Introduction

Extracting quantitative information from image data is at the heart of scientific fields ranging from biology [1,2] to physics [3,4]. Often, the first step in image analysis is the identification of objects of interest, *i.e.* "particles". As examples, one can look to the identification of stars in telescope images, the tracking of individual cells in a biomedical experiment, or the tracking of grains of sand or other media in table top physical systems. While many algorithms exist to identify particles that are diffusely distributed throughout an image, e.g. stars in a telescope image, there are few algorithms that are able to reliably identify particles that are in close contact, as would be encountered in a dense population of living cells or a closely-packed colloidal system.

One particularly interesting case, which has been used to some degree as a model system, is the tracking used to study the dynamics of confined, quasi-two-dimensional granular systems. The typical experimental setup consists of granular particles placed in a shallow, enclosed container that is vibrated vertically. Measurements are performed by taking images or videos with a camera from above. Physically, this system is of interest because it undergoes state phase transitions (*e.g.* liquid to solid) when energy is injected into it during vibration [5,6]. The confined geometry has a great advantage because it permits the observation of both individual trajectories and collective behavior, which enables one to study both the microscopic and macroscopic dynamics. Computationally, this system creates a complex and challenging tracking situation because following particles through the phase transition requires particle identification in both diffuse and dense conditions.

From a computational point of view, this kind of identification and tracking analysis is performed with two main post-processing strategies: (i) particle-image velocimetry (PIV) and (ii) particle tracking (PT) [7]. PIV has the advantage that it is capable of extracting motion vector fields from images without requiring the identification of each particle. This is achieved by looking at the correlation of small windows of the field of view from one image to the next. For computer vision, this kind of method corresponds to the so called "optical flow" family of algorithms [7]. The main disadvantage of PIV methods is that the information about individual particles is lost, thus statistical measurements such as

* Corresponding author at: Anatomy and Developmental Biology Program, Institute of Biomedical Sciences, Facultad de Medicina, Universidad de Chile, PO Box 70031, Santiago, Chile.
E-mail address: mauriciocerda@med.uchile.cl (M. Cerda).

https://doi.org/10.1016/j.cpc.2018.02.010
0010-4655/

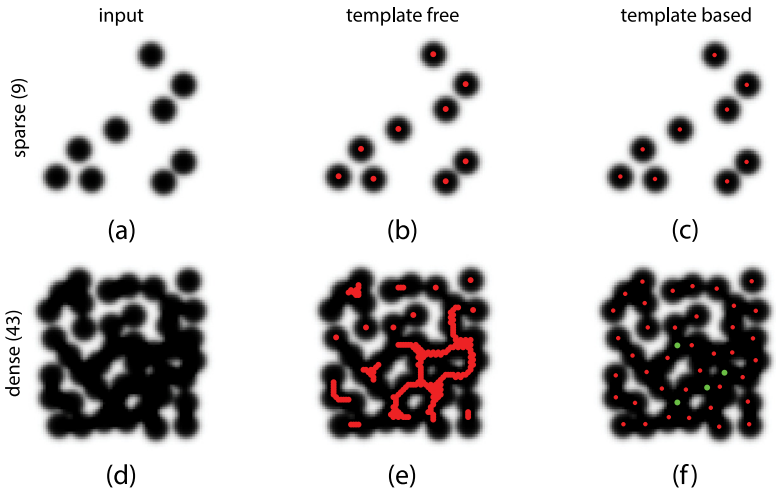


Fig. 1. Comparison of PT segmentation. The two scenarios: sparse (upper row), and dense (lower row) of PT are shown using synthetic images. The input column shows the granular media input image. Red dots show particle detection result for template free (column 2) and template based algorithm (column 3). Green dots show missing particle for the template based algorithm. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

velocity profiles are only approximated [8,9]. On the other hand PT methods have the advantage of extracting information at the single object level, being a direct and more precise way to look at the microscopic details [10]. However, PT has the disadvantages that it comes at significant computational cost and fails in situations where particles are so closely packed together that individuals are difficult to detect.

This work proposes an automatic PT method that can handle in-contact media as well as accelerate its performance by offloading its most computationally-intensive tasks to a Graphics Processing Unit (GPU). GPU computing is a useful tool for the field of computational physics as it can produce up to an order of magnitude of speedup on data-parallel problems when compared to a CPU [11,12]. We illustrate the strategy used to accelerate the algorithm with GPU computing and benchmark its effectiveness through analysis of synthetic images that imitate the well-studied yet computationally challenging physical system that was first studied by Olafesn and Urbach [13]: quasi-two-dimensional grains undergoing a phase transition from a diffuse-gas like state to a dense solid-like state. This system serves as an ideal candidate for this purpose as its transitions from a relatively easy tracking problem in the gas state, where particles are fully separated, to a nearly intractable one in the solid state, where particles clump together to create large borderless units. For a review of the physics behind this system see [14]. The proposed strategy allows us to handle experiments that are dense, long in duration, and recorded at high resolution all in a single computer. The paper is organized by first presenting a review of common PT methods in Section 2. Section 3 presents the new algorithm to handle in-contact particles, and Section 4 presents the GPU implementation of the proposed algorithm. Finally, in Section 5 we detail the results and in Section 6 we summarize the main findings.

2. Related work

Particle tracking problems have been extensively addressed in the literature and are typically separated into three stages: segmentation, correspondence, and parameter extraction [7]. The first stage aims to identify the objects in the image, from simple dot-like features to complex shapes. The correspondence stage attempts to match identified objects from one frame (or set of frames) to the next. At the end the set of identified trajectories are studied, where the parameter extraction stage obtains descriptors such as trajectory length, mean speed, directionality, and also domain-specific descriptors such as mean squared displacement, bond-orientational parameter, structure factor, velocity correlations and so forth. In the two dimensional granular system we work with here, correspondence is simple as data is obtained from controlled conditions with regard to lighting and camera parameters, thus a closest neighbor criterion has been reported as satisfactory [15]. For this reason, we focus our efforts on the segmentation stage, which has a higher degree of difficulty.

In the case of the system we study, the segmentation stage is a straightforward process if objects are sparse and have high contrast and uniform shape, such as those shown in the upper row of Fig. 1. This task becomes much more difficult in dense configurations, e.g. as shown in the lower row of Fig. 1. Within this context, several segmentation methods have been propose. These can be classified in as those that involve a template model and those that do not, which we detail in the following subsections.

2.1. Template Free PT methods (TFPT)

Perhaps the simplest strategies for performing image segmentation are the template free PT methods (TFPT), which are based on the detection of local intensity maxima (or minima) of the image. To illustrate the key idea, a practical realization of the algorithm is shown in Fig. 1b, e. The procedure has 3 steps: filtering (Gaussian filter with parameters $\sigma = 2.5$), a regional maximum operation (*imregionalmax* MATLAB function), and finally a step to reduce each object found to a single location. By construction, the location of a particle's center is limited to a precision of 1 pixel. The main drawback of this approach is that when particles are in-contact, their centers are no longer local maxima/minima, as shown in Fig. 1e. This causes most TFPT methods to systematically overdetect particles in high-density regions, as well as miss particles in other regions.

For an image of N pixels TFPT methods have a computational complexity of $O(N)$, where N is the number of image pixels [16]. Thus, for a movie of T frames, the overall complexity is $O(TN)$, which makes the algorithm fast and suitable for parallel architectures. The main issue with TFPT methods is that they fail in dense scenarios. The difficulty in these cases is that particles can be in contact, thus there is no measurable decrease in the image intensity between neighboring peaks. Another issue is that, due to the lack of geometrical constraints, there is no built-in crosscheck for the minimum spacing between particles, although this can be added as a post-processing step.

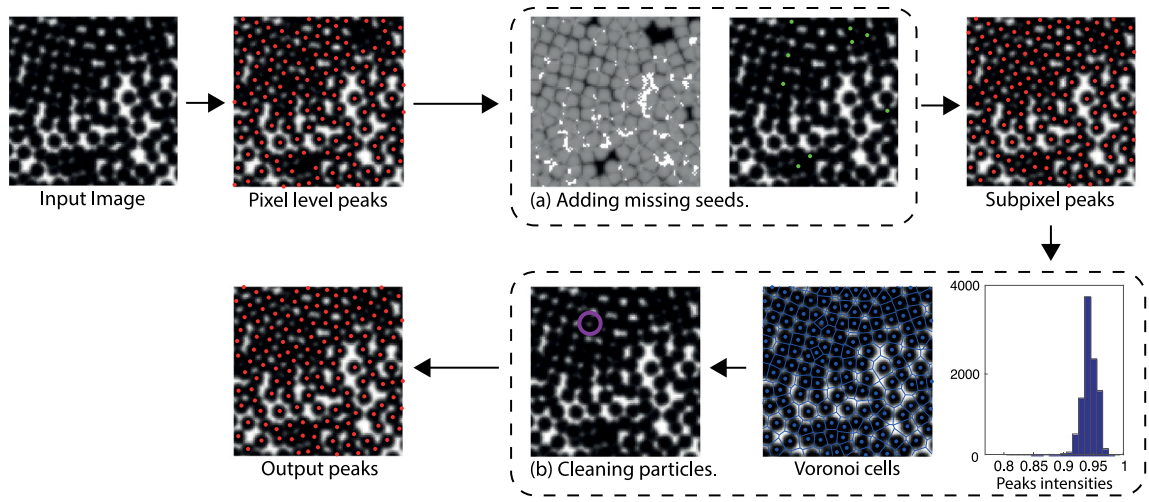


Fig. 2. Proposed algorithm. Starting from the input image (from a confined quasi-two-dimensional experiment [5,6]) a first pixel-level peak detection is performed, followed by adding missing seeds in the difference image (peaks added in green). After peaks are refined with subpixel precision, the particle cleaning process is applied to remove false positive peaks by removing identifications of abnormally high intensity, thresholding from data histogram distribution, or associated to small Voronoi cells, obtaining the final output peaks. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

2.2. Template Based PT methods (TBPT)

Template based particle tracking (TBPT) methods offer a more precise alternative to the TFPT methods, but come at the cost of increased complexity. One potent and widely-used variant of this family was proposed by Franklin & Shattuck [15], which we choose in this work to base our comparisons. This serves as a good standard because the Franklin & Shattuck algorithm starts by simplifying the tracking problem and assuming that all particles have the same unknown diameter (D) and can be fit to a 2D hyperbolic intensity profile. The position of a particle is then estimated from the parameters of this fit.

Concretely, one often uses the hyperbolic particle intensity profile

$$I_p(\vec{x}; D, \omega) = \frac{1}{2}[1 - \tanh(\frac{|\vec{x}| - D/2}{\omega})], \tag{1}$$

where D is the particle diameter and ω defines the 'sharpness' of its border. The algorithm starts by generating a filtered image

$$\tilde{\chi}_1^2(D, \omega) = \frac{I^2 * I_p - 2I * I_p^2}{\langle I_p^3 \rangle} + 1, \tag{2}$$

where I is the input image, $*$ is the convolution operation, and $\langle a \rangle >= 1 * a$. Similar to TFPT methods, local maxima with pixel precision are obtained from $\tilde{\chi}_1^2$. To achieve sub-pixel accuracy, particles positions are estimated with a minimization process of the target function χ_2^2 defined as follows:

$$\chi_2^2(\vec{x_n}; D, \omega) = \int \left[I(\vec{x}) - I_c(\vec{x}, \vec{x_n}; D, \omega)\right]^2 d\vec{x}, \tag{3}$$

$$I_c(\vec{x}, \vec{x}_n) = \sum_n W_n(\vec{x}) I_p(\vec{x} - \vec{x}_n; D, \omega), \tag{4}$$

where $W_n(\vec{x})$ is a weight function with value one inside the Voronoi [17] area of particle n and zero elsewhere, and $I(\vec{x})$ the input image. The target function χ_2^2 can be minimized using the Gauss–Newton iterative algorithm [18] from the initial approximated solution using Eq. (3). In this case the minimization condition is given by $\partial \chi_2^2(\vec{x}_n^*; D^*, \omega^*)/\partial \vec{x}_n = 0$. Numerically, at each iteration the particle positions are corrected by Δ

$$\Delta = -\mathbf{H}^{-1}\vec{g}, \tag{5}$$

where $\mathbf{H}$ is the Hessian Matrix and $\vec{g}$ the gradient vector. Finally, to link particles from one frame to the next, the total displacement is minimized. The combinatorial space is greatly simplified if a maximum traveled distance $D/2$ is imposed, a condition directly related to the experimental particle speed and camera sampling rate.

The TBPT implementation by Franklin & Shattuck [15] is shown in black in the pseudocode of the Algorithm 1 . In terms of computational complexity (for an image of N pixels and P particles) the algorithm is dominated by the minimization of χ_2^2 by Gauss–Newton method stage of $O(PN)$. Thus, for a movie of T frames, the overall complexity is $O(TPN)$. The algorithm checks for minimal distance, but still fails to detect objects that are closely packed as the seeds of the iterative process are still local maxima, as shown in Fig. 1f. The crux of this failure is that in dense regions there are multiple solutions for the minimization of the target functions, $\tilde{\chi}_1^2$ and χ_2^2. Compounding this disadvantage is the fact that the algorithm experiences considerable slowdown when implemented in a sequential way.

3. Proposed tracking algorithm for dense granular media

The TFPT and TBPT methods just described fail to correctly detect particles in high density configurations, were objects are in direct contact. In this section we propose our new solution for this problem.

3.1. Adding missing seeds

The key consideration for our approach is the fact that the TBPT segmentation problem comes from missing peaks in the initial stage. We add an iterative process to the TBPT algorithm over the target function remainder $\Delta I(\vec{x})$, defined as

$$\Delta I(\vec{x}) = \left[I(\vec{x}) - I_c(\vec{x}, \vec{x_n}; D, \omega)\right]^2. \tag{6}$$

With this step, the intent is to recover missing seeds by looking at local maxima with pixel precision in Eq. (6), as illustrated in Fig. 2a. As a result, the new method is able to deliver peak detection in high-density scenarios at sub-pixel accuracy. However, this solution can deliver false positives, an issue that we handle in a cleaning stage we detail in the following subsection.

3.2. Cleaning particles

The proposed solution to add missing seeds delivers false positive detections that must be handled. Typically these false positives occur in dense areas (solid phase) and are associated with bright pixel intensities, as highlighted in purple in Fig. 2b. To identify particles in the solid phase, we compute the Voronoi diagram for all peaks and then use the associated Voronoi cell area as an estimation of each particle's area. If a given particle area is lower than a threshold area A ($A = 50$ pix^2 for our images with $D = 9$ pix and $w = 1.49$ pix), then the particle is identified as belonging to the solid phase. In this case, if its intensity is higher than a certain threshold (τ), then it is filtered out. We automatically compute the intensity threshold τ as $\tau = \mu - 1.5\sigma$, where μ and σ are the mean and standard deviation of all particle intensities (at the center pixel). The introduction of new parameters for this filtering increases the algorithm's complexity. However, the minimal area A can be estimated from the system itself, and the image intensity τ can be estimated reliably from the data as explained. With this approach, the segmentation process becomes robust even in highly packed configurations. The proposed algorithm is summarized in Algorithm 1 (red text), and illustrated in Fig. 2a, b.

3.3. Example application and implications for physical calculations

In order to demonstrate the effectiveness of the proposed algorithm, and in particular the implications on subsequent calculations based on particle positions, we now perform a test. We compare results obtained from particles detected with our algorithm and those obtained from Shattuck's et al. [15]. With the identified particles from each of these, we then compute the static structure factor—a physical quantity that gives direct insight into the spatial arrangement of the particles.

The experimental setup and procedures are similar to those presented in [5,6]. The surface filling fraction is $\phi = N\pi D^2/4L^2 \approx 0.92$. Particles are stainless steel spheres of diameter $D = 1$ mm. The number of particles is $N \approx 11\,700$. The cell's lateral dimensions are $L_x = L_y \equiv L = 100D$, and its confinement height is $h = (1.94 \pm 0.02)D$. The whole system is forced sinusoidally with displacement $z(t) = A\sin(\omega t)$, with $f = \omega/2\pi = 80$ Hz and normalized acceleration $\Gamma = A\omega^2/g = 4.50 \pm 0.01$, where g is the gravitational acceleration.

The structure factor is a measure of the intensity of density fluctuations in Fourier space. The particle density field is described as

$$\rho(\vec{r}, t) = \sum_{j=1}^{N} \delta(\vec{r} - \vec{r}_j(t)), \tag{7}$$

where δ is the Dirac delta function. Particle positions $\vec{r}_j(t)$ in the plane (x, y) are determined for each time t. Experimentally, there is no access to the z coordinate. Thus, the 2D microscopic density field Fourier components are

$$\widehat{\rho}(\vec{k}, t) = \int d^2r\, e^{i\vec{r}\cdot\vec{k}} \rho(\vec{r}, t) = \sum_{j=1}^{N} e^{i\vec{k}\cdot\vec{r}_j(t)}. \tag{8}$$

The static structure factor $S(\vec{k})$ is then defined as

$$S(\vec{k}) = \frac{\langle |\widehat{\rho}(\vec{k}, t) - \langle \widehat{\rho}(\vec{k}, t)\rangle|^2 \rangle}{N}, \tag{9}$$

where $\langle\ \rangle$ denotes time averaging. In general, $\langle \rho(\vec{k})\rangle \neq 0$ due to inhomogeneities induced by boundary conditions. The wave vectors are computed from $\vec{k} = \pi(n_x\hat{\imath} + n_y\hat{\jmath})/L$, where $n_x, n_y \in \mathbb{N}$. We have previously shown that the system is isotropic, i.e. $S(\vec{k}) = S(k = |\vec{k}|)$ [6].

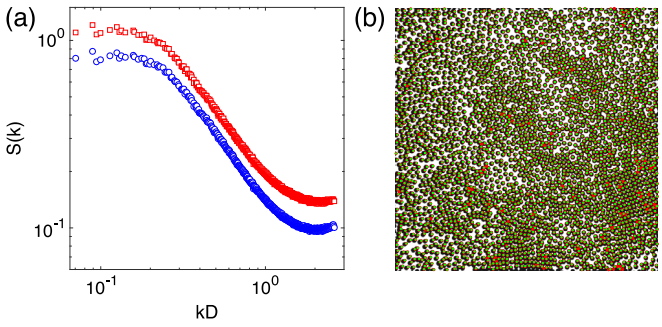


Fig. 3. (a) Structure factor comparison. Red squares represent $S(k)$ computed from positions obtained form the original Shattuck algorithm; blue circles correspond to the computations carried out with the positions detected with our improved algorithm. (b) Image of about a quarter of the experimental system ($L/2 \times L/2$). The green dots show the particle positions detected with our algorithm, whereas red dots show those obtained with Shattuck's version. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

In Fig. 3a we present $S(k)$ computed using positions obtained with our algorithm and Shattuck's. A total of 6000 images were acquired at 10 fps, which provides sufficient time between images to ensure decorrelation, thus improving the statistics (under these conditions the time average $\langle\ \rangle$ is equivalent to an ensemble average). Although qualitatively both curves are very similar, for a given wavenumber, $S(k)$ computed from positions obtained via Shattuck's algorithm is systematically larger than the one computed via ours. This systematic difference is about 40%, which is quite large when quantitative information is required from the static structure factor. As an example, for fluids at equilibrium, it is well established that in the low wavenumber limit,

$$S(k \to 0) = \frac{\chi_T}{\chi_T^0}, \tag{10}$$

where the fluid's isothermal compressibility is χ_T and $\chi_T^0 = 1/(\rho k_B T)$ is the one of the ideal gas [19]. We attribute such systematic over-estimation of $S(k)$ by the number of additional particles that Shattuck's algorithm incorrectly detects. For the set of images that we have studied, the number of additional false particles is about 250 per image. In Fig. 3b we show an example of the differences in particle detection. Most of the particles are well detected by both algorithms, but Shattuck's finds more false particles. This is acutely visible in dense regions, where the older algorithm detects false positives in the regions between neighboring particles.

4. Parallelization of the proposed algorithm

Particle tracking algorithms have a high computational cost when images are large in size and have a high quantity of particles, as in the granular media experiments reported in [5,6] that serve as the model for the algorithm benchmark we present here. In this section, we detail a GPU parallelization scheme for the proposed TBPT algorithm to improve the $O(TPN)$ time complexity. A preliminary implementation and results were reported in [20].

There are two ways of approaching the parallelization of the TBPT algorithm: (1) by simultaneous image processing and (2) by simultaneous processing within an image. The first approach, where all T images can be simultaneously processed in $O(PN)$ time, is not detailed in this section, but it is worth mentioning that it is a *pleasingly parallel* scenario that can be achieved by running T instances of the TBPT algorithm on one or more devices. If the number of GPUs is less than T then one can opt to run multiple instances on each GPU, as the job scheduler is capable to execute

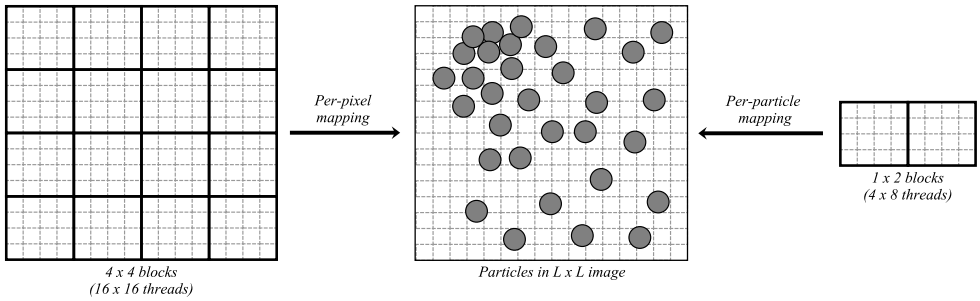


Fig. 4. Mapping schemes. On the left, a per-pixel mapping acts by assigning a fixed number of pixels to each thread. The center illustrates a 32 particle and 16 × 16 pixels image example. On the right, a per-particle mapping assigns particle locations to each thread.

input : A *movie* composed of T frames, a maximum iterations number (*maxIterations*), a minimum chi2 variation (*minDeltaChi2*). The detected object must have a minimum brightness *th* and area *thArea*.
output: An array of 2D positions of N in T frames (*positions*).

```
for t ← 1 to T do
    image ← movie[t];
    chi1 ← buildingTargetFunction1(image, D, ω);
    positions(t)← getPeaks(1/chi1);
    npeaks ← 1;
    while npeaks > 0 do
        chi2 ← buildingTargetFunction2(image,
        positions(t), D, ω);
        newPositions ← getPeaks(1/chi2);
        npeaks ←length(newPositions);
        positions(t)← [positions(t) newPositions];
    end
    i ← 0;
    deltaChi2 ← minDeltaChi2;
    while i < maxIterations and deltaChi2 >= minDeltaChi2
    do
        chi2 ← buildingTargetFunction2(image,
        positions(t), D, ω);
        [Δx, Δy] ←newtonCenter(chi2, D, ω, positions);
        positions(t)←positions(t)+[Δx, Δy];
    end
    for i ← 0 ; i < length(positions(t)) ; i ← i+1 do
        if image(positions(t, i)) > th or
        voronoiArea(positions(t, i)) < thArea then
            delete(positions(t, i));
        end
    end
end
trajectories ←tracking(positions,D);
```

Algorithm 1: Franklin & Shattuck tracking (black) and proposed high density algorithm (red).

multiple concurrent kernels. This is possible given current GPU architectures can run up to 32 concurrent kernels [21].

The parallelization proposed in this work focuses on the second approach where the algorithm has to be re-adapted to the GPU computing model in order to scale performance with the image size. From a global perspective, two types of GPU mappings can be distinguished when doing simultaneous processing within an image; (1) per-pixel mapping and (2) per-particle mapping. For the first one, the amount of parallelism scales with N (*i.e.*, the number of pixels), while for the second approach parallelism scales with the number of tracked particles P. Fig. 4 illustrates both mapping schemes.

The grids of Fig. 4 correspond to parallel spaces, *i.e.*, where threads are contained before being mapped onto the data domain. The dimensions of a grid are specified in terms of blocks, which for the purpose of this example has blocks[1] of 4 × 4 threads. Since $P \ll L^2$, one can expect that for large images the per-pixel tasks will provide most of the computing acceleration. The per-particle grid can also be a one dimensional grid of 1 × 8 blocks. During a full particle tracking pass, several tasks will require one or the other mapping scheme in order to take advantage of GPU parallelism.

4.1. Choosing tasks for parallelization

The tasks chosen for parallelization are the ones involved in the minimization of $\tilde{\chi}_1^2$ and χ_2^2 as they have higher relative computational cost. A performance profiling process was performed on the MATLAB and C implementations of Shattuck's method and the proposed high density algorithm. The relative computational cost for each task is obtained as the fraction of time employed from the total time excluding I/O and initialization routines (*i.e.*, time is measured once data is main memory). Fig. 5 shows the relative sequential times for all three implementations for two different image sizes.

From the relative costs, one can note that the distribution of percentages of the proposed algorithm and those of Shattuck's display significant differences regarding their MATLAB and C implementations. In the MATLAB Shattuck (MS) results with an image size of $N = 1000 \times 1000$, 30% is dedicated to the *gengrid* task, 40% to newton-center, 20% to getpeaks. On the other hand, in the CS results of $N = 1000 \times 1000$ gengrid is less significant having less than 10%, but chi2diff and convolution increase to 16% and 70%, respectively. The reason for such differences comes in great part from the fact that several MATLAB routines decrease their cost when being implemented in C, since it can handle for loops and other instructions in a much more efficient way. For the high-density algorithm, one can note a similar change from MATLAB (MHD) to C (CHD), where the costs of chi2diff and convolution become the most significant ones in the C implementations for $N = 1000 \times 1000$.

For $N = 4000 \times 4000$, the scenario of CHD is more balanced than the CS one. The tasks chi2diff, getpeaks, newtonc, convolution and gengrid have comparable costs among each other, while in the case of CS the getpeaks function is the one that takes most of the computing time. With the relative costs of both medium and large images analyzed, the selection of tasks to be parallelized become getpeaks, newtoncenter, gengrid, convolution, and normalize which account for at least 96% of the total time. All parallelizations are implemented with CUDA [22]. In the following subsections, we provide more detailed technical descriptions

[1] Block sizes of 256, 512 or 1024 threads are typically used for higher performance.

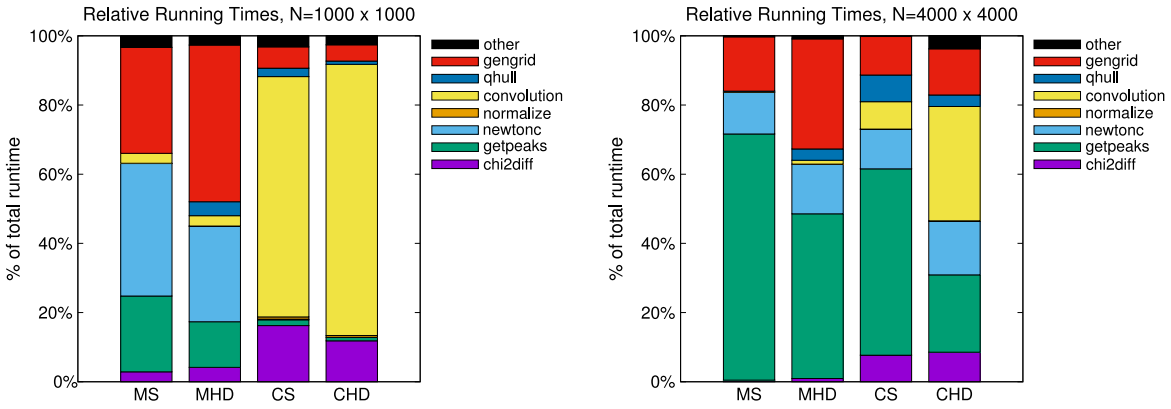


Fig. 5. The relative runtime percentages of the high-density proposed algorithm differ from those of Shattuck and also vary with the image size. MATLAB Shattuck (MS), MATLAB Proposed (MHD), C Shattuck (CS), C Proposed (CHD).

regarding the GPU parallelization approach chosen for each task. We will frequently refer to the symbol, k, which is a complexity parameter that refers to the number of processors that are used to express the asymptotic costs of the parallel tasks.

4.1.1. Pixel level peaks (getpeaks)

This task processes the image and identifies peaks. The parallel GPU approach uses a *per-pixel mapping* approach, where threads handle different small regions of the image to search for local minima. The cost of this parallel computation is $O(N/k)$ operations.

4.1.2. Gauss–Newton iterations (newtonCenter)

Here we compute (Δ_x, Δ_y) on each peak location, as defined in Algorithm 1. This task has no data race condition as the accesses on the neighbor pixels are read-only and the write operations are performed only on the peak locations that are independently parallelized with GPU threads. The parallel GPU strategy uses a *per-particle mapping* approach that costs $O(PN/k)$. It is important to mention that a synchronization barrier is required in order to prevent memory accesses on outdated information, *i.e.*, when the neighbor accesses include another derivative location.

4.1.3. W matrix (gengrid)

The W *matrix* task finds, for each pixel, the closest particle to that pixel as well as its distance from it. The GPU parallelization uses a *per-particle mapping* approach, with a local neighborhood of pixels for each particle to search through. Although this mapping approach may produce race conditions when particles are too close (*i.e.*, overlapped neighborhoods), we found by experimentation that on average four repetitions make W rapidly converge to the result given by the sequential implementation, with a standard deviation of 0.5 for the mean of tracked particles. Because of this favorable behavior, it was not necessary to opt for *per-pixel mapping*, where the amount of parallel resources increase in the order of the image size. This chosen parallel GPU approach has an asymptotic cost of $O(P/k)$ operations.

4.1.4. Convolutions

The GPU parallelization of the convolution uses the *cuFFT* library [23] which is a highly optimized code for GPU-based FFT computations. The strategy used by cuFFT is a *per-pixel mapping* approach, where information from image space is transformed into information in the frequency spectrum. The cost of a parallel FFT computation in theory is $O(\frac{n}{k}\log 2(n))$.

4.1.5. Normalize

The normalize task does not have a significant cost when compared to the other tasks. Nevertheless, we still included it in GPU parallelization since it is a *pleasingly parallel* computation pattern [11] that does not require significant implementation time. This task normalizes the color magnitudes of each pixel within a fixed range relative to the maximum and minimum color values found. The GPU-based implementation of this task uses a *per-pixel mapping* approach to modify each pixel value according to the new range given by the minimum and maximum color values. The cost of the parallel implementation is $O(N/k)$.

4.1.6. Difference matrix J (chi2diff)

This routine computes a matrix J of differences defined as follows:

$$J_{x,y} = \frac{1 - \tanh\left([W_{x,y} - D/2]/w\right) - 2I_{x,y}}{2}, \tag{11}$$

where $J_{x,y}$ is difference value at pixel (x, y) and D, w are the diameter and sharpness of the particles, respectively. The parallelization of this computation uses per pixel-mapping in order to compute $J_{x,y}$ simultaneously on all (x, y) locations.

5. Results

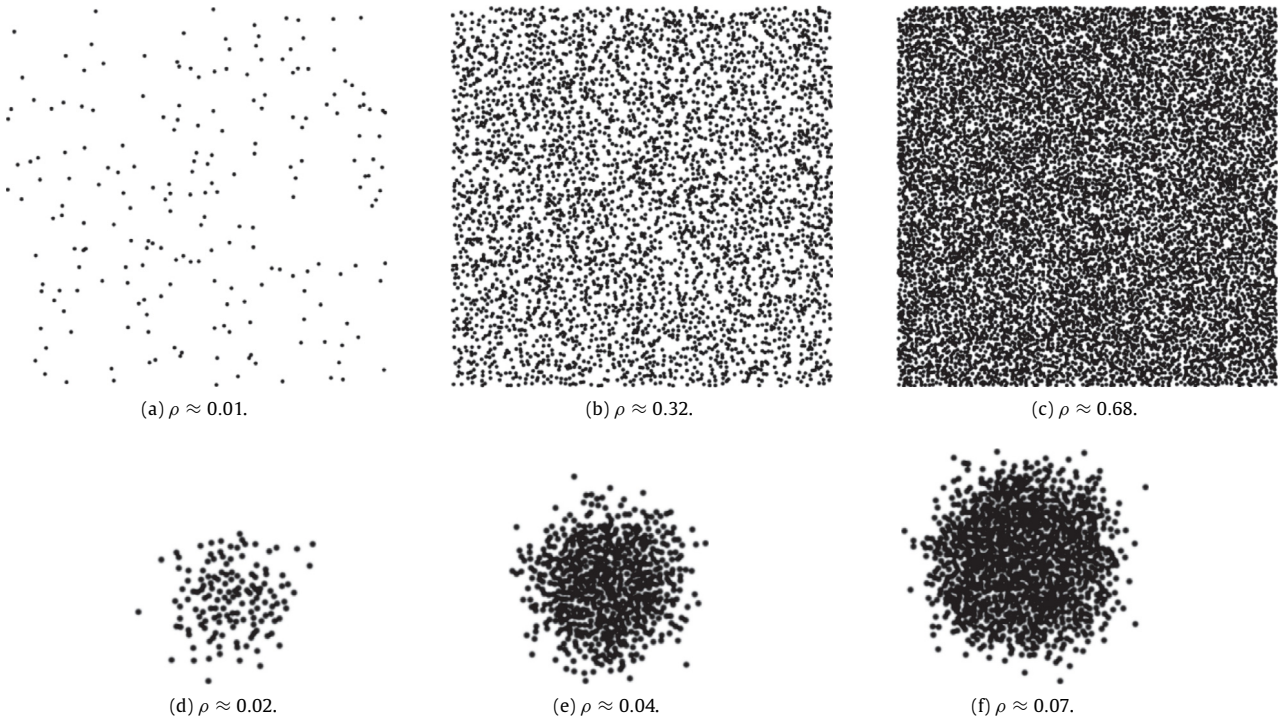
Both the CPU and GPU implementations of the new tracking algorithm, chi2HD and chi2HDCuda, respectively, as well as the C-language version of Shattuck's sequential tracking algorithm (named chi2), are tested in terms of quality and performance. The quality of a tracking process is

$$Q = N_t/N, \tag{12}$$

where N_t is the number of tracked particles and N is the original number of particles in the image. The performance of a single particle tracking execution process is defined as the sum of the performances of the main sub-routines present in all three implementations, *i.e.*,

$$T = T_{gengrid} + T_{convolution} + T_{normalize} + T_{newtoncenter} + T_{getpeaks} + T_{chi2diff} + T_{other}, \tag{13}$$

where T_{other} corresponds to the time of the rest of the computations, which have a lesser impact. For the GPU implementation, all routines except *other* run on the GPU. The input tests consist of synthetic images of $L \times L$ pixels where particle coordinates follow a Beta distribution $B(\alpha, \beta)$ with parameters $D = 10$, $\omega = 1.4273$. Two categories of tests were generated; (1) the uniform distribution, *i.e.* $B(\alpha = 1, \beta = 1)$, and (2) the cluster distribution,



(a) $\rho \approx 0.01$. (b) $\rho \approx 0.32$. (c) $\rho \approx 0.68$.

(d) $\rho \approx 0.02$. (e) $\rho \approx 0.04$. (f) $\rho \approx 0.07$.

Fig. 6. Synthetic image generation for benchmarking. The uniform (upper row) and cluster distributions (lower row) at different densities for $L = 1000$. The cluster is embedded in the same frame of $L \times L$ pixels as in the uniform case.

Table 1
Workstation used for performance tests.

OS	Arch Linux 64-bit
RAM	128 GB
CPU	Intel i7-6950X @ 3.0 GHz
GPU	NVidia GTX Titan X (Pascal, GP102), 3584 CUDA cores, 12 GB

i.e. $B(\alpha = 50, \beta = 50)$, see Fig. 6. For each category several instances of synthetic images were generated with different N, L.

The packing density, which is referred here just as density, is

$$\rho = \frac{\pi N (D/2)^2}{L^2} \tag{14}$$

and it is used as the independent variable in order to explore the behavior of quality and performance when increasing the number of particles. The L parameter is also used as an independent variable in order to plot the behavior quality stability as well as performance scaling. The computer used in all tests is specified in Table 1.

5.1. Uniform distribution

Fig. 7 shows the tracking quality and performance of the sub-routines for particles that follow a uniform distribution. On the top-left plot, as density increases (ρ), the % of detected particles (Q) drops to 93% for chi2 while chi2HD has a slightly drop of 1%. On the top-right, the tracking quality is preserved for different L, except for $L < 500$ where tracking quality is less than 93%. From the bottom-left plot, the speedup provided by the GPU implementation is significantly faster than the rest of the implementations, achieving close to an order of magnitude with respect to the CPU version of chi2HD. It is important to note that the sequential version chi2HD is even slower than the chi2 algorithm, making the CUDA implementation even more useful. This performance behavior is preserved for the density range. Another aspect to note is that at lower densities the cost of the algorithms is dominated by the convolution, and for higher densities the other sub-routines take a more relevant role affecting performance. From the bottom-right plot, it is possible to see that L does affect the speedup obtained by the GPU implementation, making chi2HD the fastest for high resolution images ($L > 1000$).

5.2. Cluster distribution

Fig. 8 shows the tracking quality and performance of the sub-routines for the cluster distribution test. On the top-left plot, one can note that the tracking quality of the chi2 algorithm drops at a faster rate than in the uniform distribution test. At a density of 8%, the chi2 algorithm tracks only 80% of the particles, thus fairing far poor than our algorithm, which still manages to track more than 95% of the particles (in both CPU and GPU). The concave shape of the curve can be explained by the fact that once a cluster reaches maximal density no more particles fit in the core, thus the only particles that end up being added to an image are the peripheral ones which have more distance among them. When varying L (top-right plot), the tracking quality becomes stable once $L > 1000$. In terms of performance it is interesting to note that in the bottom-left plot most of the computing time is dominated by the convolution. This is because, although a cluster is locally dense, the number of particles does not contribute significantly to the cost. In terms of scaling with L (bottom-right), the chi2HDCuda implementation manages to be the fastest of all three for high resolution images ($L > 1000$).

Although the CPU performance results of chi2 and chi2HD come from sequential implementations, they are still useful measures for comparison as they allow one to estimate an upper bound to the eventual performance on a fully utilized multi-core chip. A parallel multi-core CPU implementation would produce higher performance indeed, however it is unlikely to scale perfectly linearly with the number of cores, *i.e.* to have 100% parallel multi-core efficiency. In practice, bottlenecks from memory bandwidth and caching mechanisms begin to manifest as more processors are

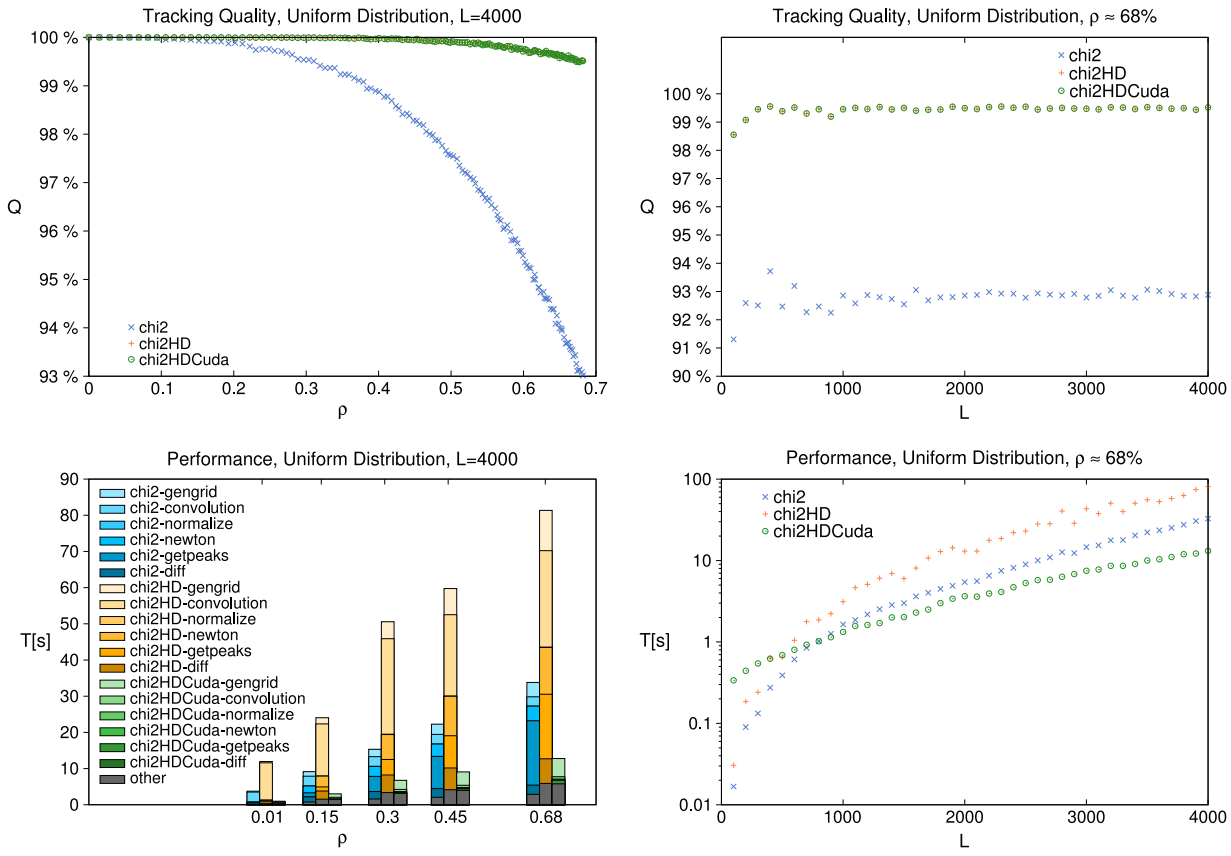


Fig. 7. Quality (top) and performance (bottom) results for the uniform distribution.

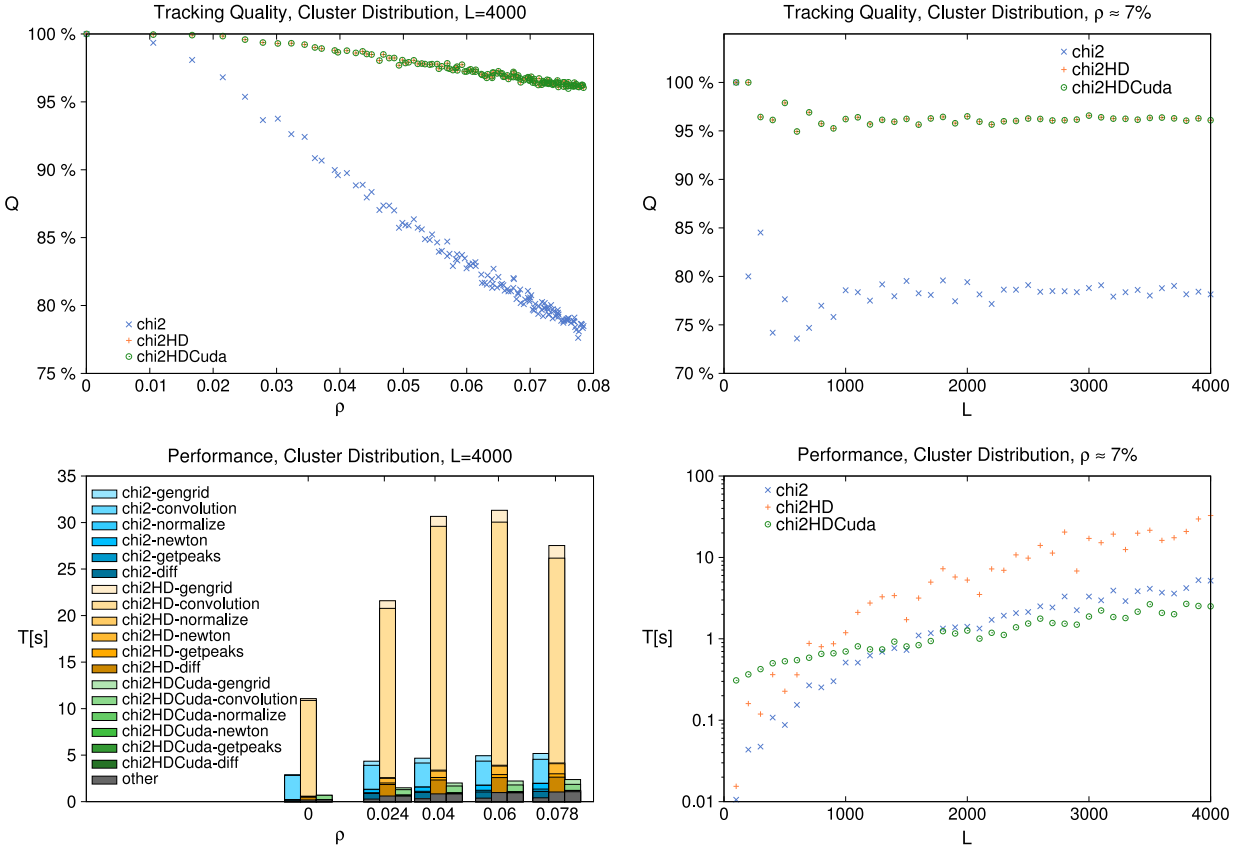


Fig. 8. Quality (top) and performance (bottom) results for the cluster distribution.

used. A relatively optimized multi-core implementation on a 4-core CPU could produce a parallel efficiency in the range 70%–80%, which would put the CPU result much closer to the GPU ones. Such approach can be an interesting alternative for workstations that lack a GPU. For the results obtained here, the GPU implementation provides a speedup of up to 10× over the CPU implementation.

6. Conclusions

In this work we have presented a new algorithm for high-speed particle tracking for dense media. The algorithm adds missed particles to an initial local minima peak detection routine to ultimately refine peak detection at the sub-pixel level. False positives in this peak detection are reliably and efficiently cleaned using the particle's Voronoi area and the image intensity. We have shown that the proposed approach is precise, achieving at least 95% particle detection in dense scenarios, which is 15% better than the most widely-used reference algorithm. We have illustrated how this difference in detection fidelity can lead to errors of up to 40% in physical quantities such as the static structure factor. In terms of speed, the proposed GPU-based parallelization speeds up computations by up to 10× in comparison to its sequential CPU version, and this speedup is most significant when the images studied are high-resolution images ($L > 1000$). Considering that the *C* version is already 4× faster than the reference MATLAB particle tracking library [15], one sees that the proposed algorithm is up to 40× faster than the MATLAB standard bearer for high-resolution, high-density data. The results obtained in this work illustrate that the proposed algorithm is an excellent tool for performing particle tracking in dense media recorded on high resolution images. The authors make the source code available in [24].

Acknowledgments

We acknowledge the support of Fondecyt Grants #1150393, #3160182, #11161033, #1151029. S.R.W and N.M. acknowledge T. Witten and the Chicago-Chile Materials Collaboration funded through the National Science Foundation's MRSEC Program and its office of international programs under Awards DMR-0303072 and DMR-0807012. M.C. acknowledges support to FONDEQUIP EQM140119; Ring Initiative ACT1402, the Chilean Millennium Science Initiative (grant P09-015-F), CORFO (16CTTS-66390), and DAAD (57220037 & 57168868).

References

[1] E. Meijering, O. Dzyubachyk, I. Smal, W.A. van Cappellen, Semin. Cell Dev. Biol. 20 (2009) 894–902.
[2] K.M. Taute, S. Gude, S.J. Tans, T.S. Shimizu, Nat. Comm. 6 (2015) 1–9.
[3] T.A. Caswell, Z. Zhang, M.L. Gardel, S.R. Nagel, Phys. Rev. E 87 (2013) 012303.
[4] S.R. Waitukaitis, H.M. Jaeger, Rev. Sci. Instrum. 84 (2013) 025104.
[5] G. Castillo, N. Mujica, R. Soto, Phys. Rev. Lett. 109 (2012) 095701.
[6] G. Castillo, N. Mujica, R. Soto, Phys. Rev. E 91 (2015) 012141.
[7] V. Castañeda, M. Cerda, F. Santibañez, J. Jara, E. Pulgar, K. Palma, C.G. Lemus, M. Osorio-Reich, M.L. Concha, S. Härtel, Curr. Mol. Med. 14 (2) (2014) 291–307.
[8] S.P. Pudasaini, S.-S. Hsiau, Y. Wang, K. Hutter, Phys. Fluids 17 (2005) 093301.
[9] X. Cheng, L. Gordillo, W.W. Zhang, H.M. Jaeger, S.R. Nagel, Phys. Rev. E 89 (2014) 042201.
[10] W. Losert, D.G.W. Cooper, J. Delour, Chaos 9 (1999) 682.
[11] C.A. Navarro, N. Hitschfeld-Kahler, L. Mateu, Commun. Comput. Phys. 15 (2014) 285–329.
[12] C.A. Navarro, W. Huang, Y. Deng, Comput. Phys. Comm. 205 (2016) 48–60.
[13] J.S. Olafsen, J.S. Urbach, Phys. Rev. Lett. 81 (1998) 4369–4372.
[14] N. Mujica, R. Soto, in: J. Klapp, L.D.G. Sigalotti, A. Medina, A. López, G. Ruiz-Chavarría (Eds.), In Recent Advances in Fluid Dynamics with Environmental Applications, Springer, 2016.
[15] S.V. Franklin, M.D. Shattuck, Handbook of Granular Materials, CRC Press, 2015.
[16] H. Lee, S. Lee, IEEE J. Biomed. Health Inform. 99 (2017).
[17] M. Berg, M. Kreveld, M. Overmars, O.C. Schwarzkopf, Computational Geometry: Algorithms and Applications, Springer Berlin Heidelberg, 2000.
[18] J. Nocedal, S.J. Wright, Numerical Optimization, Springer Berlin Heidelberg, 1999.
[19] J.-P. Hansen, I.R. McDonald, Theory of Simple Liquids, Academic Press, 1990.
[20] J. Silva, Memoria Para Optar Al Título de Ingeniero, Departamento de Computación Universidad de Chile, 2015.
[21] Nvidia Corporation, CUDA C Programming Guide, 2017.
[22] J. Nickolls, I. Buck, M. Garland, K. Skadron, Queue 6 (2008) 40–53.
[23] Nvidia Corporation, CUDA CUFFT Library, 2007.
[24] Proposed algorithm source code and benchmarking scripts (branch benchmarking), 2017. https://github.com/ParticleTracking2, (accessed 13.09.17).