An Adaptive Distributed Resampling Algorithm with Non-Proportional Allocation

Ömer Demirel*, Ihor Smal†, Wiro Niessen†, Erik Meijering† and Ivo F. Sbalzarini*
*MOSAIC Group, Center of Systems Biology Dresden (CSBD),
Max Planck Institute of Molecular Cell Biology and Genetics,
Pfotenhauerstr. 108, 01307 Dresden, Germany.
Email: {demirel,ivos}@mpi-cbg.de
†Biomedical Imaging Group Rotterdam, Erasmus MC,
University Medical Center Rotterdam, Rotterdam, The Netherlands.
Email: {i.smal,w.niessen}@erasmusmc.nl, meijering@imagescience.org

***Abstract*—The distributed resampling algorithm with proportional allocation (RNA) [1] is key to implementing particle filtering applications on parallel computer systems. We extend the original work by Bolić *et al*. by introducing an adaptive RNA (ARNA) algorithm, improving RNA by dynamically adjusting the particle-exchange ratio and randomizing the process ring topology. This improves the runtime performance of ARNA by about 9% over RNA with 10% particle exchange. ARNA also significantly improves the speed at which information is shared between processing elements, leading to about 20-fold faster convergence. The ARNA algorithm requires only a few modifications to the original RNA, and is hence easy to implement.**

***Index Terms*—Distributed resampling, particle filter, parallel computing, tracking, image processing.**

I. Introduction

Particle filters (PF) have experienced impressive improvement since their introduction [2]–[4] and are considered the *de facto* standard tool to estimate and track targets with non-linear and/or non-Gaussian dynamics. Due to their computational cost, however, many PF applications are limited to small problems or require long execution times. In order to relax this issue by leveraging parallelism in modern hardware, Bolić *et al*. introduced a distributed algorithm in their seminal work [1]: the distributed resampling algorithm with non-proportional allocation (RNA), which enabled the development of PF applications that efficiently use modern multi-core and multi-processor hardware, such as computer clusters.

Here, we propose a simple, yet effective improvement to RNA based on a *randomized* particle-routing scheme with an *adaptive* particle-exchange ratio. This *adaptive RNA* (ARNA) algorithm improves the runtime performance and the efficiency of RNA. We benchmark these improvements in two situations of object tracking, where (1) the particles on all PEs are initialized at the location of the object to be tracked and with ground-truth velocity, hence testing the (*tracking*) performance, and (2) the particles on only one PE are initialized near the object to be tracked, on all others they are initialized uniformly at random. The latter tests how fast information is shared between PEs once one of them converged to the object (*information sharing*).

II. Particle Filters

A generic PF algorithm consists of two parts: (i) sequential importance sampling (SIS) and (ii) resampling [3]. A popular combined implementation of these two parts is the sequential importance resampling (SIR) algorithm [3].

Recursive Bayesian importance sampling [5] of an unobserved and discrete Markov process $\{\mathbf{x}_k\}_{k=1,\dots,K}$ is based on three components: (i) the measurement vector $\mathbf{Z}^k = \{\mathbf{z}_1, \dots, \mathbf{z}_k\}$, (ii) the dynamics (i.e., state-transition) model, which is given by a probability distribution $p(\mathbf{x}_k|\mathbf{x}_{k-1})$, and (iii) the likelihood (i.e., observation model) $p(\mathbf{z}_k|\mathbf{x}_k)$. Then, the state posterior $p(\mathbf{x}_k|\mathbf{Z}^k)$ at time k is recursively computed as:

$$\underbrace{p(\mathbf{x}_k|\mathbf{Z}_k)}_{\text{posterior}} = \frac{\overbrace{p(\mathbf{z}_k|\mathbf{x}_k)}^{\text{likelihood}}\ \overbrace{p(\mathbf{x}_k|\mathbf{Z}^{k-1})}^{\text{prior}}}{\underbrace{p(\mathbf{z}_k|\mathbf{Z}^{k-1})}_{\text{normalization}}}, \tag{1}$$

where the prior is defined as:

$$p(\mathbf{x}_k|\mathbf{Z}^{k-1}) = \int p(\mathbf{x}_k|\mathbf{x}_{k-1})\, p(\mathbf{x}_{k-1}|\mathbf{Z}^{k-1})\, \mathrm{d}\mathbf{x}_{k-1}. \tag{2}$$

PFs approximate the posterior at each time point k by N weighted samples (i.e., particles) $\{\mathbf{x}_k^i, w_k^i\}_{i=1,\dots,N}$. This approximation is achieved by sampling a set of particles from an importance function (proposal) $\pi(\cdot)$ and updating their weights according to the dynamics and observation models. This process is called sequential importance sampling (SIS) [3]. However, SIS suffers from *weight degeneracy*, whereby small particle weights become successively smaller and do not contribute to the posterior any more. To overcome this problem, a *resampling* step is performed [3] whenever the number of particles with relatively high weights falls below a specified threshold. In order to parallelize the SIR algorithm, one only needs to focus on the *resampling* step, since all other parts of the SIR algorithm are local and can trivially be executed in parallel. The complete SIR algorithm is given in Algorithm 1.

III. Classical RNA

In a distributed-memory computer system with M processing elements (PEs, $m = 1, \dots, M$), the resampling step in

RNA is performed locally by each PE. While the number of particles per PE hence remains constant, ensuring perfect data-balance (i.e., all PEs hold the same amount of data), the weight distribution across PEs can become unbalanced. This requires particle routing (i.e., dynamic load balancing (DLB)) in which every PE moves a constant fraction of its particles to another PE, such that the particle weights become more evenly mixed. A pseudocode for ARNA is shown in Algorithm 2.

Algorithm 1 Sequential Importance Resampling (SIR)

1: (P) Propagate all particles according to the transition prior: $\mathbf{x}_k^{(i)} \sim p(\mathbf{x}|\mathbf{x}_{k-1}^{(i)})$, $i = \{1, \ldots, N\}$
2: (U) Update the weights taking into account the measurements at time k, $\mathbf{z}_k$, as $\tilde{w}_k^{(i)} = p(\mathbf{z}_k|\mathbf{x}_k^{(i)})w_{k-1}^{i)}$
3: Renormalize the weights as $w_k^{(i)} = \tilde{w}_k^{(i)} / \sum_{j=1}^{N} \tilde{w}_k^{(j)}$
4: Compute the estimate $\hat{\mathbf{x}}_k = \sum_{i=1}^{N} w_k^{(i)} \mathbf{x}_k^{(i)}$
5: Compute $N_{\text{eff}} = (\sum_{i=1}^{N} (w_k^{(i)})^2)^{(-1)}$
6: Resample if $N_{\text{eff}} < N_{\text{thresh}}$ using Systematic Resampling

Algorithm 2 Resampling with Non-proportional Allocation (RNA)

1: Exchange N_{ex} of particles with neighboring PEs
2: Renormalize weights as $w_{k-1}^{(m,i)} = w_{k-1}^{(m,i)} / W_{k-1}$
3: Perform (P) and (U) steps of SIR to get $\mathbf{s}_k^m$
4: Compute the estimate $\hat{\mathbf{x}}_k^m$ and the sum of unnormalized weights $W_k^{(m)}$
5: Resample $\mathbf{s}_k^m$ using the locally normalized weights $\tilde{w}_k^{(m,i)} = w_k^{(m,i)} / W_k^{(m)}$
6: Set the i-th weight to $w_k^{(m,i)} = W_k^{(m)}$
7: Send $\hat{\mathbf{x}}_k^m$ and $W_k^{(m)}$ to the master PE
8: The master PE computes $\hat{\mathbf{x}}_k$ and W_k and broadcasts the result to all PEs

A. Particle Routing via Local Exchange

The *local exchange* method uses a fixed number of $N_p = N/M$ particles on each PE and also fixes the number N_{ex} of particles to be exchanged. In this RNA configuration, the PEs are arranged in a ring topology and each PE sends N_{ex} particles to its (counter-)clockwise neighbor in the ring. Since each PE only communicates with its neighbor, several rounds of communications are required until the weights are approximately evenly distributed and the accuracy of the particle representation of the posterior $p(\mathbf{x}_k|\mathbf{Z}^k)$ is recovered.

B. Deterministic Particle Routing Schedule

The *local exchange* method with a particle-exchange ratio of 10% or 50% is a popular choice when implementing RNA [1], [6], [7]. This avoids the need for application-dependent DLB schedules. Fixing N_{ex} in the local exchange method, the DLB scheme is easier and faster to design and implement. However, since this DLB scheme is static, it does not adapt to the dynamics of the application, where different load imbalance situations may arise.

C. Ring topology

In the original RNA, the PEs are arranged in a ring and only communicate with their adjacent neighbors. PE P_m randomly selects N_{ex} (out of its N_p) particles and sends them to PE P_{m+1}. Concurrently, it receives N_{ex} new particles from P_{m-1}. While the ring topology leads to a simple communication schedule, it also has the lowest *conductance* (i.e., speed of information spreading) from a graph-theory point of view. Thus, the information of "good" particle weights is shared only slowly across PEs. Furthermore, the performance of this DLB scheme in the ring topology degenerates as the number of PEs increases [8].

IV. Adaptive RNA

We propose the ARNA algorithm, which improves over the classical RNA by using dynamically adaptive particle-exchange ratios and randomized ring topologies.

A. Adaptive Particle-Exchange Ratio

The traditional RNA uses a fixed particle exchange ratio that need to be set by the user. We relax this constraint by making N_{ex}/N_p dynamically adaptive, allowing it to vary between $0 \ldots 50\%$ as:

$$N_{\text{ex}} = N_p \left[0.5 - \frac{0.5(PE_{\text{eff}} - 1)}{M - 1} \right] . \tag{3}$$

Hence, N_{ex} is negatively correlated with the tracking efficiency PE_{eff}, which is defined as:

$$PE_{\text{eff}} = \frac{\left(\sum_{m=1}^{M} \sum_{i=1}^{N} w_k^{(m,i)} \right)^2}{\sum_{m=1}^{M} \sum_{i=1}^{N} (w_k^{(m,i)})^2} , \tag{4}$$

where $w_k^{(m,i)}$ is the weight of i-th particle on m-th PE. PE_{eff} measures the percentage of PEs that have already located the object and track it successfully.

The adaptive exchange rate in ARNA frees the user of fixing this parameter, and helps reduce communication-network congestion and thus increases the parallel performance. The advantage of this adaptive approach becomes more pronounced for high tracking accuracies, i.e., in the *tracking* case.

B. Randomized Ring Topology

In a complete graph, information can be shared between any two PEs in single communication step. However, such all-to-all communication limits the parallel scalability of the algorithm. We introduce an improved (in the sense of faster mixing) DLB scheme for ARNA that has the same communication cost as the original RNA, i.e., the same number of send and receive operations per PE.

We exploit the power of *randomization* methods, which are well-established for approximately solving NP-complete problems, such as the present one. As a simple change to RNA, we randomize the vertex labeling in the ring topology. This is equivalent to having a complete graph and selecting different, random Hamiltonian paths (i.e., paths that visit each node exactly once) in this graph. Projecting the complete

graph onto a ring topology via a Hamiltonian path, each PE only communicates with two other PEs, as in the classical RNA. We use Fisher-Yates shuffling [9] to efficiently compute randomized ring topologies. One could also apply other regular graphs with low maximum degree, but such topologies would require knowledge about the hardware network connecting the PEs in the actual machine. With no prior knowledge about process-to-PE assignment and hardware network topology, the present random ring labeling provides a simple tool to increase the efficiency of information spread in ARNA.

C. Algorithm

ARNA only requires a few minor modifications to RNA in steps 1 and 2. A pseudocode for ARNA is given in Algorithm 3.

Algorithm 3 Adaptive RNA (ARNA)

1: Randomize the PE topology using Fisher-Yates shuffle [9]
2: Update the particle-exchange ratio N_{ex}/N_p according to Eq. 3. This requires a global communication in order to compute PE_{eff}.
3: Exchange N_{ex} of particles with neighboring PEs
4: Renormalize weights as $w_{k-1}^{(m,i)} = w_{k-1}^{(m,i)}/W_{k-1}$
5: Perform (P) and (U) steps of SIR to get $\mathbf{s}_k^m$
6: Compute the estimate $\hat{\mathbf{x}}_k^m$, and the sum of unnormalized weights $W_k^{(m)}$
7: Resample $\mathbf{s}_k^m$ using the locally normalized weights $\tilde{w}_k^{(m,i)} = w_k^{(m,i)}/W_k^{(m)}$
8: Set the i-th weight to $w_k^{(m,i)} = W_k^{(m)}$
9: Send $\hat{\mathbf{x}}_k^m$ and $W_k^{(m)}$ to the master PE
10: The master PE computes $\hat{\mathbf{x}}_k$ and W_k and broadcasts the result to all PEs

V. Benchmarks

We benchmark the improvements of the proposed ARNA over RNA using an application from object tracking in fluorescence microscopy imaging [10], [11]. The goal here is to track the motion of small structures that are labeled with fluorescent dyes. From this, one can then characterize the dynamics of those objects and quantify, e.g., their velocity, spatial distribution [12], motion correlations, etc.

We use the same previous sequential implementation of SIR [13], [14] inside both RNA and ARNA. The dynamics model assumes nearly constant velocity, and the appearance model approximates each object by Gaussian intensity profile in the final microscopy image. These are standard models that adequately describe biological fluorescence microscopy [13], [14]. The state vector in this case is $\mathbf{x} = (\hat{x}, \hat{y}, v_x, v_y, I_0)^T$, where $\hat{x}$ and $\hat{y}$ are the estimated x- and y-positions of the object, (v_x, v_y) its velocity vector, and I_0 its estimated fluorescence intensity. An example image of a object tracking in fluorescence microscopy imaging is shown in Fig. 1.

For the performance evaluation, 10 different, synthetically generated image sequences are used, each containing 50 frames of size 512×512 pixels. The tracking performance is evaluated for two different modes: *tracking* and *information sharing*. In the first mode, all PEs contain particles that are initialized at the true object state. In the second scenario, the particles are uniformly randomly initialized in state space on all but one PE. On one PE, the particles are initialized at the true state. This models the situation that one PE has discovered and converged on the object and needs to efficiently share this information with the other PEs. After that, the two distributed SIR implementations (one with ARNA and one with RNA) are used to locate the object in the subsequent frames and continue with accurate tracking and position estimation.

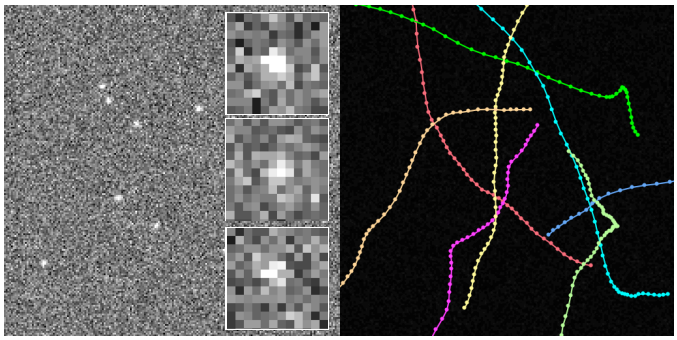


Fig. 1. Examples of synthetic images used in the benchmarks. Left: One frame of a typical 2D image sequence with signal-to-noise ratio 2, containing the small, bright objects of interest. Zoomed insets show noisy object appearance, modeled using a 2D Gaussian intensity profile corrupted with Poisson noise. Right: Typical object trajectories, generated according to the nearly-constant-velocity model.

We compare ARNA against RNA with 0%, 10%, and 50% particle-exchange ratios. The memory footprint of a single particle is 52 kB (i.e., six doubles and one integer. The six doubles are the five components of the state vector and the particle weight. The integer is the process ID of where that particle belongs). All tests of *tracking* are repeated 50 times for statistical significance. For *information sharing*, we benchmark the recovery curve of PE_{eff} on five different synthetic image sequences, each test repeated 10 times. All experiments are run on the MadMax computer cluster of MPI-CBG, Dresden, which is equipped with 128 GB DDR3 800-MHz memory per node and two Intel® Xeon® E5-2640 six-core processors per node with a clock speed of 2.5 GHz. Both ARNA and RNA are implemented in Java (v. 1.7.0_13) in the Parallel Particle Filtering (PPF) library [15]. We use OpenMPI's Java bindings (v. 1.9a1r28750) for inter-process communication [16].

A. Tracking performance

We initialize 19.2 million particles at the location of the targeted object and thus we ensure high-accuracy tracking. In such a scenario, if correct dynamics and observation models are used, inter-process communication is virtually unnecessary since all PEs independently track the object. The classical RNA model, however, is oblivious to the mode of the application, as the process topology and the particle-exchange ratio are fixed. In ARNA, the particle exchange ratio N_{ex}/N_p is

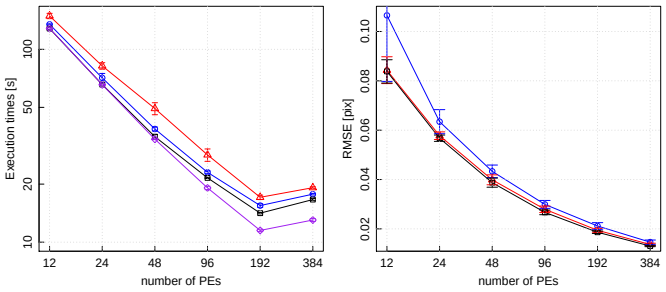


Fig. 2. Left: Execution times of RNA with 50% exchange (red), 10% exchange (blue), and 0% exchange (purple) compared with the timings for ARNA (black). A fixed total number of 19.2 million particles is distributed over an increasing number of PEs (strong scaling). ARNA is faster than RNA with 10% and 50% exchange. RNA with 0% exchange (i.e., embarrassingly parallel RNA) defines the lower bound for this test case, where no communication is necessary. Beyond 192 PEs, the number of particles per processor is too small to amortize the constant communication overhead. Right: RMSE tracking accuracy in pixels (40 particles per PE, initialized at the target.) RNA with 50% particle exchange (red) and ARNA (black) show comparable tracking accuracy, whereas RNA with 10% exchange (blue) yields lower accuracy. As the total number of particles increases, the tracking becomes more accurate in all cases.

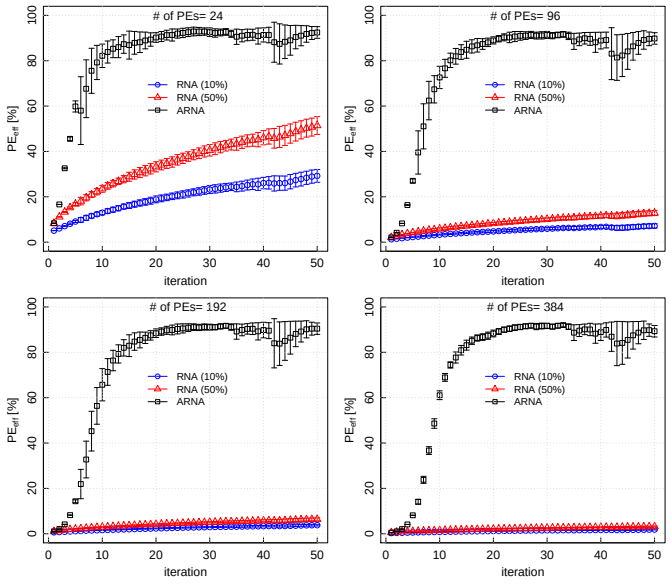


Fig. 3. Percentage of PEs engaged in successful tracking of the target (PE_{eff}) as a function of iteration number during the information sharing phase: ARNA (black), RNA with 10% particle exchange (blue), and RNA with 50% particle exchange (red) on 24, 96, 192, and 384 PEs. The randomized ring topology of ARNA leads to a faster spread of information and hence a higher tracking efficiency.

negatively correlated with the tracking efficiency. PEs do not exchange any particles if PE_{eff} is above 99%. The runtime results of the benchmarks are shown in Fig. 2. The tracking accuracy of ARNA is comparable to that of RNA with 50% particle exchange. When exchanging only 10% of the particles in RNA, the accuracy drops. Visually, however, all resulting trajectories are indistinguishable, as the Root Mean Square Error (RMSE) of the tracking is below 0.1 pixel in all cases.

B. Information Sharing Performance

In applications with no prior information about the initial state of the system, it is common practice to initialize the particles uniformly at random throughout the state space. This helps explore the state space and first detect the object to be tracked. At some point, one of the PEs will (stochastically) detect the object to be tracked and the particles on the PE converge around the object. Until this point, all PEs uniformly sample the state space and communication between them does not help. Once one PE has found the target, however, this information should be disseminated among all PEs as quickly as possible, in order to allow the other PEs to contribute to the tracking accuracy. In a parallel PF application we want all PEs to contribute to the result (i.e., not waste computational resources). PE_{eff} should hence reach 100% as quickly as possible after initialization.

In ARNA, the randomized ring topology helps share the detection information more rapidly. Figure 3 shows how PE_{eff} evolves with algorithm iterations for the different parallel algorithms, counting iterations from the time point where one of the PEs has found the object.

VI. Conclusions

We presented ARNA, an adaptive randomized version of the classical RNA [1] algorithm for parallel particle filtering, which is easy to implement and requires only few minor changes with respect to RNA. ARNA uses a dynamically adapted, tracking efficiency-dependent particle-exchange ratio. In cases where only little communication is required, ARNA is about 9% faster than RNA with a 10% exchange ratio. Moreover, ARNA enhances information sharing via randomizing the ring topology of the PEs. This leads to a faster increase in the percentage of PEs that have successfully located the target once at least one PE has converged. ARNA hence improves the tracking accuracy and effectiveness by having more PEs contribute to the result earlier. Benchmarks show that in a large network of 384 PEs, the difference between RNA and ARNA is even more pronounced: both RNA versions score below 4% PE_{eff}, whereas ARNA reaches 60% after 10 iterations, converging to over 80% after 20 iterations.

Future work could further improve ARNA by including prior knowledge about how the processes are assigned to PEs and how the latter are connected in the machine by the hardware network. This way, neighboring PEs in the ring can reside on the same cluster node, hence further reducing communication overhead. Using hardware-topology information would also enable the use of other regular graphs with low maximum degree as communication topologies, which may better reflect a specific hardware than a generic ring topology.

Acknowledgments

The authors would like to thank the MOSAIC Group and MadMax cluster team (MPI-CBG). Ömer Demirel was funded by grant #200021–132064 from the Swiss National Science Foundation (SNSF), awarded to I.F.S. Ihor Smal was funded by a VENI grant (#639.021.128) from the Netherlands Organization for Scientific Research (NWO).

REFERENCES

[1] Miodrag Bolic, Petar M Djuric, and Sangjin Hong. Resampling algorithms and architectures for distributed particle filters. *Signal Processing, IEEE Transactions on*, 53(7):2442–2450, 2005.

[2] Arnaud Doucet, Simon Godsill, and Christophe Andrieu. On sequential monte carlo sampling methods for bayesian filtering. *Statistics and computing*, 10(3):197–208, 2000.

[3] Arnaud Doucet, Nando De Freitas, Neil Gordon, et al. *Sequential Monte Carlo methods in practice*, volume 1. Springer New York, 2001.

[4] Petar M Djuric, Jayesh H Kotecha, Jianqui Zhang, Yufei Huang, Tadesse Ghirmai, Mónica F Bugallo, and Joaquin Miguez. Particle filtering. *Signal Processing Magazine, IEEE*, 20(5):19–38, 2003.

[5] John Geweke. Bayesian inference in econometric models using monte carlo integration. *Econometrica: Journal of the Econometric Society*, pages 1317–1339, 1989.

[6] Joaquín Míguez. Analysis of parallelizable resampling algorithms for particle filtering. *Signal Processing*, 87(12):3155–3174, 2007.

[7] Sven Zenker. Parallel particle filters for online identification of mechanistic mathematical models of physiology from monitoring data: performance and real-time scalability in simulation scenarios. *Journal of clinical monitoring and computing*, 24(4):319–333, 2010.

[8] Bhaskar Ghosh and S Muthukrishnan. Dynamic load balancing by random matchings. *Journal of Computer and System Sciences*, 53(3):357–370, 1996.

[9] Ronald Aylmer Fisher, Frank Yates, et al. Statistical tables for biological, agricultural and medical research. *Statistical tables for biological, agricultural and medical research.*, (Ed. 3.), 1949.

[10] A. Akhmanova and C. C. Hoogenraad. Microtubule plus-end-tracking proteins: Mechanisms and functions. *Current Opinion in Cell Biology*, 17(1):47–54, 2005.

[11] Y. Komarova, C. O. de Groot, I. Grigoriev, S. Montenegro Gouveia, E. L. Munteanu, J. M. Schober, S. Honnappa, R. M. Buey, C. C. Hoogenraad, M. Dogterom, G. G. Borisy, M. O. Steinmetz, and A. Akhmanova. Mammalian end binding proteins control persistent microtubule growth. 184(5):691–706, 2009.

[12] Jo Helmuth, Grégory Paul, and Ivo Sbalzarini. Beyond co-localization: inferring spatial interactions between sub-cellular structures from microscopy images. *BMC bioinformatics*, 11(1):372, 2010.

[13] I. Smal, K. Draegestein, N. Galjart, W. Niessen, and E. Meijering. Particle filtering for multiple object tracking in dynamic fluorescence microscopy images: Application to microtubule growth analysis. 27(6):789–804, 2008.

[14] I. Smal, E. Meijering, K. Draegestein, N. Galjart, I. Grigoriev, A. Akhmanova, M. E. van Royen, A. B. Houtsmuller, and W. Niessen. Multiple object tracking in molecular bioimaging by Rao-Blackwellized marginal particle filtering. 12(6):764–777, 2008.

[15] Ömer Demirel, Ihor Smal, Wiro Niessen, Erik Meijering, and Ivo F. Sbalzarini. Ppf - a parallel particle filtering library. *arXiv preprint arXiv:1310.5045*, 2013.

[16] Open mpi: Trunk nightly snapshot tarballs, 09 2013.