Diss. ETH No. 22042

Dynamic load balancing in parallel particle methods

A dissertation submitted to
ETH Zurich

for the degree of
Doctor of Sciences

presented by
Ömer Demirel
M.Sc. Computational Science and Engineering
Technische Universität München, Germany
born on March 20, 1983
citizen of Turkey

accepted on the recommendation of
Prof. Dr. Peter Widmayer
Dr. Ivo F. Sbalzarini
Prof. Dr. Erik Meijering

2014

Declaration of Authorship

This thesis is a presentation of my own original research work. Wherever contributions of others are involved, every effort is made to indicate this clearly, with due reference to the literature. This work has been started at ETH Zurich in Switzerland and later continued at the Center for Systems Biology Dresden, Max Planck Institute of Molecular Cell Biology and Genetics (MPI-CBG) in Dresden, Germany under the guidance of Dr. Ivo F. Sbalzarini. I hereby declare that this thesis has not been submitted before to any institution for assessment purposes.

Most of the work presented in chapters 3, 4, and 5 has been published [1–3]. Some of the work presented in chapters 1 and 5 has been submitted for publication [4].

Abstract

Contemporary scientific simulations require vast computing power to solve complex problems in natural and life sciences. High-performance computing (HPC) enables fast execution of scientific simulation codes by parallelizing the computational workload across processing elements (PEs). This is usually done by decomposing the computational domain into smaller subdomains and assigning each subdomain to a PE. The subdomains encapsulate computational elements such as computational meshes or particles. The workload of each PE is defined by the number of operations done using these computational elements.

In many scientific simulations, the initial load balance is not preserved during the course of simulation. Outside factors (e.g., congestion in the cluster network) as well as simulation dynamics (e.g., movement of particles in the computational domain) may alter the loads of PEs and may cause load imbalance. In such situations, overloaded PEs require more time to finish their tasks while underloaded PEs remain idle and wait for the overloaded PEs. This protracts simulation runtime and results in redundant energy consumption and wasting resources, which we would like to avoid to the extend possible. To handle these load imbalance situations efficiently, *dynamic load balancing* (DLB) techniques have been developed. In DLB, one is interested in making fast corrections to the load imbalance. As opposed to static load balancing, DLB tries to avoid a brand new subdomain-to-PE assignment.

In this thesis, we investigate DLB solutions for parallel particle methods. First, we look at domain-decomposition-based [5] parallel simulations. We analyze different models used in the literature to model computational loads (i.e., subdomain costs). We argue that considering the loads *indivisible* and *real valued* is the best way to model loads since subdomain costs are closely related to wall-clock times of PEs, which are measured and stored as real numbers. Following that, we discuss how a DLB protocol could be designed such that it offers a scalable and fast solution to redistributing loads across PEs. In this context, we analyze distributed DLB protocols based on Balancing Circuit Model (BCM) [6] and develop theoretical bounds on the expected load imbalance when *indivisible, real valued* loads need to be balanced. Further, we explain several distributed DLB protocols, which can be employed to solve the load imbalance problem efficiently. By combining best features of these DLB protocols, we come up with a `HybridBalancer` which can reduce an initial load imbalance by 3x in just a few DLB iterations in a network of more than a million PEs.

Second, we take a particular numerical method and investigate possible DLB solutions that it can benefit from. Particle filters (PFs) [7–10] are sequential Monte Carlo methods (SMC) [11–13] developed to solve estimation problems optimally in nonlinear and non-Gaussian state-space models using particles as main data structure. Here, we focus on tracking of sub-cellular objects in biological images and discuss several parallel PF algorithms that would allow us to use HPC systems to tackle these problems faster.

Parallel PF algorithms suffer from two types of load imbalance, namely, *particle imbalance* and *particle weight imbalance*. In this work, we improve existing parallel PF algorithms by discussing advanced DLB strategies. These DLB methods help us increase tracking efficiency by more than 20x and runtime performance by 9%. Later, we introduce a new parallel PF called Box Exchange Method (BEM), which reduces communication overhead in the DLB step and accelerates parallel execution of a PF. BEM outperforms the fastest parallel PF algorithm by more than 2x. Moreover, a detailed look at the classical PF algorithm allows us to develop an approximation algorithm (pcSIR), which provides on par tracking accuracy but offers remarkable performance improvements of up to several orders of magnitude.

Kurzfassung

Moderne wissenschaftliche Simulationen erfordern große Rechenleistung, um komplexe Probleme in Natur- und Lebenswissenschaften zu lösen. High-Performance Computing (HPC) ermöglicht eine schnelle Ausführung von wissenschaftlichen Simulationen durch Parallelisierung der Rechenlast auf Rechenelementen (REs). Dieses kann in der Regel durch Teilen des Rechengebietes in kleinere Subdomains und Zuordnung jeder Subdomain auf eine RE erreicht werden. Die Subdomains beinhalten Rechenelemente wie Gitter oder Partikel. Die Rechenlast der einzelnen REs wird durch die Anzahl von Operationen unter Verwendung dieser Rechenelementen definiert.

In vielen wissenschaftlichen Simulationen kann der anfängliche Lastausgleich im Verlauf einer Simulation nicht erhalten werden. Sowohl äußere Faktoren (z.B. Staus in dem Cluster-Netzwerk) als auch Simulationsdynamiken (z.B. die Bewegung der Partikel im Rechengebiet) können die Lasten von REs verändern und dadurch ein Lastungleichgewicht verursachen. In solchen Situationen benötigen die überlasteten REs mehr Zeit, um ihre Aufgaben zu beenden. Währenddessen bleiben die unterbelasteten REs untätig und warten auf die überlasteten REs. Dadurch verlängert sich die Simulationslaufzeit und es erzeugt einen redundanten Energieverbrauch und eine Verschwendung von Ressourcen, die wir auf ein Minimum reduzieren wollen. Um mit diesem Lastungleichgewicht effizient umzugehen, wurde die dynamische Lastverteilung (DLV) entwickelt. Im Gegensatz zu der statischen Lastverteilung versucht DLV eine neue Subdomain-zu-RE

Zuordnung zu vermeiden.

In dieser Doktorarbeit untersuchen wir DLV für parallele wissenschaftliche Simulationen. Zunächst betrachten wir parallele Simulationen basierend auf Domain-Dekomposition [5]. Wir analysieren verschiedene Modelle aus der Literatur, um die Rechenlast, bzw. Kosten der Subdomain, zu modellieren. Wir argumentieren, die beste Art die Rechenlast zu modellieren, ist sie als unteilbare und reellwertige Zahl zu definieren. Da die Subdomainkosten eng mit den gemessenen RE-Zeiten, die auch als reelle Zahlen gespeichert sind, verknüpft sind, ist es natürlich, die Lasten als unteilbare und reelle Zahlen zu speichern. Danach besprechen wir, wie ein DLV-Protokoll geplant werden sollte, um eine schnelle skalierbare Lösung für einen neuen Lastausgleich zu finden.

In diesem Zusammenhang analysieren wir die verteilten DLV-Protokolle basierend auf dem Balancing Circuit Model (BCM) [6] und entwickeln theoretische Grenzen für das erwartete Lastungleichgewicht, wenn unteilbare und reellwertige Lasten auf REs ausbalanciert werden müssen. Weiterhin erklären wir mehrere verteilte DLV-Protokolle, die angewandt werden können, um das Lastungleichgewichtsproblem effizient zu lösen. Durch die Kombination der besten Eigenschaften dieser DLV-Protokolle entwickeln wir einen `HybridBalancer`, der das anfängliche Lastungleichgewicht in einem Netzwerk von mehr als einer Million REs in wenigen DLV-Schritten dreimal verringern kann.

Zweitens nehmen wir uns ein sequenzielles Monte-Carlo-Verfahren [11–13], die Partikel-Filter, als Beispiel und diskutieren, wie es von den DLV-Protokollen profitieren kann. Diese Partikel-Filter (PF) [7–10] sind entwickelt worden, um Zustandschätzungsprobleme in nichtlinearen und nicht-Gaußschen Zustandsraummodellen unter Verwendung von Partikeln als Hauptdatenstruktur zu lösen. Hier konzentrieren wir uns auf die Verfolgung von subzellulären biologischen Objekten in Bildern. Wir diskutieren mehrere parallele PF-Algorithmen, die die Verwendung von HPC-Systemen ermöglichen, um solche Verfolgungsprobleme schneller lösen zu können.

In parallelen PF-Algorithmen gibt es zwei Arten vom Lastungleichgewicht: Das Partikel-Ungleichgewicht und das Partikelgewicht-Ungleichgewicht. In

dieser Arbeit verbessern wir zunächst die schon bestehenden parallelen PF-Algorithmen, indem wir fortschrittliche DLV-Strategien einbauen. Diese DLV-Methoden helfen uns die Verfolgungseffizienz um das 20-fache zu erhöhen und die Laufzeit der Simulationen um 9% zu kürzen. Später stellen wir einen neuen parallelen PF, die Box Exchange Methode (BEM), vor. Diese Methode reduziert den Kommunikationsaufwand in der DLV-Phase, wodurch sich die Ausführung eines parallelen PFs beschleunigt. Es übertrifft den vormals schnellsten parallelen PF um mehr als das 2-fache. Desweiteren gibt uns ein detaillierter Blick auf den klassischen PF-Algorithmus die Möglichkeit eine Annäherungsmethode für PF, die pcSIR, zu entwickeln. Die pcSIR bietet eine sehr ähnliche Verfolgungsgenauigkeit wie der originale PF, liefert jedoch Laufzeitverbesserungen bis auf mehrere Größenordnungen.

Acknowledgments

First and foremost, I would like to express my deepest appreciation to Ivo for his supervision during my PhD studies. He has not been only a vast source of new ideas, but also a great motivator and an exceptional guide in the past four-and-half years. I profoundly thank Ivo for helping me grow as a scientist and for encouraging me to follow my own research ideas. I feel privileged to have been a part of his international and truly interdisciplinary MOSAIC group and to watch it grow, evolve, and experience a new blossom in Dresden. I count myself very lucky to have had the opportunity to work and share fond memories with him in Switzerland, Croatia, Denmark, Greece, and finally (but, for sure not lastly) in Germany. Thank you!

Next, I want to thank all the past and current MOSAICians I had the pleasure to work and to interact with. They all contributed to the stimulating and friendly atmosphere in various ways and made my PhD adventure a very pleasant experience. When I started my PhD studies, Omar (aka Alter) offered me not only help in getting started with the PPM library, but also a great friendship that went far beyond the office walls. A special thanks goes to Gong (aka Gong), who literally got in a van in Zurich with me and traveled to Germany to kickstart a new beginning in Dresden at MPI-CBG. His fellowship in the past years has been very meaningful. I thank Janick (aka Herr Kollege) for great office times and for bearing with me whenever I was nervously double-kick-pedaling the floor in CAB building. I also wish to thank Sylvain for many fruitful discussions. He was very resourceful with his problem-finding skills and his witty personality

was beyond joy to have around. I wish to thank Birte for introducing me to the world of computational biophysics and Rajesh for his invaluable contributions to the group dynamics.

After joining MPI-CBG, I had the enjoyment of welcoming further awesome MOSAICians to the group. First of all, Pietro (aka Del Pietro) always amazed me with his critical thinking and almost endless "theoretical" rebutting skills. Thanks to him I think thrice now before getting into any argument. I want to thank Sophie (aka Zofi) and Josefine (aka Josi) for bringing additional laughter and joy to the group. I really enjoyed our conversations and extracurricular activities with you, lads. I feel privileged to get to know Yaser (aka Mösyö). I highly respect and admire his humble and hard-working personality. I am also thankful to Bevan for making the group more alive with his cheerful and humorous personality. Furthermore, I thank George (aka Yorgos) for bringing in some more Mediterranean mindset and Ulrik for various discussions. Lastly, it has been nice to have Xun around - another fellow crazy about European football.

Special thanks go to Anas and Dalgıçlar for years of awesomeness and true friendship. A big thanks goes to Onur, Salih, Hosam, Ferit, Damla, Milan, Ayşe, Ceren, and all other good friends of mine in Dresden and Zurich. I had an amazing time in both cities thanks to all of you! Further, I would like to mention my students: Kevin, Roman, Matthias, and Fabian. Thanks so much for your contributions!

I would like to express my sincere gratitudes to Nicolas Fiétier and Ihor Smal. They both played crucial roles in my scientific journey in the past years. I want to thank Nicolas for his invaluable help with the Maxwell project. Working closely with him, I learned a lot from his experience and his systematic way of approaching complex problems. Moreover, whenever I felt lost, his optimism was there to boost me. The project would not have progressed this far without him.

I met Ihor when he visited our group in summer 2013. He introduced me to particle filters and we briefly discussed load imbalance situations in parallel particle filtering applications over a cup of coffee. This thesis is largely based on the ideas we discussed back then and therefore I am truly

thankful to him. It has been also my utmost pleasure to get to know him personally and work with him closely. This thesis would not have been the same otherwise.

I am very grateful to have Prof. Peter Widmayer and Prof. Erik Meijering on my thesis committee. I feel honored by their interest in my work. I would like to thank Prof. Peter Widmayer - again - for kindly agreeing to become my host at ETH Zurich after I moved to Dresden.

Last but not least, I wish to thank my parents and my sister with my deepest heartfelt gratitude. Your unconditional support and love give me strength to go further every day. I dedicate this thesis to you.

Dresden, April 2014

Ömer Demirel

Contents

Nomenclature

ARNA	Adaptive RNA
BCM	Balancing circuit model
BEM	Box exchange method
BPF	Box particle filter
DLB	Dynamic load balancing
DRA	Distributed resampling algorithm
GPU	Graphics processing unit
HF	Histogram filter
HPC	High-performance computing
IPC	Interprocessor communication graph
IS	Importance sampling
KF	Kalman filter
MPI	Message passing interface
pcSIR	Piecewise constant sequential importance resampling
PDE	Partial differential equations

PDF Probability density function

PE Processing element

PF Particle filter

PPF Parallel particle filtering

PSF Point-spread-function

RMSE Root mean square error

RNA Distributed resampling algorithm with non-proportional allocation

RPA Distributed resampling algorithm with proportional allocation

SIR Sequential importance resampling

SIS Sequential importance sampling

SLB Static load balancing

SMC Sequential Monte Carlo

SNR Signal-to-noise ratio

List of Figures

Introduction

Since the advent of the first programmable computer Colossus [14], which was designed and constructed by mathematician Alan Turing and his colleagues in England in late 1943, we have witnessed an ever-increasing role of computers in science and engineering. Rising up as the third pillar of scientific investigation alongside theory and lab experimentation, this younger field of science is termed as *computational science* or *scientific computing*. Computational science is concerned with the reconstruction or prediction of physical phenomena and technical processes of scientific interest by developing mathematical models and quantitative analysis techniques, which are later tested and validated with the help of computer simulations.

Over the past decades, computer simulations boosted the quest of answering many scientific questions and paved the way to discovering invaluable insight into numerous engineering challenges, which would not have been tackled otherwise with traditional experiments. Success stories of numerical simulations range across scales and disciplines, e.g., down from atomic scale (e.g., protein folding in molecular biology [15]) up to macroscopic scale (e.g., studies of perturbations of planetary systems [16]).

The advancements in simulation codes has closely followed the breakthroughs in microchip technology. For considerable time, they enjoyed "free speedup" as the chip performance of computers doubled almost every 18 months in line with Moore's Law [17, 18]. Back then, the trend was

buying new, more advanced hardware to have a faster execution of legacy codes and improving the sequential performance of simulation codes by introducing faster algorithms. However, with the inception of multicore processors and later many-core processing units (e.g., graphics processing units (GPUs)) in mid-2000s, a paradigm shift happened: It became clear that future simulations need to exploit parallel hardware infrastructure to further shorten application runtime since clock rates of cores stagnated.

The emergence of parallel systems has again emphasized an important recipe for a fast parallel code design, which can be summarized in two fundamental engineering principles: (i) *Algorithm engineering* is concerned with improving an existing algorithm or developing a new algorithm, which offers a much lower computational (i.e., runtime and/or space) complexity; (ii) *implementation engineering* deals with efficient implementation of an algorithm such that it utilizes the underlying hardware. Exploiting the synergism of both algorithm and implementation engineering principles usually leads to a better hardware-software co-design [19] and thus, shorter runtime of simulation codes and less energy consumption, which is a vital issue in future HPC systems [20, 21].

In parallel scientific numerical applications, computational tasks or data are distributed across participating PEs. Despite initial equal workload sharing, the workloads of PEs may however become easily uneven as the simulation progresses. This situation forces underloaded PEs to remain idle and wait for overloaded PEs to finish their tasks. This *load imbalance* wastes energy and computational resources. With the upcoming emergence of exascale systems[1], we cannot afford such energy-wasting situations in the future and novel solutions need to be researched.

As a remedy to load imbalance in parallel simulations, *dynamic load balancing* [22] (DLB) has been introduced for distributed-memory systems. The idea is to execute a fast DLB protocol to make the loads on PEs as even as possible in shortest possible time. This would help to reduce redundant use of computational resources and help scientists obtain their simulation results much faster.

[1] Large computer systems that can execute 10^{18} floating point operations per second.

In this thesis, we discuss various DLB protocols that one can employ in two main types of parallelization techniques for scientific codes. In parallel applications, two typical domain decomposition schemes are used to parallelize the workload. The first one is called *geographic domain decomposition* approach where the initial computational domain is divided into smaller subdomains, which are later assigned to PEs. The second approach is usually termed as *non-geographic domain decomposition* where computational elements are shared by the participating PEs. This thesis explores ideas how to apply DLB in both domain decomposition approaches efficiently. An overview of the thesis can be seen in Fig. 1.

In this work, we first discuss *distributed* DLB protocols one can employ in generic parallel scientific numerical simulations based on a *geographic domain-decomposition* approach [5]. A vast majority of scientific numerical simulations adopt this approach to exploit parallel computers. In this approach, the computational domain is decomposed into smaller subdomains and each PE is given a set of these subdomains to operate on. During the course of a simulation, the computational cost of these subdomains may change and a DLB step may be required to balance the workloads amongst PEs. This thesis focuses on distributed DLB protocols to overcome such load imbalance problems in parallel simulations and analyzes performances of several such DLB protocols. Later, we investigate a specific particle method, namely *particle filters* (PFs), which are based on *non-geographic domain decomposition* approach and discuss how one can solve two different types of load balancing problem efficiently.

We present a number of scientific applications, which are solved numerically using *particles* as main computational element, and explore possibilities to make these simulation codes run faster by applying *algorithm* and *implementation engineering* principles with a DLB perspective. Before explaining the details of our contributions, we would like to give some preparatory information on the key concepts such as particle methods, domain decomposition, and distributed DLB algorithms, which constitute the backbone of this thesis.

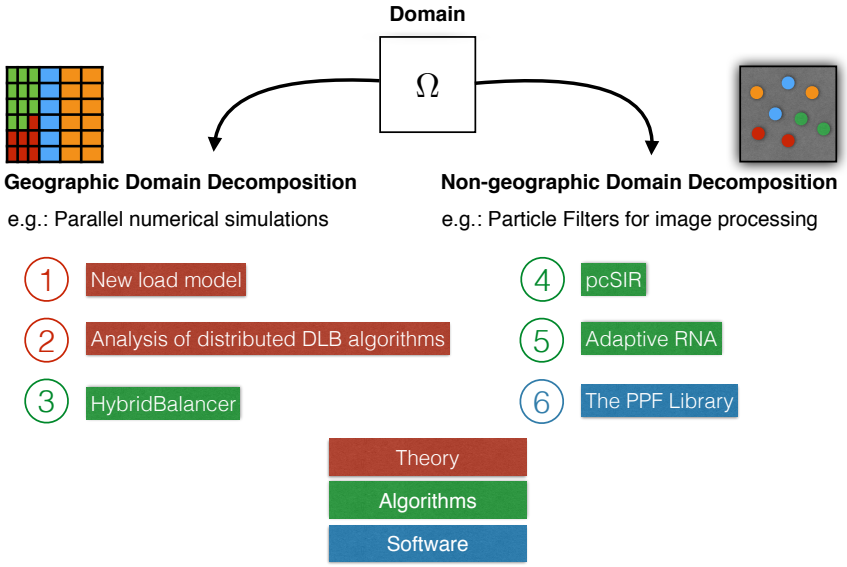


Figure 1: This thesis investigates DLB solutions to scientific simulation codes that are parallelized based on (non)-geographic domain decomposition approaches.

	deterministic	stochastic
continuous	PDEs	SDEs
	diffusion	reaction-diffusion with low molecule numbers
discrete	interacting particles	random events
	molecular dynamics	Brownian agents

Figure 2: Four "kingdoms" of simulation models with their representative applications are shown [23].

Particle Methods

Many simulated systems belong to a subclass of so-called spatiotemporal systems. They can be distinguished mainly in two independent dimensions: discrete - continuous and deterministic - stochastic. Different modeling techniques and therefore different computational methods are available depending on characteristics of the chosen system. An overview of most common modeling techniques along with examples of spatiotemporal systems are given in Fig. 2.

All numerical simulations that aim to investigate a spatiotemporal system numerically, employ at least one, or several types of *computational elements* to discretize the problem in space. The computational complexity of a simulation mainly depends on the number of the employed computational elements and also, on the mathematical operations they undergo throughout the course of the simulation. Prominent types of computational elements in such simulations are computational *lines* (e.g., Finite difference method [24]), *surfaces* (e.g,, Finite element method [25]), *volumes* (e.g., Finite volume method [26]), and *particles* [27, 28]. A computational mesh is composed of vertices and edges connecting them. The nodes usually carry some properties, i.e., a function value specific to the scientific problem at hand. Additionally, the adjacency information (i.e., neighborhood information) of the nodes is given explicitly.

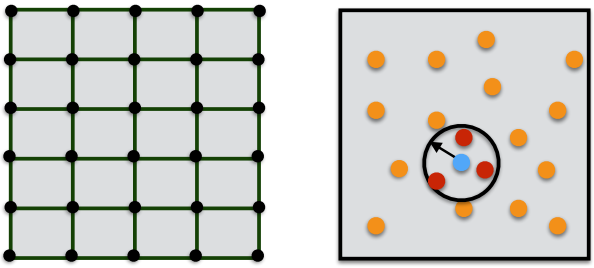


Figure 3: Illustration of a regular Cartesian mesh (left) and particles (right). The neighborhood in a mesh is explicitly given, whereas particles need to search for neighboring particles. In this example, the neighborhood range of the blue particle is shown by a black circle. Three red particles are marked as the neighbors of the blue particle.

Computational particles are data structures that have a location in some space and may carry one or several *properties*. They can *move*, *evolve*, and *interact* with other computational particles. Local searches are required to find other neighboring particles. A pictorial overview of meshes and particles is shown in Fig. 3.

Particle methods provide a unifying computational framework for simulations of continuous and discrete systems. They facilitate seamless analysis of such systems, both stochastically and deterministically. In *continuous deterministic* models, we investigate the temporal and spatial evolution of a smooth field (e.g., velocity, electric charge density, etc.), which are usually modeled by the partial differential equations (PDEs). For example, in computational electromagnetism, quasi-static Maxwell equations are used to model the electromagnetic phenomena. In such applications, differential operators and the electric field are discretized over a set of particles [29]. In computational fluid dynamics (CFD) applications, e.g., in wave simulations [30], the Navier-Stokes equations are discretized again over particles. In these examples, particles present a data structure that acts as a discretization point that moves, evolves, and interacts with other particles according to the application specifications. In *discrete stochastic* systems, particles have usually correspondence to real-world entities (e.g.,

atoms in a molecular dynamics simulation [31]) or to samples generated randomly from a probability distribution over the domain in a Monte Carlo simulation [32]. Irrespective of the chosen system, *particles* can be seen as a prevalent data structure, which has been already adopted in many scientific simulations due to its simplicity and versatility.

In this thesis, special focus is given to a particle method called *particle filtering*. Particle filters (PFs) [7–10] are SMC methods [11–13] developed to solve estimation problems optimally in nonlinear and non-Gaussian state-space models. More precisely, they serve as a helpful computational tool to accurately estimate unknown quantities in noisy measurements, which occur commonly in various real-world applications and research efforts including, but not limited to computer vision [33–36], robotics [37–41], transportation systems and urban logistics [42–44], econometrics and finance [45–49], renewable energy research [50], sports [51], target tracking [52–59] among many others.

Unlike the Kalman filter [60] and its variants [61], particle filters (PF) do not use a fixed functional form of the posterior probability density function (PDF). Instead, they employ a finite number of points, called "particles", to discretely approximate PDF in state space [62]. Being the main data structure in PFs, particles carry application-specific properties and they evolve as the simulation progresses.

In this work, we focus particularly on the tracking of sub-cellular objects in biomedical images. This operation is concerned with the detection of bright spots in the images, tracking their motion over time, and reconstructing their trajectories.

Domain Decomposition Approach and Distributed DLB

Years of interplay between advancing computing systems and the demand for studying scientific problems of greater complexity gave rise to exploiting hardware parallelism (e.g., multi-core computer systems) to handle ever-increasing computational complexity of the simulation codes. How-

ever, this fact led to several difficulties, so-called *gaps* [5], in parallel HPC: (i) *Performance gap*: Difficulty of sustaining a high computational performance of parallel scientific simulations on high-end supercomputing clusters, (ii) *knowledge gap*: The protracted development of hardware-aware simulation codes for distributed computer architectures, (iii) *reliability gap*: The mean-time between failures is getting shorter than the runtime of a simulation as computer systems get larger, and (iv) *data gap*: Increasing difficulty of handling large simulation data.

To minimize the above-mentioned gaps, many parallel software libraries and frameworks are introduced to provide abstractions at different levels to reduce the code development time, and to enable efficient, fast execution of simulations on computer clusters. On the low-level abstraction, OpenMP [63] and pthreads [64] are libraries used for shared-memory parallelism, whereas the Message Passing Interface (MPI) [65] and MapReduce [66] are popular programming models for simulations and data processing on distributed-memory systems. Middle-level libraries usually utilize the low-level libraries but hide most of the implementation difficulties arising using these low-level libraries. Still, a certain amount of knowledge about parallel computing is required to fully optimize the runtime of simulations. Some examples include the Parallel Particle-Mesh Library (PPM) [67, 68], PETSc [69], Trilinos [70], POOMA [71], NWChem [72], and NAMD [73]. Lastly, high-level abstraction libraries such as Swarm [74] and FFTW [75] come with a collection of domain-specific numerical solvers or tools to accelerate the code development.

Learning efficient parallel programming and then applying it in a specific domain of science takes years of expertise. Middleware libraries such as the PPM library are designed and implemented to make this process as quick as possible, allowing non-computer scientists to develop their parallel numerical codes rapidly and run them on large computing systems efficiently. They provide portable and transparent interfaces for developing particle and/or mesh-based numerical methods on parallel distributed-memory computers.

The vast majority of scientific numerical simulations is parallelized using a *domain-decomposition* approach [5]: The simulation starts with an initialization phase during which the computational domain is decomposed into

disjoint *subdomains* that are then distributed across processing elements (PEs). The initialization step is followed by a *time-step loop* or a series of iterations, executed in parallel by each PE on its corresponding subdomain(s) until an application-specific termination criterion is reached. At the end of each iteration or time step, PEs communicate with their neighbors to ensure solution continuity across subdomain boundaries.

The parallel efficiency of such a numerical simulation is determined by the communication overhead between PEs and by the global load balance. While the former is defined by the employed numerical solver and the geometry of the subdomains, the latter depends on the simulated dynamics.

Each subdomain contains a (large) number of computational elements, such as mesh cells and/or computational particles. Together with the algebraic operations to be executed on them, this defines the *computational cost* of each subdomain. During initialization of a simulation, subdomains are distributed to PEs such as to balance the computation wall-clock time across PEs. The goal of this *static load balancing* (SLB) is to ensure that all PEs finish a single iteration of the simulation in about the same time, so that no PE needs to wait for another one before they can exchange the boundary information and enter the next time step. The graph describing this inter-process communication (IPC) is defined by the assignment of subdomains to PEs and can be used to optimize the communication schedule. This initialization phase is typically handled efficiently by scientific simulation libraries, such as PPM, NAMD, and PETSc.

As the simulation steps forward through iterations or time, the initial load balance may worsen due to both intrinsic and extrinsic factors. Intrinsic factors include the migration of computational elements (e.g., particles) from one subdomain to another due to, e.g., a simulated flow. Other intrinsic factors are particle birth/death processes or the dynamic refinement or coarsening of meshes in order to adapt to the evolving numerical solution. Extrinsic factors that worsen load balance include fluctuations in the background load of the machine and network congestion. An emerging load imbalance is typically quantified by the absolute load difference between the most overloaded (i.e., slowest to complete a time step) and the most underloaded PE.

Since a growing discrepancy deteriorates the parallel efficiency of the simulation, it is necessary to periodically re-balance the load assignment. One way to re-assign the loads is to stop the simulation and a fresh domain decomposition is executed. This, however, requires global communication across all PEs, limiting scalability of this load balancing scheme to very large machines. DLB protocols aim to avoid this overhead by delaying re-initialization to the extent possible. A DLB protocol operates on the existing domain decomposition and is only allowed to migrate (i.e., re-assign) subdomains from one PE to another, without changing the subdomains as such. Yet, the communication overhead generated by a DLB protocol is not zero, since the contents (mesh nodes and/or particles) of a migrating subdomain need to be communicated between the two involved PEs. In addition, every DLB protocol requires consensus between all PEs about how to migrate subdomains.

Distributed DLB protocols have emerged as a viable solution to DLB problems in parallel computing. In such protocols, the need for a global communication is removed by allowing each PE to communicate with its imminent neighboring PEs. Instead of a global communication, several rounds of these local communications are executed. Distributed DLB protocols offer a more scalable parallel execution of scientific codes than centralized DLB protocols and thus need to be analyzed more in detail.

Motivation

The overall aim of this thesis is to help improve the parallel performance of scientific numerical simulations on distributed-memory systems by means of DLB techniques.

We first discuss distributed DLB techniques for general parallel scientific computing based on a domain-decomposition approach [5]. Later, we have a closer look at parallel particle filters and present several DLB algorithms that help relax the *particle imbalance* and *particle weight imbalance* problems. In addition, analyzing PF algorithms enables us to develop faster approximate PF algorithms. A summary of our contributions is as follows:

(i) Theoretical investigation of DLB schemes

(ii) Empirical analysis of distributed DLB protocols in massively parallel

simulations

(iii) A novel fast approximation algorithm for particle filtering (pcSIR)

(iv) Analysis of existing distributed resampling algorithms (DRAs)

(v) Improved versions of existing DRAs

(vi) A novel and more efficient parallel particle filtering algorithm

(vii) A new parallel particle filtering (`PPF`) library that implements all new and old DRAs efficiently on parallel computers

Outline

Chapter 1 The importance of scalable DLB strategies is discussed and several distributed DLB protocols are compared with each other. Promising DLB candidates are then benchmarked in typical massively parallel simulation setups that reflect realistic load imbalance situations in scientific numerical simulations with more than one million PEs.

Chapter 2 This chapter explains the theory behind particle filters before we discuss DLB ideas for parallel particle filtering algorithms. It introduces Bayesian filtering and Monte Carlo methods with a special focus on statistical inference. The chapter concludes with a general description of the particle filtering algorithm.

Chapter 3 In this chapter, we introduce a novel approach to classic particle filtering, where an approximation based on the piecewise constant representation of a likelihood function is introduced. This new algorithm (pcSIR) improves the runtime of sequential PF algorithm and keeps the accuracy of the PF intact.

Chapter 4 Here, we give an insight into existing DRAs that are used in parallel PFs and the DLB problems that they suffer from. Moreover, we introduce our novel software, the `PPF` library, which implements all these DRAs along with new ones introduced in the next chapter.

Chapter 5 In this chapter, we discuss our DLB contributions to the existing DRAs as well as a novel parallel PF algorithm called BEM, which reduces communication overhead during the DLB step and outperforms fastest parallel PF algorithm to date.

Chapter 6 This chapter summarizes the contributions of this thesis in research areas of DLB, parallel particle filtering, and the `PPF` library.

Chapter 7 Lastly, this chapter points to possible directions for future research and concludes the thesis.

CHAPTER

ONE

Distributed dynamic load balancing in massively parallel scientific numerical simulations

1.1 Introduction

In a typical parallel numerical simulation, the computational domain is decomposed into smaller subdomains, which are later assigned to processors. This task has to be carried out carefully such that the computational load on each processor is even and the communication overhead amongst the processors is minimized. If particles move or the mesh gets refined during the simulation, initial domain decomposition may get invalidated. This would then require re-decomposition of the computational domain and redistribution of computational elements.

One way to handle growing load discrepancy is to use a static load balancing (SLB) technique. In a SLB approach, the simulation is halted and re-initialized as soon as the accumulated load imbalance justifies the cost of determining a new domain decomposition and distributing the resulting subdomains across PEs afresh. Several heuristics, such as the Stop-At-Rise method [76], are available to determine when the overhead of re-initialization is amortized by the expected future gain in load balance. All of them, however, require global communication across all PEs, limiting

their scalability to very large machines.

On the other hand, DLB protocols aim to solve the load imbalance problem by introducing smaller perturbations. With all these small "corrections" to the load imbalance, DLB tries to avoid a complete re-distribution of subdomains. A DLB protocol operates on the existing domain decomposition and is only allowed to migrate (i.e., re-assign) subdomains from one PE to another, without changing the subdomains as such. Yet, the communication overhead generated by a DLB protocol is not zero, since the contents (mesh nodes and/or particles) of a migrating subdomain need to be communicated between the two involved PEs. In addition, every DLB protocol requires consensus between all PEs about how to migrate subdomains. In literature, there are three main types of DLB strategies that deal differently with this communication overhead.

1.1.1 Dynamic Load Balancing Strategies

DLB strategies are classified into three categories depending on the communication protocol used to guarantee consensus about subdomain migration. The first DLB category uses a centralized paradigm [22, 77], where one master PE collects the load information from all other PEs, determines the subdomain migration plan, and broadcasts this plan to all PEs. Upon receiving the plan, all PEs start exchanging subdomains in parallel. This incurs an additional serial fraction in the program, as the migration plan is determined sequentially by a single PE, and requires global communication. The scalability of centralized DLB schemes is hence limited.

The second category comprises hierarchical DLB strategies [78–81], where the PEs are hierarchically clustered into groups and a centralized DLB protocol is executed in parallel within each group. Although several issues of centralized DLB protocols are relaxed by hierarchical strategies, performance may be hampered by excessive data collection at the bottom of the hierarchy and by redundant work done at multiple levels [82].

The third DLB category uses a decentralized, distributed communication protocol, where each PE communicates only with a bounded number of neighbors that is independent of the total number of PEs. This implies

that consensus has to be reached based on local information only. Distributed DLB schemes hence require several rounds of local communication to achieve load balance, but they are in theory scalable to arbitrarily large numbers of PEs.

Distributed DLB protocols come in two flavors with either one-to-one or one-to-all nearest-neighbor communication. In *diffusion*-based algorithms [83, 84], the load of a PE is balanced concurrently with all neighbors. One-to-one protocols include *dimension exchange* [83] and the *matching model* [85], where each PE selects a single neighbor in each round, and loads are balanced in these isolated pairs. This is formalized by the *balancing circuit model* (BCM), which has been shown to asymptotically produce better local load balance than diffusion-based models [6].

1.1.2 Load Models

In scientific numerical simulations, the computational costs of the subdomains can be considered statistically independent of each other. This renders the DLB problem tractable, as it is not reducible to graph partitioning.

The complexity analysis of distributed DLB algorithms (i.e., diffusion-based, matching model, and BCM) covers substantial ground in theoretical computer science. For theoretical analysis, loads are usually modeled as either (i) divisible, real-valued loads in the *continuous load model* or (ii) indivisible, unit-token loads in the *discrete load model*. Applied to numerical simulations, the former assumes that subdomains carry real-valued computational costs and each subdomain can be *arbitrarily* subdivided during the DLB process. The latter load model considers integer subdomain costs that are multiples of a load unit that cannot be further divided.

Theoretical analysis of the continuous load model in diffusion-based DLB schemes has been done using spectral analysis of Markov processes on graphs [86, 87]. The analysis has also been extended to distributed gossip algorithms that reach a consensus [88] or perform a collective operation (e.g., averaging) in the network [89]. Similar analysis also applies to the discrete load model [85, 90]. Some authors used randomized round-

ing to quantify the difference between the continuous and the discrete cases [86, 91, 92]. Sauerwald and Sun [93] showed tight bounds on the deviation between the continuous and the discrete models in the BCM.

While these two load models paved the way for theoretical analysis of DLB, they do not accurately describe the situation in scientific numerical simulations. While the costs of subdomains change from time step to time step, they are constant during the DLB process as such. The main reason for using DLB, as opposed to re-initialization followed by SLB, is to reuse the existing domain decomposition. While in principle, one could also allow a DLB scheme to shift subdomain boundaries, hence enlarging some subdomains and shrinking others, this would potentially invalidate topological requirements of the domain decomposition and change subdomain neighborhood relations, hence invalidating the communication schedule of the simulation. Determining a new schedule would again require global communication. This discourages the use of the continuous load model. The discrete model would be applicable if loads were to be balanced in terms of the numbers of particles and/or mesh points per PE. These are discrete, indivisible tokens. However, in practical applications loads are to be balanced in terms of wall-clock execution times, since different mesh nodes or particles may require different operations to be evaluated on them. This renders the cost a real-valued variable with no indivisible atom.

We hence consider the loads to be indivisible and real-valued [94] in a network of homogeneous PEs that all have the same clock rate and memory size. Furthermore, we neglect background fluctuations in machine load that may affect the costs. In this *fixed load model*, subdomains are not altered or subdivided by the DLB protocol and their costs are directly defined as the wall-clock execution times. An illustration of the three load models is shown in Fig. 1.1. As shown in Appendix A, the theoretical bounds derived for the discrete load model [93] also apply to the fixed load model in a BCM framework.

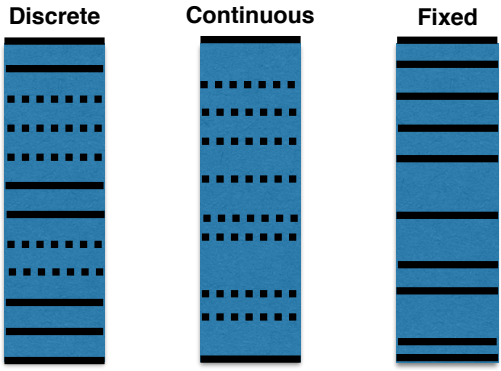


Figure 1.1: Illustration of different load models in DLB analysis. In the discrete load model (left), all loads (solid lines) are integer multiples of an indivisible load unit (dashed lines). The continuous load model (middle) assumes loads to be real numbers that can be arbitrarily subdivided during DLB. The fixed load model (right, considered here) assumes all loads to be real numbers that cannot be subdivided or changed during the DLB process.

1.2 Contributions

Here, we study the fixed load model as a realistic model for subdomain costs in scientific numerical simulations based on domain-decomposition parallelism. Using analysis techniques developed earlier [93], we derive the expected *discrepancy* (i.e., the load difference between the heaviest and the lightest PEs) between the fixed load case and the continuous case. With some modifications to the original theoretical framework, we show similar tight bounds for the discrepancy of balancing indivisible, real-valued loads in a BCM protocol.

Following that, we use numerical experiments of the proposed model to compare the performance and communication overheads of three simple DLB protocols: `Greedy`, `SortedGreedy`, and `Gradient`. We empirically show that `SortedGreedy` balances loads faster than a classical `Greedy` method, requiring fewer iterations of the DLB protocol. Moreover, `SortedGreedy` achieves a better load balance than `Greedy`. `Gradient` yields less balanced load distributions than `SortedGreedy`, but requires much fewer subdomain migrations, hence reducing the communication overhead.

In order to assess the performance based on both the load balance and the induced communication overhead, we introduce a metric S that accounts for the quality of the resulting load balance and for the number of load migrations required to achieve it. We compare three DLB protocols in a local setting where two nodes with different total loads undergo the DLB phase. We observe that `SortedGreedy` and `Gradient` are favorable algorithms to solve this local load difference problem.

We then combine these two algorithms into a `HybridBalancer` and test it for realistic DLB problems in simulations of a large-scale network with one million PEs forming realistic communication topologies as found in numerical simulations. The results show that after a few iterations of `SortedGreedy`, an initial load imbalance is reduced by up to a factor of three. While `Gradient` can reduce the initial load imbalance only by 50%, it yields a much smaller communication overhead, resulting in up to a factor of seven better S over `SortedGreedy`. We observe that the load imbalance decreases rapidly during the first few iterations of the scheme, and that the following iterations contribute less. Based on this observa-

tion, we propose `HybridBalancer` for practical application in massively parallel numerical simulations. In the first DLB round, `HybridBalancer` uses `SortedGreedy` to reduce the load imbalance as much as possible. For the subsequent rounds, it switches to `Greedy` in order to limit the communication overhead.

1.3 DLB Algorithms for Locally Balancing Loads

In BCM, each PE in the IPC graph selects a single neighboring PE in parallel at a time. These pairs of PEs are called a *matchings* (see also notation and terminology in Appendix A). The goal in each matching is to distribute the loads between those two PEs as evenly as possible. In order for all matchings in the IPC graph to be able to operate concurrently, they need to be independent, i.e., no PE must occur in more than one matching at a time. Such matchings can efficiently be found using a near-minimal edge coloring of the IPC graph. In many parallel numerical simulations, such a coloring is already available from the communication schedule used by the PEs to exchange boundary information [67] and it can be directly re-used at no additional cost. Otherwise, such a coloring can be efficiently computed using an approximation algorithm [95].

A good DLB protocol should minimize the load imbalance independently in each matching. This *local balancing problem* can be formalized as an offline balls-into-bins problem [96, 97] with two bins (see Appendix B). Here, balls represent subdomains, the non-negative ball weights represent subdomain costs, and the two bins are two PEs of the matching.

The classical balls-into-bins problem [98, 99] considers sequential placement of m balls into n bins such that the bins are maximally balanced. Historically, the problem is categorized by the type of balls (e.g., uniform [100, 101] vs. weighted [102–105]), by the number of bins a ball can choose from (e.g., single-choice vs. multi-choice [91]), and by the number of balls (e.g., $m = n$ [96] vs. $m > n$ or $m \gg n$ [97]). In applications such as load balancing, hashing, and occupancy problems in distributed computing [97, 102, 106] the d-choice variant and its subproblem, the two-choice variant have been the main focus. In the present case of balls having

individually different weights, Talwar and Wieder [103] have shown that as long as the weight distribution has finite second moment, the weight difference between the heaviest and the average bin is independent of m. Peres *et al.* [104] introduced the $(1+\beta)$-choice process analysis, and for $\beta = 1$, the discrepancy has a bound of $\Theta(\log \log n)$ even for the case of weighted balls. Dutta *et al.* [105] introduced the `IDEA` algorithm, which provides a constant discrepancy with high probability even in the heavily loaded case ($m \gg n$).

In the offline version of the problem, we are given the complete set of balls (i.e., the loads on matching PEs) *a priori*. We define the discrepancy as the weight difference between the heaviest and the lightest bin. We do not restrict the distribution from which the balls sample their weights. For simplicity, we assume that a ball can be placed into either bin, thus $d = n = 2$. As shown in Appendix B, the discrepancy is expected to decrease by the mean weight $\bar{W}$ of all balls in each iteration. The final discrepancy remains larger than $2W_1 - \bar{W}m$ for a total of m assigned balls with W_1 the weight of the heaviest ball.

1.3.1 Greedy Method

An online version of the `Greedy` algorithm has been previously proposed [100, 101] and later extended to the case of weighted balls [103]. In the offline version, two matching PEs u and v exchange the costs of all of their subdomains, such that both of them have complete information about all local loads, and all subdomains are dissociated from their previous PEs. Then, `Greedy` linearly iterates though the loads (subdomains) and assigns each load to the PE with the so far smallest total load. The n-bin version of `Greedy` is given in Algorithm 1.

1.3.2 SortedGreedy Method

`SortedGreedy` sorts the balls in order of descending weights before applying `Greedy`, hence assigning the heaviest ball first and proceeding to lighter and lighter balls. If the ball weights (i.e., subdomain costs) are samples from a uniform probability density function, the final discrepancy G_m af-

Algorithm 1 `Greedy` algorithm

1: **procedure** Greedy[N]($U_{1...N}$, W) ▷ Given are a set W of m balls and the bin arrays $U_{1...N}$
2: $U_1[1] \leftarrow$ W[1]
3: $p_{2...n} \leftarrow 1$ ▷ Initialize the pointers for all bins
4: $p_1 \leftarrow 2$ ▷ First bin has already one ball in it.
5: **for** $i = 2 \rightarrow m$ **do** ▷ Give remaining $m - 1$ balls sequentially to lightest bin
6: idx $\leftarrow findLightestBin(U_{1...N})$ ▷ Find the ID of the lightest bin which is the one with least current sum
7: $U_{\text{idx}}[p_{\text{idx}}] \leftarrow$ W[i]
8: $p_{\text{idx}} \leftarrow p_{\text{idx}} + 1$
9: **end for**
10: **return** $U_{1...N}$
11: **end procedure**

ter placing all m balls is bounded from below by: $G_m \geq W_1 - \sum_{i=2}^{m} W_i \geq 2W_1 - \sum_{i=1}^{m} \frac{1}{i}$, where W_i is the weight of ball i (see Appendix B). Moreover, since the balls are sorted in order of descending weights, W_m is the minimum ball weight and fluctuations in the discrepancy decrease as $i \to m$.As shown in Appendix B, the final discrepancy obtained by `SortedGreedy` decreases as m increases. A sufficiently fine granularity of the domain decomposition is hence important for achieving good load balance in a BCM.

If the subdomain costs are sampled from a uniform distribution over the interval $(0, 1]$, we can use a distribution-based sorting algorithm, such as bucketsort, Proxmap-sort [107], or flashsort [108]. Since these algorithms are not comparison-based, the $\Omega(m \log m)$ lower bound for comparison-based sorting does not apply to them. For example, Proxmap-sort [107] has an average time complexity of $O(mk) = O(m)$, where $k < m$ is the content number of the "buckets" used for sorting. For flashsort, $k = 0.42m$ is found a good value in empirical tests [108].

For non-uniform or unknown weight distributions, we resort to efficient comparison-based sorting algorithms, such as mergesort or quicksort [109], which have an average time complexity in $O(m \log m)$. Depending on the specific sorting algorithm, the worst-case complexity can also be in $O(m \log m)$. Highly optimized implementations of these algorithms are commonly available. The pseudocode of `SortedGreedy` is given in Algorithm 2.

Algorithm 2 `SortedGreedy` algorithm

1: **procedure** SortedGreedy[n]($U_{1...N}, W$) ▷ Given are a set W of m balls, and the bin arrays $U_{1...N}$
2: sortedW ← *quicksort*(W) ▷ Sort the array in descending order (e.g. using quicksort)
3: **return** Greedy[n]($U_{1...N}$,sortedW)
4: **end procedure**

1.3.3 Gradient Method

`Gradient` expands upon `SortedGreedy` by accounting for the previous subdomain-to-PE assignment in order to keep the number of subdomain migrations low, hence reducing the communication overhead incurred by the DLB protocol. The price one has to pay for this decrease in communication overhead is a less well-balanced load distribution after DLB. In every matching of `Gradient`, the heavier node, say u, transfers some of its loads to v, such that the load difference between u and v is minimized. Intrinsically, the heavier node finds the minimum number of loads to send in order to maximize load balance. For this, the heavier PE u sorts its local loads in order of decreasing weights and sends as many loads to v as necessary to decrease its total weight to around (plus/minus one load) the expected mean weight. The pseudocode of `Gradient` is shown in Algorithm 3.

1.3.4 Simulation Experiments on Locally Balancing Loads

We empirically compare the three offline balls-into-bins algorithms `Greedy`, `SortedGreedy`, and `Gradient` in computer simulations. We test how well they balance loads between two PEs, i.e., we consider only a single matching. We then extend the analysis to whole IPC graphs in the subsequent section.

We test cases where each PE locally has up to 100 subdomains initially (i.e., up to 200 loads per matching), hence reflecting different granularities of the domain decomposition. Each subdomain cost is sampled from a uniform distribution over the interval $[0, 1]$. All tests are repeated 500 times each. Neighboring PEs may have unequal numbers of subdomains initially, and the average initial load imbalance increases as the total number of subdomains is increased.

We quantify the goodness of load balancing by

$$S = \frac{disc}{\alpha}\,, \tag{1.1}$$

Algorithm 3 `Gradient` algorithm

1: **procedure** Gradient(u, v, loads) ▷ Compute each node's total load; maximum number of elements in *loads* is l
2: $u_{total} \leftarrow$ sum(loads[u,:])
3: $v_{total} \leftarrow$ sum(loads[v,:])
4: $dif \leftarrow abs(u_{total} - v_{total})$ ▷ The load difference should be non-negative
5: **if** $u_{total} > v_{total}$ **then** ▷ Heavier node sends some loads to the other one
6: sender ← u
7: receiver ← v
8: **else**
9: sender ← v
10: receiver ← u
11: **end if**
12: sortedLoads ← *quicksort*(loads[sender,:]) ▷ Sort the *sender* array in descending order
13: newLoads[sender,:] ← sortedLoads
14: nonzero ← nnz(loads[receiver,:]) ▷ Find the number of non-zero elements in *receiver*
15: **for** $i = 1 \rightarrow$ nonzero **do**
16: newLoads[receiver,i] ← loads[receiver,i]
17: **end for**
18: skip ← nonzero + 1
19: **for** $i = 1 \rightarrow l$ **do**
20: **if** $sortedLoads(i) < 2*$dif & sortedLoads(i) $\neq 0$ **then**
21: newLoads[receiver,skip] ← sortedLoads(i) ▷ *receiver* gets the heaviest load from *sender*
22: dif ← dif $-2*$sortedLoads(i)
23: skip ← skip + 1
24: sortedLoads(i) = [ ]
25: **end if**
26: **end for**
27: **return** newLoads
28: **end procedure**

where *disc* is the ratio between the initial discrepancy (i.e., initial load imbalance before DLB) and the final discrepancy achieved by the DLB protocol, and α is the total number of subdomain migrations required. When comparing a DLB algorithm, we use `SortedGreedy` as a baseline and define the relative figure of merit S_{rel} with respect to `SortedGreedy` as:

$$S_{\text{rel}} = \frac{S}{S_{\texttt{SortedGreedy}}} \,. \tag{1.2}$$

This measures the relative performance of a DLB protocol over `SortedGreedy` based on the communication overhead and the final load balance. This metric is independent of the specific implementation of the DLB algorithms and can hence be determined in simulations of the DLB process.

We consider two different load mobility models: (i) full mobility, where all subdomains are allowed to freely move within each matching, and (ii) partial mobility, where some subdomains cannot move arbitrarily from one PE to another. While the former is closer to the theoretical analysis presented above, the latter models more realistically a practical application. In scientific numerical simulations, we require the DLB protocol to not change the IPC graph, as otherwise global communication would become necessary and a new edge coloring would have to be determined. This effectively restricts the mobility of certain subdomains in order to preserve the original PE neighborhood relations.

1.3.4.1 Full and Partial Load Mobility Models

The full load mobility model represents the ideal case considered in the theoretical analysis. In this free-flowing model of subdomains, `SortedGreedy` and `Gradient` stand out as viable DLB protocols within the considered framework. `Greedy` transfers as many loads as `SortedGreedy`, but produces worse final load balances.

To assess a more realistic situation, we restrict 25% of the local loads to remain on their previous PE and allow the DLB protocols try to balance the loads between two PEs of a matching using only the movable loads. Again, `SortedGreedy` outperforms the other two protocols in terms of the

final load balance (Fig. 1.2). Compared to the full load mobility model, the ratio of the total number of transferred loads between `SortedGreedy` and `Gradient` drops to 1.7, which is expected since fewer loads are eligible to move in the partial load mobility model. In the case of partially mobile loads, `SortedGreedy` requires slightly less load movements than `Greedy`. Again, the relative performance of `SortedGreedy` over `Gradient` increases when more subdomains are available on the PEs (Fig. 1.3). The results of the full mobility model are omitted since their outcome is very similar to those of the partial mobility model.

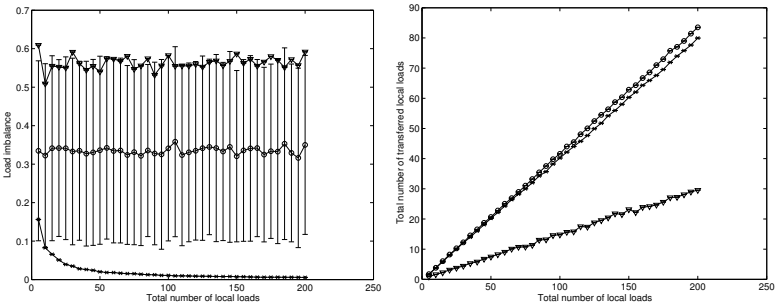
Taken together, the simulation results show that `Gradient` causes a smaller communication overhead, but produces less good a load balance than `SortedGreedy`. For a single matching, the relative performance of `SortedGreedy` is always better than that of `Gradient`, and it is growing with increasing granularity of the domain decomposition.

1.4 DLB Algorithms in Parallel Numerical Simulations

Based on the results from the previous section, we do not consider `Greedy` any further. We empirically compare `SortedGreedy` with `Gradient` in simulations of realistic IPC graphs of parallel numerical simulations. The sequence of matchings is given by an approximate minimum edge-coloring of the IPC graph, computed using Brélaz's algorithm [95]. In many practical applications, such an edge coloring is already available from the communication schedule with which the PEs exchange boundary information [67], and it can directly be re-used at no additional cost. Pseudocode for the overall parallel distributed DLB protocol is given in Algorithm 4.

1.4.1 Numerical Simulation Types

In practical numerical simulations, the computational elements (mesh nodes and/or particles) move as governed by the simulated dynamics. Here, we exemplarily consider two prototypical situations in 2D: A linear flow where the computational elements are advected from one side of the computa-



(a) (b)

Figure 1.2: Simulation results for DLB of indivisible, real-valued loads with 75% mobility on a matched edge in the BCM. Three algorithms are compared by (a) their resulting final discrepancy (i.e., maximum load difference) and (b) the required number of load movements to reach that final load balance: `SortedGreedy` ($\times$) produces the highest-quality load balancing, outperforming `Greedy` ($\circ$) by a factor of up to 80 and `Gradient` ($\triangledown$) by a factor of up to 140. `SortedGreedy` moves about 1.7-fold more loads than `Gradient`. The results for full load mobility are omitted as they produce very similar plots.

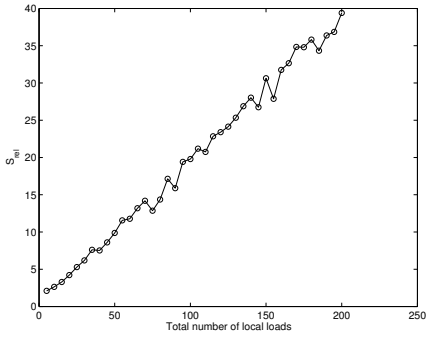


Figure 1.3: Relative performance S_{rel} of `SortedGreedy` over `Gradient` with 75% load mobility.

Algorithm 4 Distributed DLB algorithm

1: **procedure** DLB(L,$E_{colored}$,algorithm,s) ▷
Given are the global load vector `L`, the list of edges to visit $E_{colored}$, an *algorithm* to balance the loads `L`(u) and `L`(v) on a selected edge, and the number of iterations of `DLB`: s
2: **for** $i = 1 \rightarrow s$ **do**
3: **for** $j = 1 \rightarrow length(E_{colored}$ **do**
4: $\{u, v\} \leftarrow getVertices(E_{colored}[j])$ ▷ Find which vertices are connected by that edge
5: **if** algorithm == `SortedGreedy` **then**
6: [`L`(u), `L`(v)] ← `SortedGreedy`(u,v,`L`(u),`L`(v))
7: **else if** algorithm == `Greedy` **then**
8: [`L`(u), `L`(v)] ← `Greedy`(u,v,`L`(u),`L`(v))
9: **else if** algorithm == `HybridBalancer` **then**
10: **if** i==1 **then**
11: [`L`(u), `L`(v)] ← `SortedGreedy`(u,v,`L`(u),`L`(v))
12: **else**
13: [`L`(u), `L`(v)] ← `Greedy`(u,v,`L`(u),`L`(v))
14: **end if**
15: **end if**
16: **end for**
17: **end for**
18: **end procedure**

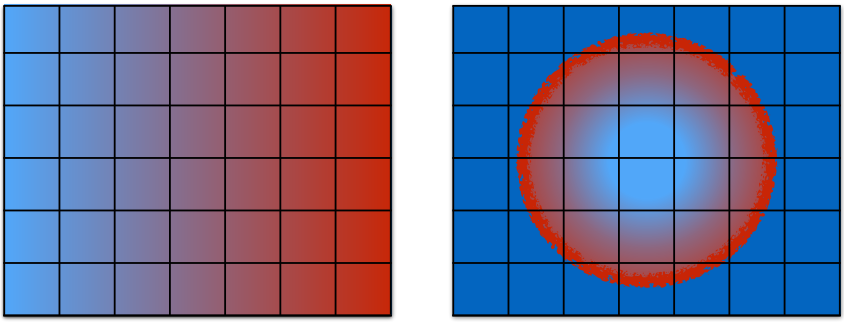


Figure 1.4: Load imbalance caused by a linear flow (left) and by a shock wave propagating from the center of the domain (right). The rectangles represent the subdomains, and the color shows the computational cost density (red: high, blue: low). The situation at an arbitrary time point of the simulation is depicted.

tional domain to the other, and the propagation of a shock wave originating from the center of the computational domain. In the latter case, the simulation mesh is adaptively refined and coarsened to track the evolution of the high gradient at the shock front. Both situations mimic realistic load imbalance evolutions in scientific simulation using particles and/or meshes to discretize the governing equations. A pictorial representation of the cost density for both simulation types is shown in Fig. 1.4.

1.4.2 IPC Topologies

We consider three different IPC topologies that are typical for numerical simulations (Fig. 1.5). The four-neighbor topology is representative for mesh-based simulations where stencils are to be evaluated at all mesh nodes and boundary layers (ghost layers) need to be communicated between neighboring PEs across edges. The eight-neighbor topology also includes the corners of the ghost layers, as is typical in particle-based simulations where each particle interacts with all other particles within a certain radius. The k-neighbor topology (here, $2 \leq k \leq 8$) represents the case of unstructured-mesh simulations, or of domain decompositions

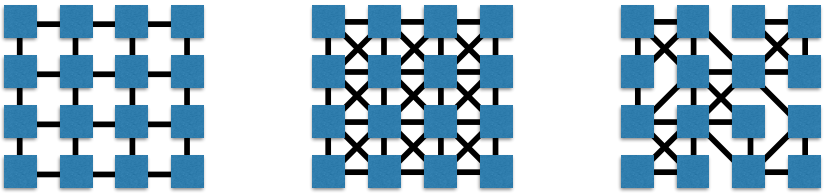


Figure 1.5: Three representative IPC topologies of numerical simulations: four-neighbor topology (left), eight-neighbor topology (middle), and k-neighbors topology (right) with $2 \leq k \leq 8$.

with adaptively varying subdomain sizes, where communication layers are needed on some edges, but not on all. Notice that these are the IPC topologies of the numerical simulation application, and not the network interconnect topologies of the computer the simulation is running on.

1.4.3 Simulation Setup

Similar to the local load balancing simulations of Sec. 1.3, each load is given a uniformly random value in the interval $(0, 1]$. In order to ensure sufficient granularity for DLB, practical domain decompositions produce significantly more subdomains than there are PEs in the computer [5, 67]. Each PE is initially assigned the same number of subdomains (e.g., 10 or 30) by SLB.

In order to preserve the initial IPC graph, and avoid global communication and re-coloring, a subdomain is only allowed to move to another PE if that move does not change the initial IPC graph, i.e., does not alter the neighborhood relationships between PEs. This inherently prevents the DLB algorithm from reaching optimal load balance. A schematic example of eligible and forbidden subdomain moves is shown Fig. 1.6.

We explore a variety of different DLB situations by varying the IPC topology (i.e., four-neighbor, eight-neighbor, or k-neighbor), the initial number of loads per PE (i.e., 10 or 30), and the changing sizes of the PE network (e.g., 4'096, 65'536, and 1'048'576). This allows us to compare the DLB al-

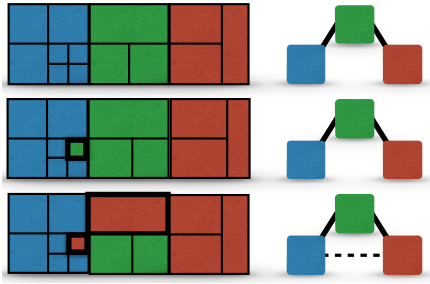


Figure 1.6: Illustration of allowed subdomain moves. Three PEs in blue, green, and red comprise an IPC as shown in the upper row. The middle row shows an allowed subdomain move from the blue PE to the green PE. However, both black-framed red subdomains (bottom row) are forbidden to move to the red PE, as that would create a new edge in the IPC graph.

gorithms at different network scales. Each simulation is repeated 50 times for different realizations of the random initial load assignment.

1.5 Simulation Results

The results of the simulations are presented in terms of S_{rel}. The results for the linear flow simulation are shown in Fig. 1.7, those for the shock wave propagation in Fig. 1.8. We average all cases (i.e., number of subdomains per PE, different IPCs, etc) in one figure in order to show an average DLB situation for that specific network size.

In all simulations, `SortedGreedy` produces better load balance than `Gradient`. The initial discrepancy is reduced up to three-fold by `SortedGreedy` and up to 50% by `Gradient`. `HybridBalancer` uses `SortedGreedy` for the first DLB round and `Gradient` for all subsequent rounds. It achieves similar final load balance as `SortedGreedy`, but at a much reduced communication overhead. All algorithms achieve their biggest improvements in S during the first few DLB rounds and quickly reach steady state. In practice, 2 to 3 rounds of a local BCM scheme are hence sufficient.

The relative performances of `Gradient` and `HybridBalancer` over `SortedGreedy`

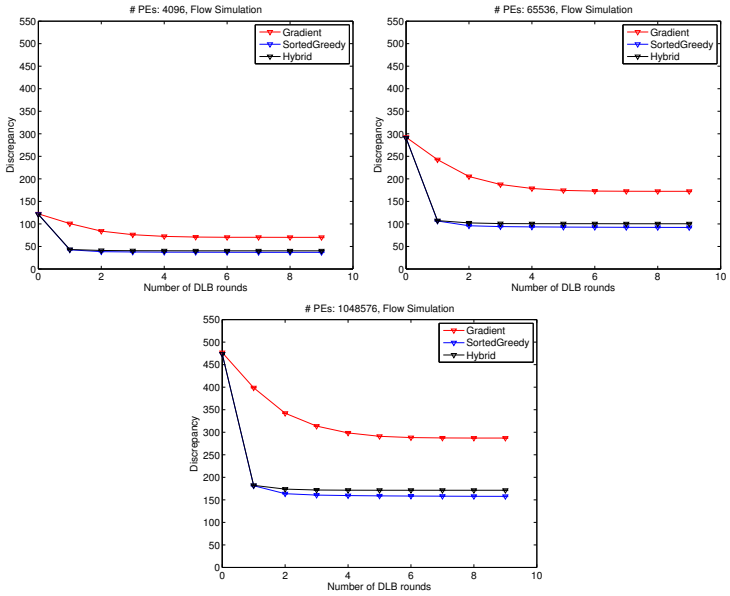


Figure 1.7: Discrepancy vs. DLB round when `SortedGreedy`, `Gradient`, and `HybridBalancer` are applied in the linear flow simulation on different numbers of PEs. `SortedGreedy` and `HybridBalancer` provide the largest decrease in discrepancy in the first DLB round. After a few rounds, steady state is reached.

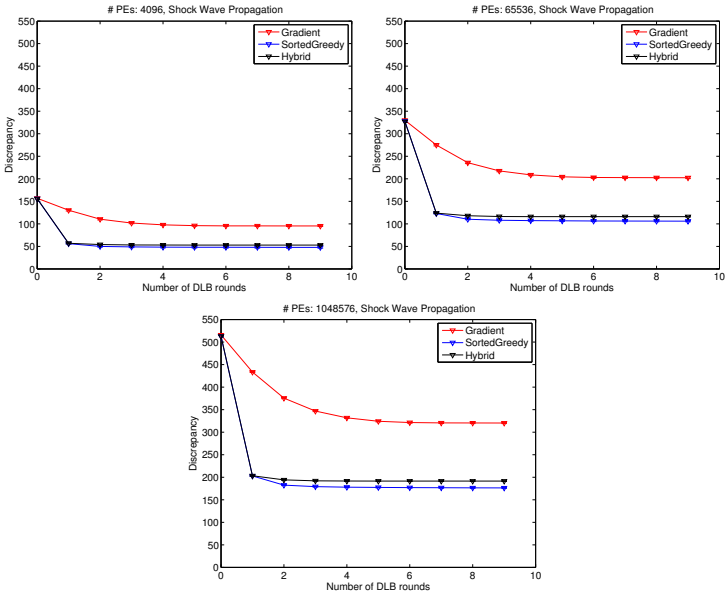


Figure 1.8: Discrepancy vs. DLB round when `SortedGreedy`, `Gradient`, and `HybridBalancer` are applied in the shock wave simulation on different numbers of PEs. An increasing gap between `Gradient` and the other methods can be seen as the number of PEs grows.

are presented in Figs. 1.9 and 1.10 for the flow and shock-wave simulations, respectively. In all cases, `Gradient` outperforms `SortedGreedy` by a factor of at least three. This is in contrast to what we saw for a single matching in the previous section, where `SortedGreedy` always outperformed `Gradient`. In a complete simulation with a realistic IPC graph, the reduced communication overhead of `Gradient` in every matching sums up to amortize the final loss in load balance compared with `SortedGreedy`. `HybridBalancer` gives favorable results for large network sizes and connectivities. Only for the small network (4'096 PEs) with four neighbors per PE (i.e., $k = 4$), `SortedGreedy` provides better performance than `HybridBalancer`. The graphs for 64k and 1M PEs are virtually indistinguishable, indicating that the performance differences reach an asymptotic plateau for large machines.

1.6 Conclusions

Efficient and scalable distributed DLB schemes are crucial for scientific numerical simulation on petascale machines and beyond. We analyzed the typical case of a numerical simulation based on a domain decomposition. The computational cost of each subdomain was defined as the wall-clock execution time required to perform all calculations for one simulation time step in that subdomain. This can be easily measured in a practical simulation by timing the main iteration loop. In order for the DLB scheme to provide a significant benefit over re-initializing the simulation, we required that DLB does not trigger any global communication. This requires using a distributed DLB scheme and preserving the IPC graph of the application. While this restricts the mobility of some loads, and disallows further dividing existing subdomains, it allows us to re-use the existing edge coloring of the IPC graph. In this model, the loads thus are indivisible and real-valued.

We provided a theoretical analysis of distributed DLB protocols for indivisible, real-valued loads. Closely following the analysis presented in Ref. [93], we showed that the bounds for the load imbalance are the same as for the discrete load model.

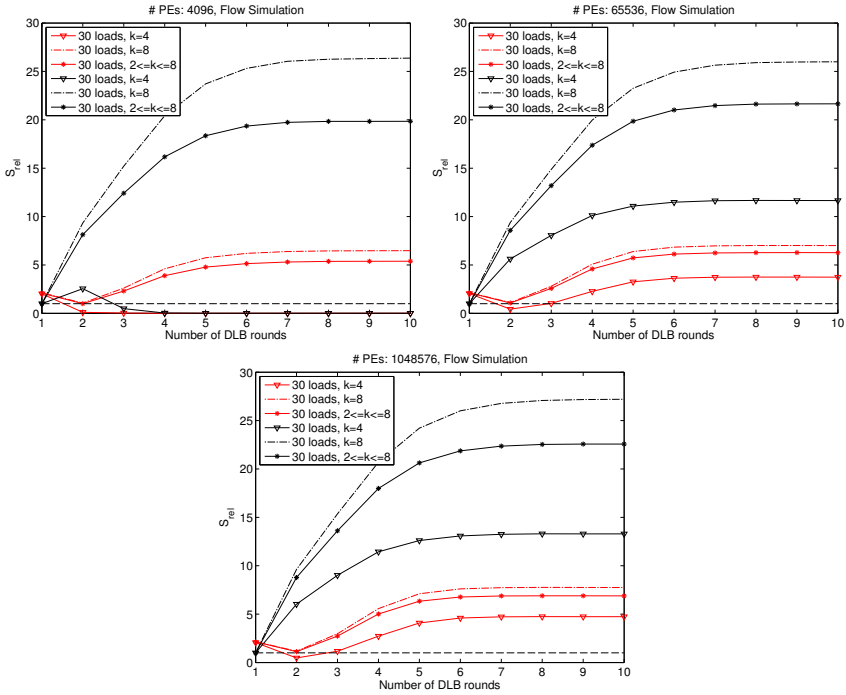


Figure 1.9: Relative performance of `Gradient` (red) and `HybridBalancer` (black) over `SortedGreedy` (dashed line at $S_{\text{rel}} = 1$) in the linear flow simulation on different numbers of PEs and for different IPC topologies and numbers of subdomains per PE.

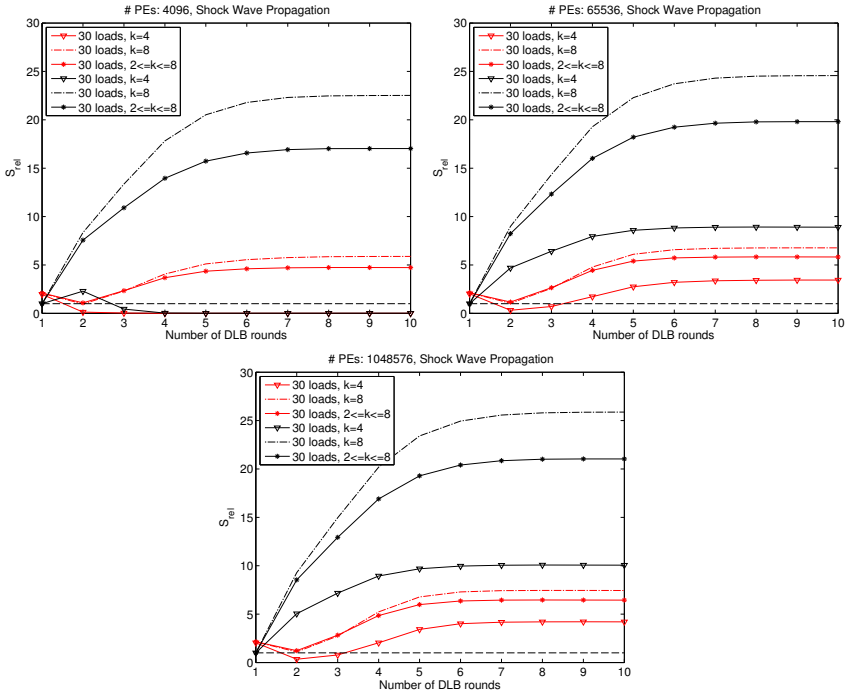


Figure 1.10: Relative performance of `Gradient` (red) and `HybridBalancer` (black) over `SortedGreedy` (dashed line at $S_{\text{rel}} = 1$) in the shock-wave simulation on different numbers of PEs and for different IPC topologies and numbers of subdomains per PE.

DLB algorithms based on the balancing circuit model (BCM) are well suited to address system-wide load imbalance. In the BCM, each PE in the network is matched with a single nearest neighbor at a time and tries to minimize the local load imbalance with that neighbor. The order of matchings is usually given by an edge-coloring algorithm, which is executed during initialization of the simulation when setting up the IPC graph. We compared the three protocols `Greedy`, `SortedGreedy`, and `Gradient` using an analogy with the offline balls-into-bins problem. The theoretical analysis showed that `Greedy` cannot be trusted to locally balance loads. `SortedGreedy` overcomes this problem, but potentially moves a large number of loads between neighboring PEs. This communication overhead is relaxed in the heuristic `Gradient`. To assess the performances of DLB methods, we introduced the metric S. Simulations have shown that in a single matching, `SortedGreedy` consistently outperformed `Gradient`.

In realistic simulations of whole IPC graphs, however, `Gradient` almost consistently outperformed `SortedGreedy`. We considered three prototypical IPC topologies as found in domain-decomposition numerical simulations of two kinds (linear flow and circular shock-wave propagation). We analyzed simulations on more than one million PEs. After only a few (2 to 3) iterations of the DLB protocol, `Gradient` outperformed `SortedGreedy` up to seven-fold. Based on the observation that `SortedGreedy` achieved the largest decrease in load imbalance during the first DLB round, we designed a hybrid method called `HybridBalancer`, which uses `SortedGreedy` in the first round and `Gradient` in all subsequent rounds. `HybridBalancer` has been shown in our simulations to achieve similar final load balance as `SortedGreedy`, but at a much lower communication overhead. Thus, `HybridBalancer` showed the best figure of merit S, especially for large networks and high IPC graph connectivities. Only for 4-connected small ($<$4096 PEs) networks, `Gradient` outperformed `HybridBalancer`. In large machines, `HybridBalancer` is a good DLB candidate that can reduce initial load imbalance up to three-fold in few DLB rounds requiring only local communication among neighboring PEs.

We neglected the specifics of the computer system and the parallel application implementation, in favor of generalizable results. While this renders our results somewhat generic, it also disregards several optimizations and implementation "tricks" one would leverage in a concrete implemen-

tation. We also did not distinguish between communication latency and bandwidth, but simply assigned a unit communication cost to each load transfer. In a practical implementation, one would pack several loads into a single message in order to reduce latency.

CHAPTER
TWO

Bayesian Filtering

2.1 Introduction

Before understanding dynamic load balancing problems in parallel particle filtering algorithms, we would like to give an introduction to Bayes' rule and Bayesian filtering to constitute the theoretical background of particle filtering algorithms.

In many real-world applications, we are interested in learning about a quantity (i.e., *state*) $\mathbf{x}_t$ via a *measurement* $\mathbf{y}_t$ at time t. This effort, as depicted in Fig. 2.1, is complicated by the fact that $\mathbf{x}_t$ is usually hidden in the obtained $\mathbf{y}_t$ and in most cases, extracting $\mathbf{x}_t$ is not trivial. The quest of discovering $\mathbf{x}_t$ is called *estimation or inversion problem*[1]. Unfortunately, there are two major factors that hamper the estimation problem. The first factor is concerned with real-world measurements, which are rarely clean. They usually contain random noises and are subject to observational errors due to technical limitations of the employed sensors. The second factor stems from a lack of in-depth understanding of the underlying phenomenon, which would have helped us otherwise to interpret measurements better and find out more about the unknown state. The desire to tackle these difficulties has led to the development of modern statisti-

[1] Both terms are used interchangeably in the text.

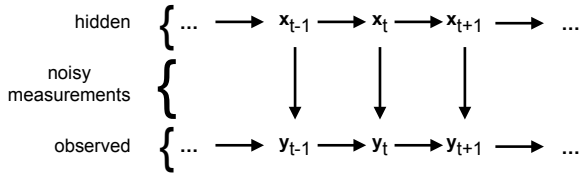


Figure 2.1: In optimal Bayesian filtering problems, the actual quantities of interest $\mathbf{x}$ are hidden in $\mathbf{y}$ due to noisy measurements, which are common in practical applications.

cal signal processing techniques, i.e., *Bayesian filters* that are rooted in a probabilistic framework, namely, Bayesian theory [110, 111].

Bayesian theory is a versatile probability law that governs processes of logical inference. It is based on Bayes' rule [112]. Following its introduction in 1763, French mathematician Pierre-Simon Laplace built upon Bayes' rule and established the foundations of modern Bayesian statistics [113–115], which is now considered an important branch of statistical inference [116, 117].

Mathematically, Bayes' rule describes the relationships between the probability density functions[2] (pdf) of $\mathbf{x}$ and $\mathbf{y}$, $\mathrm{p}(\mathbf{x})$ and $\mathrm{p}(\mathbf{y})$ and also, the conditional pdfs (pdf) $\mathrm{p}(\mathbf{x}|\mathbf{y})$ and $\mathrm{p}(\mathbf{y}|\mathbf{x})$ between them. Bayes' rule is formulated as follows:

$$\mathrm{p}(\mathbf{x}|\mathbf{y}) = \frac{\mathrm{p}(\mathbf{y}|\mathbf{x})\,\mathrm{p}(\mathbf{x})}{\mathrm{p}(\mathbf{y})}\,. \tag{2.1}$$

The goal in Bayesian filtering [118–122] is to extract information about a quantity of interest at time t by using all data available up until t using Bayesian theory, which is incorporated by Bayesian filtering into this inversion problem. This makes the problem in hand a *statistical* inversion problem where Bayesian filters try to compute *posterior distribution* of $\mathbf{x}$ given the measurements $\mathbf{y}$. This is enabled by modeling the system using a probabilistic *state-space representation* (see Appendix C for a detailed

[2]The conditional pdf $\mathrm{p}(\mathbf{x}|\mathbf{y})$ is read as how likely $\mathbf{x}$ is to happen given that $\mathbf{y}$ has occurred.

discussion).

Reformulating a statistical inversion problem in the Bayesian framework gives rise to *recursive Bayesian estimation*, which is based on two assumptions [120]:

- Each state $\mathbf{x}_{i\in[0,\dots,t]}$ follows a first-order Markov process[3]

$$\mathrm{p}(\mathbf{x}_t|\mathbf{x}_{0:t-1}) = \mathrm{p}(\mathbf{x}_t|\mathbf{x}_{t-1})\,, \tag{2.2}$$

  where $\mathbf{x}_{0:t-1} := \{\mathbf{x}_0, \dots, \mathbf{x}_{t-1}\}$.

- The measurements $\mathcal{Y}_t := \mathbf{y}_{0:t}$ are conditionally independent of the given states $\mathbf{x}_{0:t}$ and $\mathrm{p}(\mathbf{x}_t|\mathbf{y}_{0:t})$ denotes the conditional pdf of $\mathbf{x}_t$.

Following these assumptions and using Bayes' rule with the given observations, the recursive Bayes' theorem for the posterior distribution $\mathrm{p}(\mathbf{x}_t|\mathcal{Y}_t)$ of the statistical inversion problem can be derived as

$$\begin{aligned}
\mathrm{p}(\mathbf{x}_t|\mathcal{Y}_t) &= \frac{\mathrm{p}(\mathcal{Y}_t|\mathbf{x}_t)\ \mathrm{p}(\mathbf{x}_t)}{\mathrm{p}(\mathcal{Y}_t)} \\
&= \frac{\mathrm{p}(\mathbf{y}_t, \mathcal{Y}_{t-1}|\mathbf{x}_t)\ \mathrm{p}(\mathbf{x}_t)}{\mathrm{p}(\mathbf{y}_t, \mathcal{Y}_{t-1})} \\
&= \frac{\mathrm{p}(\mathbf{y}_t|\mathcal{Y}_{t-1}, \mathbf{x}_t)\ \mathrm{p}(\mathcal{Y}_{t-1}|\mathbf{x}_t)\ \mathrm{p}(\mathbf{x}_t)}{\mathrm{p}(\mathbf{y}_t|\mathcal{Y}_{t-1})\ \mathrm{p}(\mathcal{Y}_{t-1})} \\
&= \frac{\mathrm{p}(\mathbf{y}_t|\mathcal{Y}_{t-1}, \mathbf{x}_t)\ \mathrm{p}(\mathbf{x}_t|\mathcal{Y}_{t-1})\ \mathrm{p}(\mathcal{Y}_{t-1})\ \mathrm{p}(\mathbf{x}_t)}{\mathrm{p}(\mathbf{y}_t|\mathcal{Y}_{t-1})\ \mathrm{p}(\mathcal{Y}_{t-1})\ \mathrm{p}(\mathbf{x}_t)} \\
&= \frac{\mathrm{p}(\mathbf{y}_t|\mathbf{x}_t)\ \mathrm{p}(\mathbf{x}_t|\mathcal{Y}_{t-1})}{\mathrm{p}(\mathbf{y}_t|\mathcal{Y}_{t-1})}\,.
\end{aligned} \tag{2.3}$$

Recursive, also called *sequential*, estimation using Bayes' theorem is the keystone of Bayesian filters. It facilitates numerical solutions of online estimation problem where observations arrive sequentially in an online manner. As shown in Eq. 2.3, the posterior $\mathrm{p}(\mathbf{x}_t|\mathcal{Y}_t)$ consists of three terms: the *prior*, the *likelihood*, and the *evidence/normalizer*:

[3] A Markov process satisfies the Markov property that the conditional pdfs of states of the process depend solely upon the immediately preceding state and thus, that Markov process is a first-order Markov chain [123, 124].

- **Prior (dynamics model):** The prior distribution $\mathrm{p}(\mathbf{x}_t|\mathcal{Y}_{t-1})$ combines all knowledge of the model:

$$\mathrm{p}(\mathbf{x}_t|\mathcal{Y}_{t-1}) = \int \mathrm{p}(\mathbf{x}_t|\mathbf{x}_{t-1})\,\mathrm{p}(\mathbf{x}_{t-1}|\mathcal{Y}_{t-1})\,\mathrm{d}\mathbf{x}_{t-1}\,, \tag{2.4}$$

  where $\mathrm{p}(\mathbf{x}_t|\mathbf{x}_{t-1})$ is the probability distribution of the transition from state $\mathbf{x}_{t-1}$ to $\mathbf{x}_t$.

- **Likelihood (measurement/observation model):** The likelihood $\mathrm{p}(\mathbf{y}_t|\mathbf{x}_t)$ tells how probable it is to observe $\mathbf{y}_t$ given $\mathbf{x}_t$. In other words, how does $\mathbf{y}_t$ depend upon $\mathbf{x}_t$. In real-world applications, the noise model is incorporated into the likelihood.

- **Evidence/Normalizer:** The evidence, or the normalizer, is the total probability of the measurement, which is written in integral form as:

$$\mathrm{p}(\mathbf{y}_t|\mathcal{Y}_{t-1}) = \int \mathrm{p}(\mathbf{y}_t|\mathbf{x}_t)\,\mathrm{p}(\mathbf{x}_t|\mathcal{Y}_{t-1})\,\mathrm{d}\mathbf{x}_t\,. \tag{2.5}$$

  It is constant because it does not depend on $\mathbf{x}$ and thus, it can be substituted for a coefficient, which is often done in practical applications. In practice, it is almost never possible to compute this integral because the integration is over all possible $\mathbf{x}$ and the space of $\mathbf{x}$ is generally very large and cannot be explored.

Recursive Bayesian estimation using Eq. 2.3 aims to find the *optimal* estimate given all prior knowledge and observations. Here, the estimation problem involves calculating the integrals occurring in the prior and likelihood models. Since only in a few cases the estimation problem can be solved analytically, we need to clarify what it is meant by "optimal" filtering.

In Bayesian filtering, there are two types of optimality: The first one is concerned with whether the integrals can be solved analytically. If closed-form solutions exist, then a Bayesian filter can solve the problem in a *mathematically optimal* way. The second optimality is the *statistical optimality*. Since closed-form solutions to most of these models are computationally intractable, the integrals involved in these models require numerical integration approximations, which are not *mathematically optimal* based

on the definition given above. However, depending on the chosen criterion (e.g., Minimum mean-squared error (MMSE), maximum a posteriori (MAP), etc.) to describe the quality of the solution obtained by a Bayesian filter, we can speak of *statistically optimal* [125] solutions.

Arguably, the most famous *mathematically optimal* Bayesian filter is the Kalman Filter (KF) [60, 126], which is obtained as the analytical solution to linear Gaussian estimation problems. The KF provides an optimal solution to estimation problems where linear dynamics and linear observation models are perturbed by Gaussian noise. In practice, the KF and its statistically-optimal extension to nonlinear problems (i.e., extended KF [127]) are much celebrated and have been applied in various applications such as tracking objects (e.g., hands [128], faces [129], missiles [130]), analysis of time-series in econometrics [131, 132], navigation [58, 133] and numerous computer vision applications such as data fusion [134, 135] and depth estimation [136].

Except for few cases (e.g., linear Gaussian and conjugate approaches for some nonlinear non-Gaussian systems), finding an analytical solution to the integrals appearing in the dynamics and observation models is intractable. Solving the estimation problem then requires numerical integration, for which many techniques have been introduced. Some approximation methods are illustrated in Fig. 2.2 and a shortlist of examples includes [120]:

- Gaussian/Laplace approximation methods [127, 137, 138]
- Iterative quadrature [139, 140]
- Multigrid method [141–143] and point-mass approximation [141, 144, 145]
- Moment approximation [146, 147]
- Histogram approximation [148]
- Riemann sum approximation [149]
- Gaussian mixture approximation [150, 151]

- Deterministic sampling approximation (e.g., unscented Kalman Filter [61, 152, 153])
- Quasi-Monte Carlo methods [154–156]
- Monte Carlo sampling approximation [157, 158]

In this thesis, we emphasize the latter, namely, the class of Monte Carlo sampling methods, of which particle filters are a part. From a computer science point of view, these methods use particles as the main data structure to solve the estimation problem. In the following, we give the theoretical background of Monte Carlo sampling approximations with a focus on *importance sampling* and *sequential importance sampling*. These lead us to the *sequential importance resampling* algorithm, which is the core of particle filters.

2.2 Monte Carlo Methods

Many physical phenomena and complex systems cannot be understood or analyzed thoroughly by pure mathematical techniques. Shortly after their inception in statistical physics [32, 159], Monte Carlo methods have become one of the most celebrated approximation methods that offer powerful analysis of complicated problems. Roughly speaking, Monte Carlo methods have two main categories: (i) *Monte Carlo sampling* is concerned with estimation of a mathematical problem by statistical sampling; (ii) *Monte Carlo optimization* [160, 161] deals with using Monte Carlo techniques for optimization problems over non-/convex or non-/differentiable functions. In the scope of this thesis, we solely focus on Monte Carlo sampling methods.

In spite of the variety in real-world applications, *particles* provide a common data structure for all Monte Carlo methods. Particles have positions and carry some properties. The evolution of particle properties depends on the application, which is the main source of the diversity in Monte Carlo methods. Next, we explain the fundamental idea behind Monte Carlo sampling with an example of integral evaluation and then extend it to Bayesian inference problems.

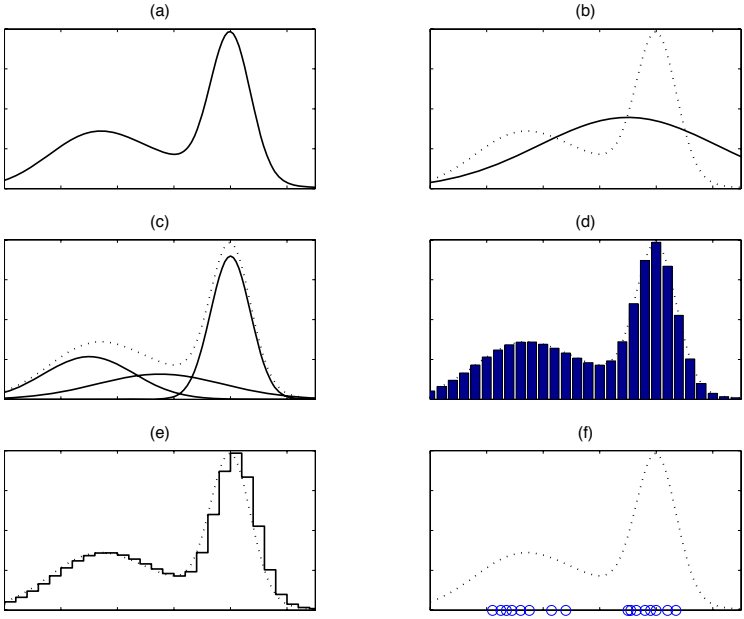


Figure 2.2: Illustration of non-Gaussian integral approximation methods as shown in [120]: (a) An example of a "true" posterior distribution; (b) Gaussian approximation; (c) Gaussian mixture approximation; (d) histogram approximation; (e) Riemannian sum approximation; (f) Monte Carlo sampling approximation.

2.2.1 Integral Approximation with Monte Carlo

To show an example of how Monte Carlo approximation works, we solve a well-known statistical problem of estimating a mean, $\mathrm{E}[f(X)]$, where f is an integrable function in a given d-dimensional space $\Omega \subset \mathbb{R}^d$ and X is a random variable with support Ω.

The mean, i.e., expectation of $f(X)$, $\mathrm{E}[f(X)]$, can be written as

$$\mathrm{E}[f(X)] = \int_{x \in \Omega} f(x) p(x) \mathrm{d}x \,, \tag{2.6}$$

where $p(x)$ is the probability density function of $\mathbf{x}$. Monte Carlo approximation of the mean starts with the selection of N i.i.d.[4] samples $\{x^1, \ldots, x^N\}$ from X. Then, we compute the mean of $f(\cdot)$ using these samples, we get an unbiased Monte Carlo estimate of $\mathrm{E}[f(X)]$ as follows

$$\mathrm{E}[f(X)] \approx \hat{f}_N(x) = \frac{1}{N} \sum_{i=1}^{N} f(x^{(i)}) \,. \tag{2.7}$$

Assuming that $\mathrm{E}[f(X)]$ exists, we can use the Weak Law of Large Numbers (i.e., Khintchine's law [162]) to show that

$$\lim_{N \to \infty} \Pr\left(\left| \hat{f}_N(x) - \mathrm{E}[f(X)] \right| \geq \epsilon \right) = 0 \,. \tag{2.8}$$

This implies that as N gets larger, the Monte Carlo approximation converges to true $\mathrm{E}[f(X)]$. The convergence rate is guaranteed by the Central Limit Theorem as

$$\sqrt{N}(\hat{f}_N(x) - \mathrm{E}[f(X)]) \sim \mathcal{N}(0, \sigma^2) \,, \tag{2.9}$$

where σ^2 is the variance of $f(x)$. The theoretical convergence rate is therefore $\mathrm{O}\big(N^{-0.5}\big)$ irrespective of the dimensionality of the state-space. Since the convergence of Monte Carlo methods do not depend on the dimensionality of the state-space, they have a practical advantage in many real-world problems over other numerical schemes. Despite their success, there are

[4] Independent and identically distributed.

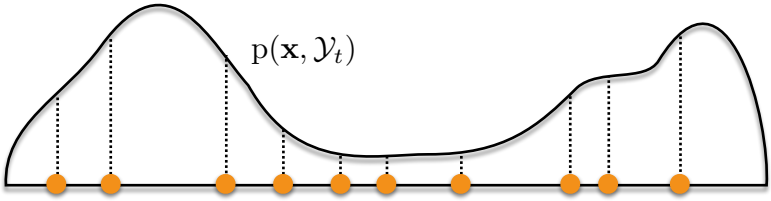


Figure 2.3: An illustration of *direct* Monte Carlo sampling is shown. We are interested in approximating the posterior $\mathrm{p}(\mathbf{x}|\mathcal{Y}_t)$. A direct Monte Carlo method tries to sample directly from the posterior, which is only possible in rare cases. Further, since particles are sampled randomly, sufficient approximation of the posterior may require an excessive amount of particles. Theoretically, the approximation error is halved when the total number of particles is quadrupled.

also reports [163, 164] where the "curse of dimensionality" has been seen in practical applications of Monte Carlo methods.

2.2.2 Monte Carlo Approximation in Bayesian Filters

In Bayesian filtering, Monte Carlo methods, also called *direct Monte Carlo methods* [122], draw N samples directly from the posterior distribution as

$$x^{(i)} \sim \mathrm{p}(\mathbf{x}|\mathcal{Y}_t), \quad i \in \{1, \ldots, N\}, \tag{2.10}$$

and following the Eq. 2.7, $\mathrm{p}(\mathbf{x}|\mathcal{Y}_t)$ can be approximated by

$$\mathrm{p}(\mathbf{x}|\mathcal{Y}_t) \approx \frac{1}{N} \sum_{i=1}^{N} \delta(\mathbf{x} - x^{(i)}), \tag{2.11}$$

where $\delta(\cdot)$ is the Dirac delta function. An illustration of such a sampling is given in Fig. 2.3. Again, the theoretical convergence of a Monte Carlo method is independent of the dimensionality of the state-space and has the rate of $\mathrm{O}\big(N^{-0.5}\big)$, which means that to halve the approximation error one needs to quadruple the sample size.

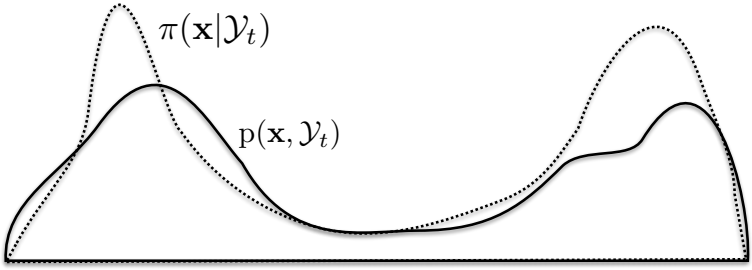


Figure 2.4: In practical Bayesian filtering applications, sampling from the posterior $\mathrm{p}(\mathbf{x}|\mathcal{Y}_t)$ is challenging. Thus, an importance distribution $\pi(\mathbf{x}|\mathcal{Y}_t)$ is usually proposed as an approximation to the posterior.

2.3 Importance Sampling

Importance sampling (IS) [165–167] has been introduced to address two main problems of *direct Monte Carlo methods*. The first one is concerned with reducing the variance of the estimator [168]. The second problem is due to the difficulty of sampling directly from the posterior distribution $\mathrm{p}(\mathbf{x}|\mathcal{Y}_t)$, which is not possible in many real-world problems.

Instead of directly sampling from the (potentially unknown) posterior, IS samples from an *importance distribution* π (i.e., proposal distribution), which is an approximation to the actual posterior distribution (Fig. 2.4). By substituting the posterior distribution $\mathrm{p}(\mathbf{x}, \mathcal{Y}_t)$ for $\mathrm{p}(x)$ in Eq. 2.6 and incorporating the equation in a Bayesian estimation problem, we show that the main idea behind importance sampling is the decomposition of the expectation over $\mathrm{p}(\mathbf{x}, \mathcal{Y}_t)$:

$$\int f(\mathbf{x})\mathrm{p}(\mathbf{x}|\mathcal{Y}_t)\,\mathrm{d}\mathbf{x} = \int \left[f(\mathbf{x}) \frac{\mathrm{p}(\mathbf{x}|\mathcal{Y}_t)}{\pi(\mathbf{x}, \mathcal{Y}_t)} \right] \pi(\mathbf{x}, \mathcal{Y}_t)\,\mathrm{d}\mathbf{x}\,, \tag{2.12}$$

where the importance distribution $\pi(\mathbf{x}, \mathcal{Y}_t)$ needs to have support that is at least as large as the support of $\mathrm{p}(\mathbf{x}, \mathcal{Y}_t)$. The bracketed term in the analytical expression is the expectation over the importance distribution

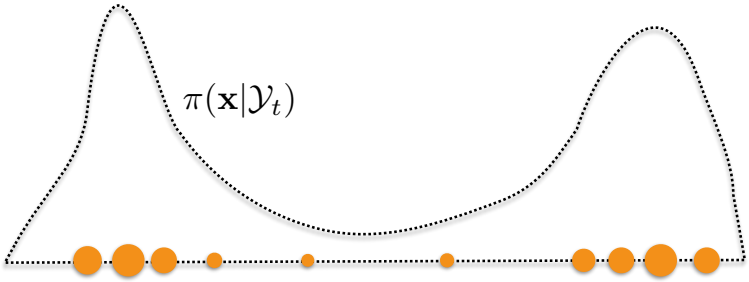


Figure 2.5: Particle weights are drawn from an importance distribution $\pi(\mathbf{x}|\mathcal{Y}_t)$, which is an approximation to the actual posterior pdf. Bigger orange circles stand for "heavier" particles. The choice of $\pi(\mathbf{x}|\mathcal{Y}_t)$ is crucial for the overall estimation success of IS, since bad approximation of the posterior would yield inaccurate estimation.

$\pi(\mathbf{x}, \mathcal{Y}_t)$, which can be approximated by N Monte Carlo samples

$$x^{(i)} \sim \pi(\mathbf{x}|\mathcal{Y}_t), \quad i \in \{1, \ldots, N\}, \tag{2.13}$$

as

$$\mathrm{E}[f(\mathbf{x})|\mathcal{Y}_t] \approx \sum_{i=1}^{N} \tilde{w}^{(i)} f(x^{(i)}), \tag{2.14}$$

where the *importance weights* $\tilde{w}$ are defined as

$$\tilde{w}^{(i)} = \frac{1}{N} \frac{\mathrm{p}(x^{(i)}|\mathcal{Y}_t)}{\pi(x^{(i)}|\mathcal{Y}_t)}. \tag{2.15}$$

An exemplary IS is shown in Fig. 2.5. A disadvantage of IS is that the weights of samples cannot be known exactly in the beginning, since they are not drawn from the actual posterior distribution but from the importance distribution. Some corrections to the weights are usually required. Another disadvantage of IS stems from the mandatory evaluation of $\mathrm{p}(x^{(i)}|\mathcal{Y}_t)$ such that it can be applied directly in Eq. 2.15 to compute the importance weights.

To compute $\mathrm{p}(x^{(i)}|\mathcal{Y}_t)$, we first formulate its Monte Carlo approximation

in a Bayesian framework:

$$\mathrm{p}(x^{(i)}|\mathcal{Y}_t) = \frac{\mathrm{p}(\mathcal{Y}_t|x^{(i)})\mathrm{p}(x^{(i)})}{\int \mathrm{p}(\mathcal{Y}_t|\mathbf{x})\mathrm{p}(\mathbf{x})\,\mathrm{d}\mathbf{x}}\,. \tag{2.16}$$

As seen from the equation, the prior distribution $\mathrm{p}(x^{(i)})$ and the likelihood $\mathrm{p}(\mathcal{Y}_t|x^{(i)})$ can be directly evaluated but the normalizer in the denominator is intractable. To tackle this, one can also apply importance sampling as an approximation to the normalizer. The approximation of the expectation integral can be derived as [122]:

$$\begin{aligned}
\mathrm{E}[f(\mathbf{x})|\mathcal{Y}_t] &= \int f(\mathbf{x})\mathrm{p}(\mathbf{x}|\mathcal{Y}_t)\,\mathrm{d}\mathbf{x} \\
&= \frac{\int f(\mathbf{x})\mathrm{p}(\mathcal{Y}_t|\mathbf{x})\mathrm{p}(\mathbf{x})\,\mathrm{d}\mathbf{x}}{\int \mathrm{p}(\mathcal{Y}_t|\mathbf{x})\mathrm{p}(\mathbf{x})\,\mathrm{d}\mathbf{x}} \\
&= \frac{\int \left[\frac{\mathrm{p}(\mathcal{Y}_t|\mathbf{x})\mathrm{p}(\mathbf{x})}{\pi(\mathbf{x}|\mathcal{Y}_t)} f(\mathbf{x})\right] \pi(\mathbf{x}|\mathcal{Y}_t)\,\mathrm{d}\mathbf{x}}{\int \left[\frac{\mathrm{p}(\mathcal{Y}_t|\mathbf{x})\mathrm{p}(\mathbf{x})}{\pi(\mathbf{x}|\mathcal{Y}_t)}\right] \pi(\mathbf{x}|\mathcal{Y}_t)\,\mathrm{d}\mathbf{x}} \\
&\approx \frac{\frac{1}{N}\sum_{i=1}^{N} \frac{\mathrm{p}(\mathcal{Y}_t|x^{(i)})\mathrm{p}(x^{(i)})}{\pi(x^{(i)}|\mathcal{Y}_t)} f(x^{(i)})}{\frac{1}{N}\sum_{j=1}^{N} \frac{\mathrm{p}(\mathcal{Y}_t|\theta^{(j)})\mathrm{p}(\theta^{(j)})}{\pi(\theta^{(j)}|\mathcal{Y}_t)}} \\
&= \sum_{i=1}^{N} w^{(i)} f(x^{(i)})\,,
\end{aligned} \tag{2.17}$$

where the importance weight w_i of particle i is defined as

$$w^{(i)} = \frac{\frac{\mathrm{p}(\mathcal{Y}_t|x^{(i)})\mathrm{p}(x^{(i)})}{\pi(x^{(i)}|\mathcal{Y}_t)}}{\frac{1}{N}\sum_{j=1}^{N} \frac{\mathrm{p}(\mathcal{Y}_t|\theta^{(j)})\mathrm{p}(\theta^{(j)})}{\pi(\theta^{(j)}|\mathcal{Y}_t)}}\,. \tag{2.18}$$

Formally, we can write the resulting posterior approximation using the Dirac delta function $\delta(\cdot)$ as

$$\mathrm{p}(\mathbf{x}|\mathcal{Y}_t) \approx \sum_{j=1}^{N} w^{(i)} \delta(\mathbf{x} - x^{(i)})\,. \tag{2.19}$$

Algorithm 5 Importance Sampling (IS)

procedure IS
  $S \leftarrow 0$
  for $i = 1 \rightarrow N$ **do**
    $x^{(i)} \sim \pi(\mathbf{x}|\mathcal{Y}_t)$ ▷ Draw a sample from importance distribution
    $\tilde{w}^{(i)} \leftarrow \frac{\mathrm{p}(\mathcal{Y}_t|x^{(i)})\mathrm{p}(x^{(i)})}{\pi(x^{(i)}|\mathcal{Y}_t)}$ ▷ Calculate its unnormalized weight
  end for
  for $i = 1 \rightarrow N$ **do** ▷ Normalize weights
    $w^{(i)} \leftarrow \frac{\tilde{w}^{(i)}}{\sum_{j=1}^{N} \tilde{w}^{(j)}}$
    $S \leftarrow S + w^{(i)} f(x^{(i)})$
  end for
  $\mathrm{E}[f(\mathbf{x})|\mathcal{Y}_t] \approx S$
end procedure

To increase the efficiency of the sampling scheme, several algorithms such as Gibbs sampling [169, 170] and Metropolis-Hastings sampling [13, 171, 172] are regularly adopted and commonly used in practice. To summarize the derivations explained above, we give a pseudo-code for IS in Algorithm 5.

2.4 Sequential Importance Sampling

While importance sampling is a good method for estimating the posterior distribution, the choice of the importance distribution plays an integral role in determining overall success of the estimator [173, 174]. In many problems with high-dimensional state-spaces, finding a good importance/proposal distribution is a challenging task. Sequential importance sampling (SIS) [8, 9] has been introduced as a remedy to such problems by constructing sequential importance samplers as more data become available. Inherently, SIS adopts the notion of time in its generic state-space model

$$\mathbf{x}_t \sim \mathrm{p}(\mathbf{x}_t|\mathbf{x}_{t-1})\,, \tag{2.20}$$
$$\mathbf{y}_t \sim \mathrm{p}(\mathbf{y}_t|\mathbf{x}_t)\,, \tag{2.21}$$

where $\mathbf{x}_t \in \mathbb{R}^n$ is the state at time t and $\mathbf{y}_t \in \mathbb{R}^m$ is the observation at t. SIS uses a set of weighted *particles* (i.e., samples) as the main data structure. Each particle carries a weight and a sample from the importance distribution

$$x_{0:t}^{(i)} \sim \pi(\mathcal{X}_t|\mathcal{Y}_t)\,, \tag{2.22}$$

where $\mathcal{Y}_t = \{\mathbf{y}_1, \ldots, \mathbf{y}_t\}$ and $\mathcal{X}_t = \{\mathbf{x}_0, \ldots, \mathbf{x}_t\}$. Note that $\mathcal{X}_t$ includes the state $\mathbf{x}_0$, which is available before the first measurement $\mathbf{y}_1$ is made. Similar to IS, the SIS estimation of the posterior distribution can be written as

$$\mathrm{p}(\mathbf{x}_t|\mathcal{Y}_t) \approx \sum_{j=1}^{N} w_t^{(i)} \delta(\mathbf{x}_t - x_t^{(i)})\,. \tag{2.23}$$

However, these weights are computed in a different way compared to IS importance weights. To formulate the SIS importance weights, we consider the full posterior distribution of states $\mathcal{X}_t$ given the observations $\mathcal{Y}_t$. Recursive Bayesian estimation again uses a recursive formulation of the posterior distribution. By assuming a hidden Markov process (Eq. 2.2), we can derive the following formulation for the recursive posterior distribution [122]:

$$\begin{aligned}\mathrm{p}(\mathcal{X}_t|\mathcal{Y}_t) &\propto \mathrm{p}(\mathbf{y}_t|\mathcal{X}_t, \mathcal{Y}_{t-1})\,\mathrm{p}(\mathcal{X}_t|\mathcal{Y}_{t-1}) \\ &= \mathrm{p}(\mathbf{y}_t|\mathbf{x}_t)\,\mathrm{p}(\mathbf{x}_t|\mathcal{X}_{t-1}, \mathcal{Y}_{t-1})\,\mathrm{p}(\mathcal{X}_{t-1}|\mathcal{Y}_{t-1}) \\ &= \mathrm{p}(\mathbf{y}_t|\mathbf{x}_t)\,\mathrm{p}(\mathbf{x}_t|\mathbf{x}_{t-1})\,\mathrm{p}(\mathcal{X}_{t-1}|\mathcal{Y}_{t-1})\,. \end{aligned} \tag{2.24}$$

By applying similar techniques as presented in the previous section, we compute the importance weights as

$$w^{(i)} = \frac{\mathrm{p}(\mathbf{y}_t|x_t^{(i)})\,\mathrm{p}(x_t^{(i)}|x_{t-1}^{(i)})\,\mathrm{p}(x_{0:t-1}^{(i)}|\mathcal{Y}_{t-1})}{\pi(x_{0:t}^{(i)}|\mathcal{Y}_t)}\,. \tag{2.25}$$

To get a recursive formula for the importance weight computation, we can write the recursive formulation of the importance distribution for $\mathbf{x}_t$ as

$$\pi(\mathcal{X}_t|\mathcal{Y}_t) = \pi(\mathcal{X}_t|\mathcal{X}_{t-1}, \mathcal{Y}_t)\,\pi(\mathcal{X}_{t-1}|\mathcal{Y}_{t-1})\,. \tag{2.26}$$

Following similar derivation steps as given in [122], the recursive formula-

tion for the importance weights can be then derived as

$$w_t^{(i)} = \frac{\mathrm{p}(\mathbf{y}_t|x_t^{(i)})\,\mathrm{p}(x_t^{(i)}|x_{t-1}^{(i)})}{\pi(x_t^{(i)}|x_{t-1}^{(i)}, \mathcal{Y}_t)}\, w_{t-1}^{(i)}\,. \tag{2.27}$$

By incorporating the new weight update scheme, the pseudo-code for SIS is given in Algorithm 6. One thing to notice in Algorithm 6 is line 8, where a sample is drawn from the importance distribution. We assume a Markovian importance function such that

$$\pi(\mathbf{x}_t|\mathcal{X}_{t-1}, \mathcal{Y}_t) = \pi(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathcal{Y}_t)\,. \tag{2.28}$$

Since we assume that each state $\mathbf{x}_t$ follows a first-order Markov process, using the same assumption for importance sampling is fine and in fact, saves us from storing the full history of importance functions. This enables a faster execution of the SIS algorithm.

One thing to notice here is that the weights (Eq. 2.27) suffer from the *weight degeneracy problem* as depicted in Fig. 2.6, which is due to the unstable nature of the sequential Monte Carlo sampling. In SIS, the importance weights usually have an increasing variance, which builds up with each time step and causes an inaccurate estimate [173, 175]. Several techniques have been proposed to deal with the degeneracy problem [7, 176].

2.5 Sequential Importance Resampling

Particle filtering (PF) algorithms are based on *sequential importance resampling* [7–9, 177] (SIR) and are also called SMC methods. They address the *weight degeneracy problem* of SIS by introducing an additional resampling step for weight equalization. The resampling step ensures that most of the particles remain active and a lot of computational resources are saved.

Resampling in SIR is done by sampling a new set of N particles from the discrete distribution represented by old particle weights, and discarding the old particle set. This enables duplicating "heavy" particles and getting rid of "light" ones. While the discrete distribution is still the same,

Algorithm 6 Sequential Importance Sampling (SIS)

procedure SIS
  for $i = 1 \rightarrow N$ **do** ▷ Initialization, t=0
    $w_0^{(i)} \leftarrow 1/N$
    $x^{(i)} \sim \pi(\mathbf{x}_0)$ ▷ Draw a sample from the prior
  end for
  for $t = 1 \rightarrow T$ **do**
    for $i = 1 \rightarrow N$ **do** ▷ SIS step
      $x_t^{(i)} \sim \pi(\mathbf{x}_t | \mathbf{x}_{t-1}, \mathcal{Y}_t)$ ▷ Draw a sample from importance distribution
      $\tilde{w}_t^{(i)} \leftarrow w_{t-1}^{(i)} \frac{\mathrm{p}(\mathbf{y}_t|\theta_t^{(i)}) \, \mathrm{p}(\theta_t^{(i)}|\theta_{t-1}^{(i)})}{\pi(x_t^{(i)}|x_{t-1}^{(i)}, \mathcal{Y}_t)}$ ▷ Update its weight
    end for
    for $i = 1 \rightarrow N$ **do** ▷ Normalize the weights
      $w_t^{(i)} \leftarrow \frac{\tilde{w}_t^{(i)}}{\sum_{j=1}^{N} \tilde{w}_t^{(j)}}$
    end for
  end for
end procedure

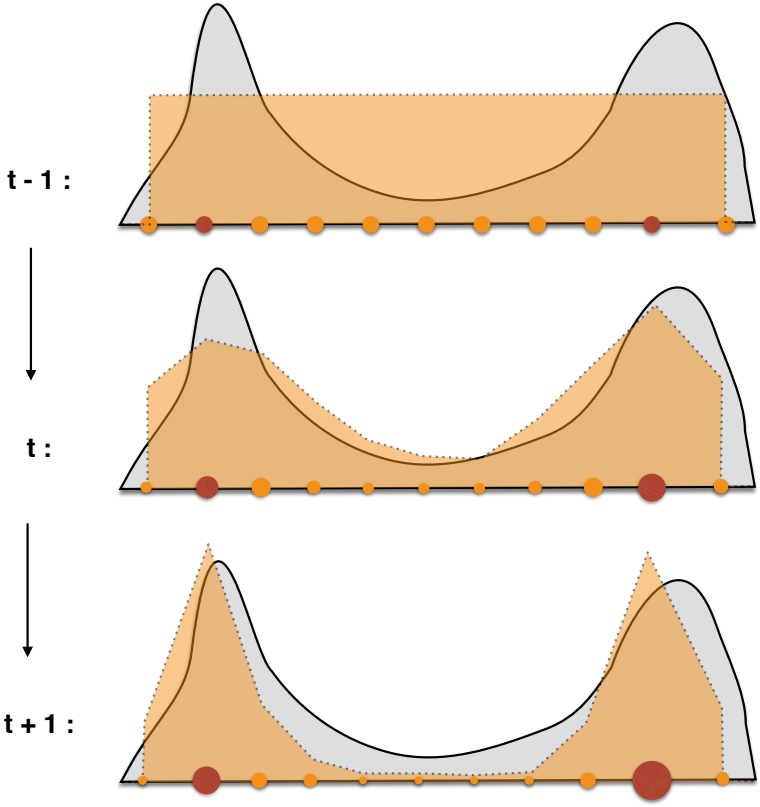


Figure 2.6: The weight degeneracy problem of SIS. Initially, all particles have the same weight. However, the particles (red) with heavier weights will become more and more dominant with time. This causes the *weight degeneracy problem*, where computations are wasted for almost trivial (zero) particle weight updates. In this example, the particles in the middle region have almost zero weights at time $t + 1$ and contribute almost nothing to the solution.

the choice of selecting particles for duplication may entail additional variance to the obtained estimates. A good choice for resampling is *stratified resampling* [177], which is optimal with respect to added variance. A general discussion on resampling algorithms can be found in [178].

The resampling phase, as depicted in Fig. 2.7, is typically executed after every m-th sampling step (i.e., fixed resampling schedule) and importance weights of particles $w_t^{(i)}$ are reset as

$$w_t^{(i)} = \frac{1}{N}\,, \tag{2.29}$$

after each resampling. Defining a criterion when to do resampling is an important issue since resampling adds computational cost to the SIS algorithm and thus, it needs to be executed smartly. Luckily, a more advanced resampling strategy becomes handy here. The *adaptive resampling* uses a criterion called "effective number of particles" [179] to monitor the variance of the particle weights. Whenever it falls below a threshold $N_{\texttt{threshold}}$, the resampling step is launched. The effective number of particles is computed as follows

$$\widehat{N}_{\texttt{eff}} = \frac{1}{\sum_{i=1}^{N}\left(w_t^{(i)}\right)^2}\,. \tag{2.30}$$

With the inclusion of the resampling step, the SIR algorithm is shown in Algorithm 9.

The algorithmic complexity of particle filters depend on the number of employed particles. Since the convergence rate of particle filters is $\mathrm{O}\left(N^{-0.5}\right)$, a large number of particles is typically required to obtain a satisfactory estimation accuracy. Depending on the application, this may have an adverse affect on the execution time of a particle filter. Thus, faster execution of such applications is much desired and one can again apply *algorithm* and *implementation engineering* principles. In the next chapter, we give an example how we address the computational bottleneck of the SIR algorithm by using an algorithmic improvement in the particle weight update phase.

Algorithm 7 Sequential Importance Resampling (SIR)

procedure SIR
  for $i = 1 \to N$ **do** ▷ Initialization, t=0
    $w_0^{(i)} \leftarrow 1/N$
    $x^{(i)} \sim \pi(\mathbf{x}_0)$ ▷ Draw a sample from the prior
  end for
  for $t = 1 \to T$ **do**
    for $i = 1 \to N$ **do** ▷ SIS step
      $x_t^{(i)} \sim \pi(\mathbf{x}_t | x_{t-1}^{(i)}, \mathcal{Y}_t)$ ▷ Draw a sample from *discrete* importance distribution
      $\tilde{w}_t^{(i)} \leftarrow w_{t-1}^{(i)} \frac{\mathrm{p}(\mathbf{y}_t|\theta_t^{(i)})\,\mathrm{p}(\theta_t^{(i)}|\theta_{t-1}^{(i)})}{\pi(x_t^{(i)}|x_{t-1}^{(i)},\mathcal{Y}_t)}$ ▷ Update its weight
    end for
    for $i = 1 \to N$ **do** ▷ Normalize the weights
      $w_t^{(i)} \leftarrow \frac{\tilde{w}_t^{(i)}}{\sum_{j=1}^{N} \tilde{w}_t^{(j)}}$
    end for
    $\widehat{N}_{\text{eff}} \leftarrow 1/\sum_{j=1}^{N} (w_k^{(j)})^2$ ▷ Calculate the effective sample size
    if $\widehat{N}_{\text{eff}} < N_{\text{threshold}}$ **then** ▷ Resampling step
      Sample a set of indices $\{s(i)\}_{i=1,\ldots,N}$ distributed such that $\Pr[s(i) = l] = w_k^{(l)}, \quad l = 1 \to N$.
      for $i = 1 \to N$ **do** ▷ Reset the weights
        $\mathbf{x}_t^{(i)} \leftarrow \tilde{\mathbf{x}}_k^{s(i)}$
        $w_k^{(i)} \leftarrow 1/N$
      end for
    end if
  end for
end procedure

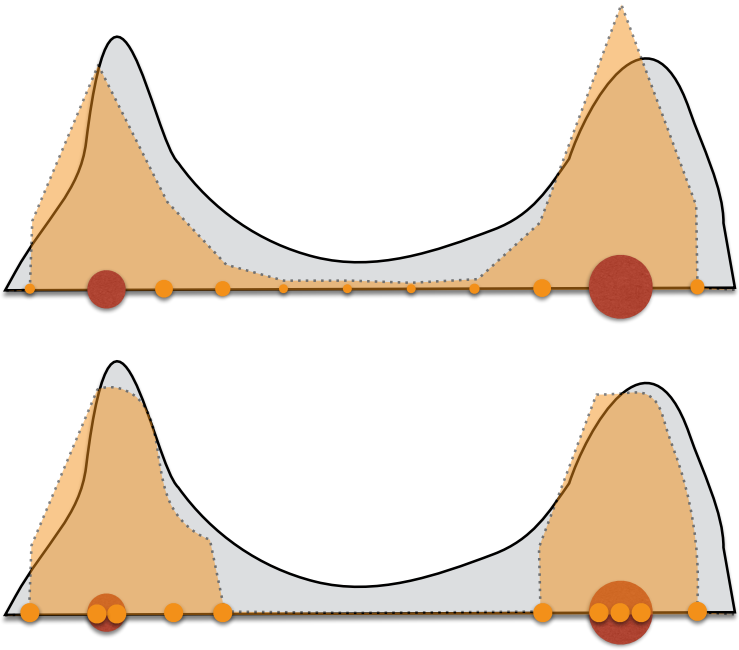


Figure 2.7: A resampling step is executed after updating the importance weights in SIS. During resampling, heavy particles (red) generate new particles close to their locations and light particles disappear. This helps discovering the modes of the hidden pdf faster. After resampling, all weights are reset to $1/N$.

CHAPTER

THREE

Piecewise Constant Sequential Importance Resampling

[1]A standard PF algorithm consists of two parts: (i) SIS and (ii) resampling [9]. A popular combined implementation of these two parts is the SIR algorithm, which is explained in detail in the previous chapter. Depending on the application, SIR may need a large number of particles to adequately sample the state space. This demands substantial computational resources that scale linearly with the number of particles and may hinder actualization of many practical real-time applications. In this work, we aim to apply *algorithm engineering* principle to PF to make it run faster and thus, allow more precise estimation results and/or real-time evaluation of medium-sized estimation problems.

Here, we introduce the *piecewise constant SIR* (pcSIR) algorithm, which reduces the computational cost of SIR while providing tracking accuracy comparable to standard SIR. The main idea behind pcSIR is to group particles in state space (i.e., creating *bins*) and to represent each group of particles by a single *representative particle*. Only the weight of this representative dummy particle is then updated. We choose the dummy particle to sit in the center of mass of the group of particles it represents and

[1]This work has been done together with Dr. Ihor Smal who helped designing test cases and co-developed the pcSIR algorithm.

to carry the mean properties of all the particles in the respective group. This is inspired by first-order multipole expansions from particle function approximation theory [180]. Once the weight of the dummy particle is computed, all other particles in the same group receive the same weight, which is copied from the dummy instead of being re-computed through the likelihood model for each individual particle, as in the original SIR. This way, a pcSIR-based PF can outperform a classical SIR-based PF by orders of magnitude in overall runtime in applications where evaluation of the likelihood is computationally expensive. Expensive likelihoods are particularly common when tracking objects in images, where each likelihood evaluation entails a numerical simulation of the image-formation process (see, e.g., Ref. [57]).

We outline the mathematical roots of pcSIR and derive an upper bound on the expected approximation error with respect to the chosen *bin* (i.e., Cartesian mesh *cell* in 2D) size. This error stems from the point-wise approximation of the likelihood function and is quantified using mid-point Riemann-sum error analysis [181, 182]. We numerically quantify the errors in the state estimates (based on the posterior distribution) obtained by SIR and pcSIR as a function of the number of particles used, and show that there is almost no difference between SIR and pcSIR in terms of tracking accuracy. Furthermore, with a focus on biological image processing, we show that relating the bin size to the pixel size of an image provides satisfactory, and sometimes even higher-quality results in pcSIR compared with standard SIR.

3.1 Related Works

Recent years have seen great interest in challenging tracking problems where the targets usually have non-linear and/or non-Gaussian dynamics. Two nonparametric algorithms, namely the histogram filters (HF) and PFs, stand out amongst others as main classes of algorithms that successfully tackle difficult tracking problems [62]. In both variants, posterior distributions are approximated by a finite set of values.

In HF, the state space is decomposed into smaller – usually rectangular –

boxes and only a single value is used to represent the cumulative posterior in each box. In a mathematical sense, HFs can be seen as piecewise constant approximations to a posterior distribution. The size and the number of the boxes affect the computational runtime and tracking accuracy of an application.

In PF, random samples (i.e., point particles) are drawn from the importance distribution, which is an approximation to the posterior distribution. Typically, a large number of particles is required to track targets successfully. This increases the computational resources needed, and many PF-based applications are limited by their computational cost.

Combining ideas from PF and HF, the box particle filter (BPF) [183] uses box-shaped particles. While BPF resembles HF with mobile boxes, these box particles are propagated based on interval analysis [184], which is fundamentally different from PF and HF. BPF is especially useful in situations where imprecise measurements yield wide posterior densities [185]. Despite its advantages, however, BPF is not well understood and lacks important theoretical background, such as a proof of convergence and insight into the resampling step based on interval analysis [185]. Also, its exact computational cost yet remains to be investigated and compared with traditional HF and PF.

3.2 The Piecewise Constant SIR

In classical SIR, all particle weights are updated according to the likelihood, which may impart a high computational load. Moreover, the computational cost scales linearly with the number of particles. Therefore, depending on the application, the likelihood evaluation often constitutes the most time-consuming part of a PF.

To address this problem, we propose the pcSIR algorithm, which aims at reducing the computational cost of importance weight update by exploiting the nature of the particle function approximation underlying SIR [180]. We do this by grouping the particles into non-overlapping multi-dimensional *bins* (i.e., Cartesian mesh cells in higher dimensions), which are then rep-

resented by only a single dummy particle positioned at the center of mass of the real particles in that bin. The center of mass is computed using the state vectors and weights of all particles within the bin and is solely used to *represent* that bin by a single *dummy particle*. This amounts to a first-order multipole expansion of the pdf approximated by the particles [180]. Higher-order approximations are easily possible by storing on the dummy particle not only the mean, but also higher-order moments of the particle distribution in the bin. However, the overall error of a PF is dominated by the Monte-Carlo sampling error, which is of order 1/2. A first-order function approximation is hence sufficient.

The importance weight update is then only applied to the dummy particle. All other particles in the same bin are assigned the same weight that the dummy particle received. Thus, we approximate the likelihood by a mixture of uniform pdfs and bypass the costly likelihood update step for all particles. The pcSIR algorithm differs from SIR only in the SIS part, where the particles are binned and several averaging operations are performed. This makes it straightforward to implement pcSIR in any existing SIR code. The detailed pseudo-code is given in Algorithm 8.

The final function approximation used in pcSIR is related to BPF, where the box support is also approximated by a mixture of piecewise constant functions [186]. However, the theoretical motivation and the algorithmic implementation of this piecewise constant approximation is very different in pcSIR and in BPF. Gning *et al.* [185, 186] used interval analysis [184] to show that the uniform PDF approximation of the *posterior* becomes more accurate as the number of intervals increases. In pcSIR, the piecewise constant approximation of the *likelihood* is rooted in particle function approximation theory and can be understood as a first-order multipole expansion [180]. This dispenses with the need for interval analysis and provides a different algorithmic implementation and error analysis.

Unlike BPF, pcSIR still uses point particles. Thus, state estimation problems that result in narrow posterior densities can easily be handled by pcSIR, which is not the case for BPF. With pcSIR, we provide a simple way of using uniform PDFs to approximate the *likelihood* function, which eventually results in a satisfactory posterior representation through the Bayesian formulation. Moreover, it requires only few modifications to the

Algorithm 8 Piecewise Constant Sequential Importance Resampling (pc-SIR)

procedure pcSIR
  for $i = 1 \rightarrow N$ **do** ▷ Initialization, t=0
    $w_0^{(i)} \leftarrow 1/N$
    $\theta^{(i)} \sim \pi(\mathbf{x}_0)$ ▷ Draw a sample from the prior
  end for
  Create B bins of equal size $I_{1,\dots,B}$
  for $k = 1 \rightarrow K$ **do**
    for $i = 1 \rightarrow N$ **do** ▷ piecewise constant SIS step
      $\theta_t^{(i)} \sim \pi(\mathbf{x}_t|\theta_{t-1}^{(i)}, \mathcal{Y}_t)$ ▷ Draw a sample from *discrete* importance distribution
      Assign $\theta_k^{(i)}$ to a bin
    end for
    for $j = 1 \rightarrow B$ **do** ▷ Visit all bins
  ▷ Create a *representative* particle that has the mean values of the state vector of all particles in the same bin
      $\theta_{\text{dum}} \leftarrow \text{mean}\{\theta_k^{(1)}, \dots, \theta_k^{(N_{I_j})}\}$
      $w_{\text{dum}_k} \leftarrow w_{\text{dum}_{k-1}} \frac{p(\mathbf{y}_t|\theta_{\text{dum}})p(\theta_{\text{dum}}|\theta_{k-1}^{(i)})}{\pi(\mathbf{x}_{\text{dum}}|\theta_{k-1}^{(i)}, \mathbf{Z}^k)}$ ▷ Update importance weights
      for all $\tilde{\theta}_k^{(i)}$ in bin I_j **do**
        $w_k^{(i)} \leftarrow w_{\text{dum}_k}$
      end for
    end for
    $\widehat{N}_{\text{eff}} \leftarrow 1/\sum_{j=1}^{N}(w_k^{(j)})^2$ ▷ Calculate the effective sample size
    if $\widehat{N}_{\text{eff}} < N_{\text{threshold}}$ **then** ▷ Resampling step
      Sample a set of indices $\{s(i)\}_{i=1,\dots,N}$ distributed such that $\Pr[s(i) = l] = w_k^l, \quad l = 1 \rightarrow N$.
      for $i = 1 \rightarrow N$ **do**
        $\theta_k^{(i)} \leftarrow \theta_k^{s(i)}$
        $w_k^{(i)} \leftarrow 1/N$ ▷ Reset the weights
      end for
    end if
  end for
end procedure

classical SIR, which makes pcSIR an attractive choice for practical implementations.

3.3 Theoretical framework

The pcSIR algorithm is a function-approximation algorithm. It divides the d-dimensional state space into d-dimensional bins. In each bin, a sufficiently differentiable likelihood function is approximated by a constant value. The error analysis of such piecewise constant approximations is well understood on the basis of Taylor's theorem for multivariate functions.

For the sake of example, we present the theoretical framework of pcSIR with a focus on image processing. When processing a sequence of 2D images, the likelihood function $p(\mathbf{y}_t|\mathbf{x}_t)$ is typically a two-dimensional function that is discretized over a finite set of particles. In SIR, the likelihood is approximated by N particles, where the particle number N defines the accuracy for the specific application. Therefore, the approximation error of SIR is denoted $\mathtt{E}_{\mathrm{SIR}}(N)$.

With pcSIR, in the considered application, only the positions of the particles play a role in the likelihood update. This allows pcSIR to bin the state space. Therefore, the approximation error in $p(\mathbf{y}_t|\mathbf{x}_t)$ depends on both the number of particles N and the maximum lengths of the bins l_x and l_y in both dimensions. Hence, the overall approximation error of pcSIR is denoted $\mathtt{E}_{\mathrm{pcSIR}}(N, l_x, l_y)$.

First, we analyze the effect of bin size on $\mathtt{E}_{\mathrm{pcSIR}}(N, l_x, l_y)$. For that purpose, we consider two cases: The first considers bins of varying rectangular shapes (i.e., $l_x \neq l_y$). In this setting, we fix N and let the approximation error depend on the bin lengths in both dimensions, hence $\mathtt{E}_{\mathrm{pcSIR}}(l_x, l_y)$. In the second case, all cells are squares of edge length l. The pcSIR approximation error can then be expressed as $\mathtt{E}_{\mathrm{pcSIR}}(l)$.

Second, we compare SIR with an pcSIR in which each bin corresponds to a single pixel in a "pseudo"-tracking test case (see Section 3.5) In this comparison, we assume Gaussian and uniform priors of different sizes. A

smooth likelihood function is approximated by SIR and pcSIR and later applied to the prior. Thus, we obtain the estimation errors for the state. We call this experiment "pseudo"-tracking, since by eliminating the explicit dynamics pdf, we can focus on the approximation error and its convergence with increasing N.

3.4 The effect of cell size on $\mathbb{E}_{\text{pcSIR}}$

The particle locations in a cell cannot be determined *a priori* since the movement of the particles depends on the data. We hence assume that for small cells and statistically large numbers of particles, we have a uniform particle distribution within a cell. The approximation errors introduced by the pcSIR algorithm in 2D are described in detail in the Appendix D.

In pcSIR, the state space is decomposed into non-overlapping cells. Choosing an appropriate cell size is hence crucial for pcSIR. Similar to histogram filters [62], the accuracy of pcSIR is determined by the cell size. In the highest possible resolution, there is one particle per cell, which recovers the classical SIR algorithm.

In image processing, it is convenient to choose the image pixels as the cells of pcSIR. This constitutes a good choice since in typical image-processing applications, the pixel size already reflects the sizes of the objects represented in the image in order not to under-sample the objects and not to store unnecessary data. We call pcSIR with single-pixel cells pcSIR-1x1. Due to the characteristics of the likelihood function, however, there may be cases where sub-pixel resolution or higher accuracy is needed. Therefore, we also investigate pcSIR-2x2, where each pixel is divided into four cells. In the following Section, we empirically benchmark the effect of cell size on pcSIR performance and accuracy.

3.5 Experimental Results

We study the performance of pcSIR by considering a biological image-processing application: the tracking of sub-cellular (here, "cell" refers to

the biological cell being imaged and is not to be confused with the pcSIR bin cells) objects imaged by fluorescence microscopy [187–189]. There, intracellular structures such as endosomes, vesicles, mitochondria, or viruses are labeled with fluorescent dyes and imaged over time with a confocal microscope. Many biological studies start from analyzing the dynamics of those structures and extracting parameters that characterize their behavior, such as average velocity, instantaneous velocity, spatial distribution [190, 191], motion correlations, etc.

3.5.1 Dynamics Model

The motion of sub-cellular objects can be represented by a variety of dynamics models, ranging from random walks to constant-velocity models to more complex dynamics where switching between motion types occurs [56, 192].

Here, we use a nearly-constant-velocity model, which is frequently used in practice [193**?**]. 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 x- and y-positions of an object, (v_x, v_y) its velocity vector, and I_0 its fluorescence intensity.

3.5.2 Likelihood / Appearance Model

Many sub-cellular objects are smaller than what can be resolved by the microscope, making them appear in a fluorescence image as diffraction-limited bright spots with an intensity profile given by the impulse-response function of the microscope, the so-called point-spread-function (PSF) [56, 57, 189].

In practice, the PSF of a fluorescence microscope is well approximated by a 2D Gaussian [194, 195]. Object appearance in a 2D image is hence modeled as:

$$I(x, y; x_0, y_0) = I_0 \exp\left(-\frac{(x - x_0)^2 + (y - y_0)^2}{2\sigma_{\mathrm{PSF}}^2}\right) + I_{\mathrm{bg}}, \tag{3.1}$$

where (x_0, y_0) is the position of the object, I_0 is its intensity, I_{bg} is the background intensity, and σ_{PSF} is the standard deviation of the Gaussian PSF. Typical microscope setups yield images with pixel edge lengths corresponding to 60 to 200 nm real-world length in the imaged sample. For the images used here, the pixel size is 67 nm and the microscope has $\sigma_{\mathrm{PSF}} = 78\,\mathrm{nm}$ (or 1.16 pixels). During image acquisition, the "ideal" intensity profile $I(x, y)$ is corrupted by measurement noise, which in the case of fluorescence microscopy has mixed Gaussian-Poisson statistics. For the resulting noisy image $\mathbf{y}_t = Y_t(x, y)$ at time point t, the likelihood $\mathrm{p}(\mathbf{y}_t|\mathbf{x}_t)$ is:

$$\mathrm{p}(\mathbf{y}_t|\mathbf{x}_t) \propto \exp\left(-\frac{1}{2\sigma_\xi^2} \sum_{(x_i,y_i)\in\mathbb{S}_{\mathbf{x}}} [Y_t(x_i, y_i) - I(x_i, y_i; \hat{x}, \hat{y})]^2\right), \tag{3.2}$$

where σ_ξ controls the peakiness of the likelihood, (x_i, y_i) are the integer coordinates of the pixels in the image, $(\hat{x}, \hat{y})$ are the spatial components of the state vector $\mathbf{x}_t$, and $\mathbb{S}_{\mathbf{x}}$ defines a small region in the image centered at the object location specified by the state vector $\mathbf{x}_t$. Here, $\mathbb{S}_{\mathbf{x}} = [\hat{x} - 3\sigma_{\mathrm{PSF}}, \hat{x} + 3\sigma_{\mathrm{PSF}}] \times [\hat{y} - 3\sigma_{\mathrm{PSF}}, \hat{y} + 3\sigma_{\mathrm{PSF}}]$.

3.5.3 Experimental Setup

We focus on single sub-cellular object tracking (a problem which is related to the "track-before-detect" problem [196]) and compare pcSIR with SIR in two test cases, which differ in the size of the tracked object. We consider two different object sizes in order to compare cases where the likelihood is computationally cheap to evaluate with cases where this is more costly. 20 synthetic image sequences of different quality (i.e., signal-to-noise ratios, SNR) are generated by simulating a microscope. Each sequence is composed of 50 frames of size 512×512 pixels. The movies show a single object moving according to the dynamics model. Examples are shown in Fig. 3.1.

The two object sizes correspond to $\sigma_{\mathrm{PSF}} = 1.16$ and $\sigma_{\mathrm{PSF}} = 13$, and are named "small object tracking" and "large object tracking", respectively (Fig. 3.1(a-c)). The positions and directions of motion of the objects are randomly chosen within the image plane. The speed (i.e., the displacement in pixels per frame) is drawn uniformly at random over the interval $[2, 7]$

for large objects and over $[2, 4]$ for small objects. The SNR of the images of large objects is 2 (ca. 6 dB), that for small objects is 4 (ca. 12 dB). We use the SNR definition for Poisson noise [197]. In the literature on sub-cellular object tracking, a SNR of 4 is considered critical, as for lower SNRs many of the available tracking methods fail [194].

Knowing the ground-truth object positions and those estimated by the PF, we quantify the tracking accuracy by the root-mean-square error (RMSE) in units of pixels. The likelihood kernel for the large objects has a support of 65×65 pixels and is correspondingly costly to evaluate. The kernel for the small objects has a support of 9×9 pixels and is cheaper to evaluate. Examples of noise-free and noisy object profiles, together with their likelihood kernels, are shown in Fig. 3.2.

Using double-precision arithmetics, a single PF particle requires 52 B (i.e., six doubles and one integer) of computer memory. The particles are initialized at the ground-truth location and all tests are repeated 50 times for different realizations of the image-noise process on a single core of a 12-core Intel® Xeon® E5-2640 2.5 GHz CPU with 128 GB DDR3 800 MHz memory on MPI-CBG's MadMax computer cluster. All algorithms are implemented in Java (v. 1.7.0_13) within the `PPF` library [2]. The results are summarized in Figs. 3.3 and 3.4 for large and small objects, respectively.

3.5.4 Results

When tracking large objects (Fig. 3.3), both pcSIR versions provide significant speedups over the classical SIR algorithm. For 12'800 particles, pcSIR-1x1 is more than two orders of magnitude faster than SIR with a 2.4% loss in tracking accuracy. pcSIR-2x2 provides an up to 5.8% better tracking accuracy than SIR while running over 50 times faster. Since SIR is also an approximation of the actual posterior distribution, in some cases pcSIR may provide a better representation of the posterior and thus a higher tracking accuracy. This phenomenon has been previously described [198].

When tracking small objects, the likelihood support requires sub-pixel resolution and the effect of bin size is more visible (Fig. 3.4). pcSIR-1x1 uses

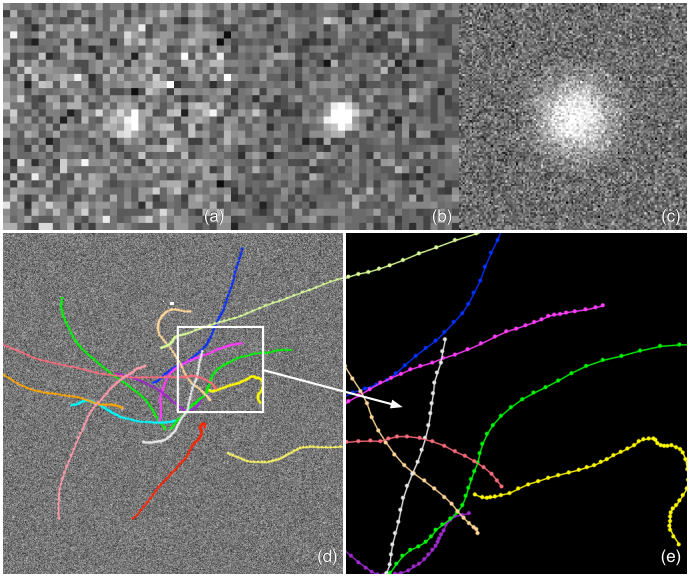


Figure 3.1: Examples of object appearance for different object sizes and SNR: (a) $\sigma_{\mathrm{PSF}} = 1.16$, SNR=2, (b) $\sigma_{\mathrm{PSF}} = 1.16$, SNR=4, (c) $\sigma_{\mathrm{PSF}} = 13$, SNR=2. (d/e): Typical object trajectories generated using the nearly-constant-velocity dynamics model.

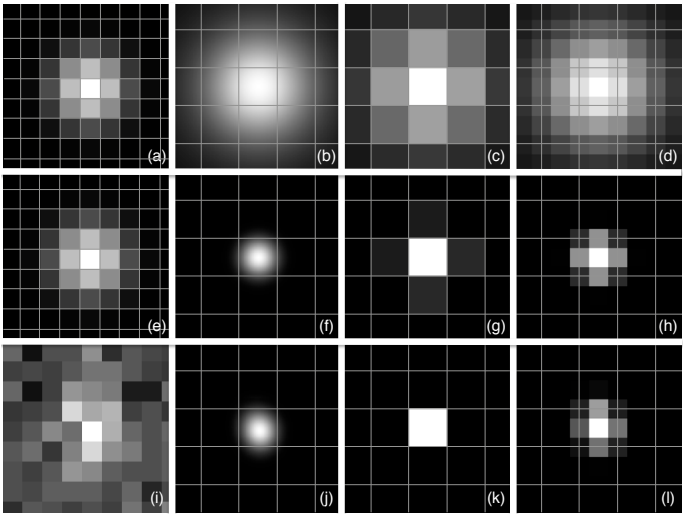


Figure 3.2: Examples of likelihood profiles. The noise-free objects are shown in (a, e), and the noisy (SNR=2) object in (i) with $\sigma_{\mathrm{PSF}} = 1.16$. We show the corresponding likelihood kernels (b, f, j), the approximated likelihoods used by pcSIR-1x1 (c, g, k), and the approximated likelihoods used by pcSIR-2x2 (d, h, l). In (b, c, d), the parameter σ_ξ is 30, for the rest $\sigma_\xi = 10$. The distance between the grid-lines corresponds to the size of the image pixel.

rather coarse bins compared to the likelihood support (Fig. 3.2), resulting in a pronounced loss of tracking accuracy. Visually, however, the trajectories produced by SIR and pcSIR-1x1 are virtually indistinguishable, since the tracking accuracy of pcSIR-1x1 is still in the sub-pixel regime (about 0.27 pixel). When finer bins (pcSIR-2x2) are used, the tracking accuracy of pcSIR is again better than that of SIR, and pcSIR runs more than five times faster than SIR.

3.5.5 Convergence of SIR and pcSIR

Both SIR and pcSIR employ particle approximations of a smooth, differentiable function, the order of accuracy of which depends on the number of particles N. In order to eliminate uncertainties resulting from the dynamics model, we assume the prior $\mathrm{p}(\mathbf{x}_t|\mathbf{y}_{t-1})$ to be either a uniform distribution over 3×3 or 5×5 pixels, or a Gaussian with $\sigma_{\mathrm{prior}} = \{0.5, 0.8\}$, respectively. We then evaluate the likelihood in Eq. (3.2) with $\sigma_\xi = 20$ using both SIR and pcSIR. We call this a "pseudo"-tracking experiment. The object is a single PSF (Eq. (3.1)) with $\sigma_{\mathrm{PSF}} = 1.16$. Visualizations of the object, likelihood, and prior are shown in Fig. 3.5.

We compare two versions of pcSIR, which differ in the placement of the dummy particles: In pcSIR-CoC, the dummy particles are placed at the geometric centers of the bins, whereas in pcSIR-CoM, the centers of mass of the state vectors of all particles inside that bin are used. Each convergence experiment is repeated 1'000 times for different realizations of the random process, and the number of particles is increased up to 100'000. We quantify the RMSE of the state estimation as a function of the number of particles used. The resulting convergence plots for pcSIR and SIR are shown in Fig. 3.6.

We observe no significant differences between SIR and the two pcSIR variants. The error of pcSIR-CoM is always slightly lower than that of pcSIR-CoC. SIR is generally the most accurate, but is outperformed by pcSIR-CoM in some cases, confirming our experimental tests as well as the findings in Ref. [198]. As N increases, the errors of all methods decrease at the same rate. In all cases, however, the runtimes of both pcSIR variants were significantly less than that of SIR.

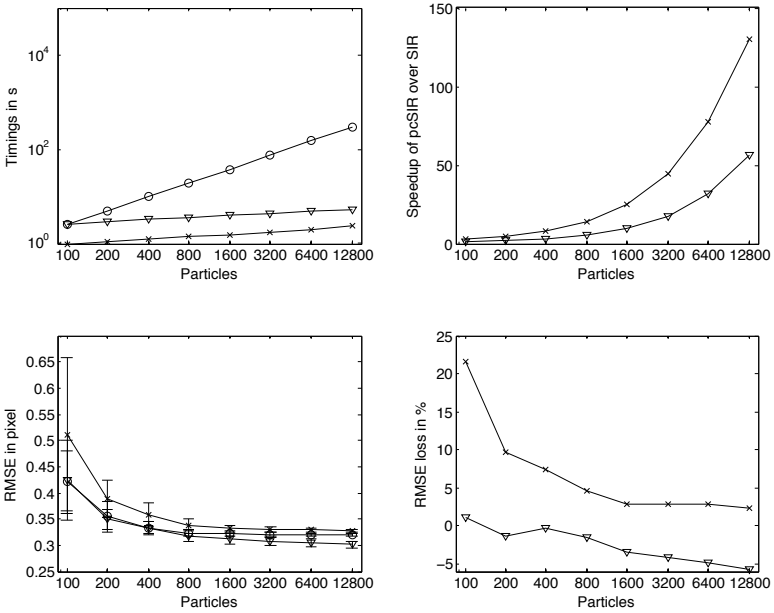


Figure 3.3: Runtime performance and tracking accuracy of pcSIR-1x1 ($\times$) and pcSIR-2x2 ($\triangledown$) compared with SIR ($\circ$) for a 65 pixel wide likelihood kernel. The number of particles used starts from 100 and is doubled for each case until 12'800. The timings of all three methods are presented in log-log scale (upper left), whereas the relative speedups of the pcSIR methods over SIR are shown in the upper-right plot. The accuracy loss (lower right) of pcSIR-1x1 drops rapidly as the number of particles in the system is increased. Error bars show standard deviations across the 50 repetitions of each experiment.

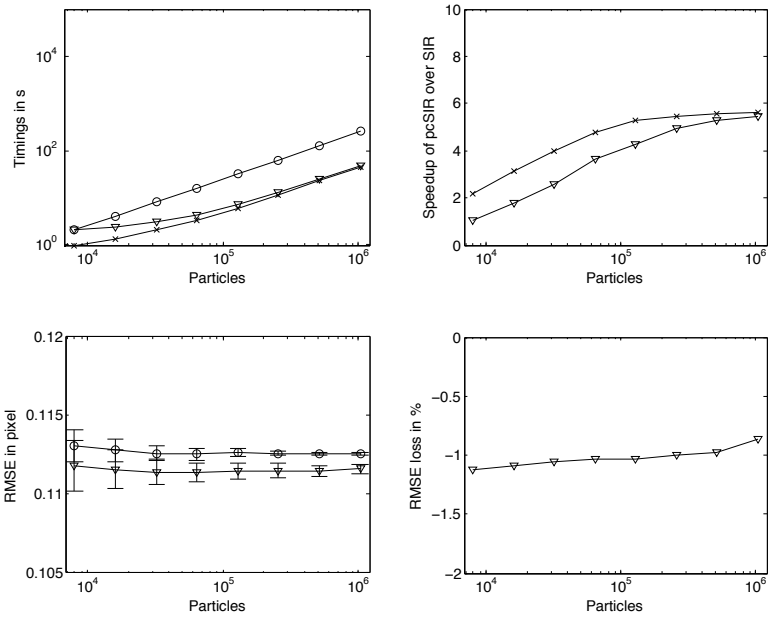


Figure 3.4: Runtime performance and tracking accuracy of pcSIR-1x1 ($\times$) and pcSIR-2x2 ($\triangledown$) compared with SIR ($\circ$) for a nine-pixel wide likelihood kernel. The number of particles used starts from 8'000 and is doubled for each case until 1'024'000. The timings of all three methods are presented in log-log scale (upper left), whereas the relative speedups of the pcSIR methods over SIR are shown in the upper-right plot. For the accuracy comparisons (lower left), we show only the results for pcSIR-2x2 and SIR, since pcSIR-1x1's coarse bin resolution results in a 150% worse tracking accuracy than SIR. Error bars show standard deviations across the 50 repetitions of each experiment.

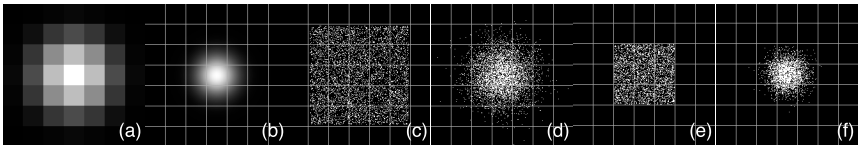


Figure 3.5: The "pseudo"-tracking experiment: (a) the object with $\sigma_{\mathrm{PSF}} = 1.16$, SNR=2; (b) the corresponding likelihood with $\sigma_{\xi} = 20$; (c) a uniform prior of support 5×5 pixel; (d) a Gaussian prior with $\sigma_{\mathrm{prior}} = 0.5$; (e) a uniform prior of support 3×3 pixel; (f) a Gaussian prior with $\sigma_{\mathrm{prior}} = 0.8$. Thin white lines indicate the image pixel grid.

3.6 Conclusions

We proposed a fast approximate SIR algorithm, called pcSIR. pcSIR is based on spatially binning particles in *cells* and representing each cell by a single dummy particle at the center of mass of the cell's particle distribution, carrying the average state vector of all particles in that cell. This approximates the likelihood by a first-order multipole expansion [180]. pcSIR significantly reduces the computational cost of SIR and enables tackling larger problems as well as tackling mid-size problems in real time. In some configurations, especially when sub-pixel resolution is used for the bins, pcSIR may yield more accurate results than SIR.

We performed both theoretical and experimental error analysis of pcSIR. We showed that the error in the posterior decreases as the number of particles increases. Moreover, pcSIR converges at the same rate as SIR, since the Monte-Carlo sampling error masks the error from the function approximation. We presented theoretical upper bounds on the likelihood approximation error as a function of cell size in pcSIR.

We experimentally tested the tracking accuracy and runtime performance of two pcSIR variants for image processing: pcSIR-1x1 and pcSIR-2x2. In our benchmarks, pcSIR showed significant speedups over SIR. As more particles are used, the relative speedup over SIR seems to grow exponentially for large-object tracking scenarios, where the likelihood is costly to

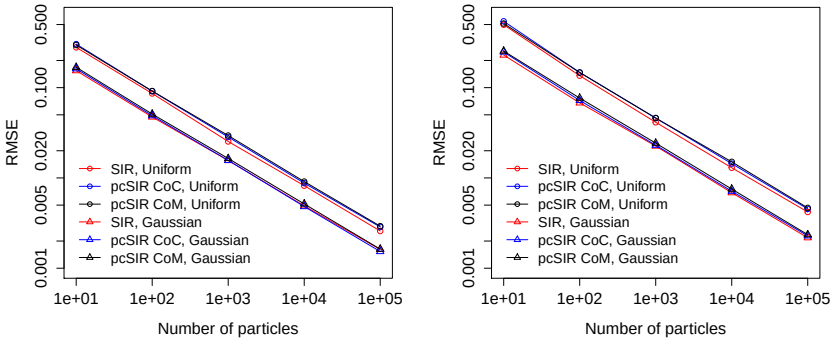


Figure 3.6: The "pseudo"-tracking experiment results for the Gaussian prior with $\sigma_{\text{prior}} = 0.5$ and the uniform prior with 3×3-pixel support (left), and for the Gaussian prior with $\sigma_{\text{prior}} = 0.8$ and the uniform prior with 5×5-pixel support (right). The state estimation errors of pcSIR relative to SIR range between $-12\% \ldots +6\%$. The difference between pcSIR and SIR decreases as N increases. Both pcSIR and SIR converge with increasing number of particles. The RMSE error is reduced by about 30% every time the number of particles doubles, corresponding to a convergence order of $\sqrt{N}$, as expected for a Monte Carlo method. Error bars are below symbol size.

evaluate. In the presented benchmarks with 12'800 particles, SIR required 5 minutes to track the large object through a 50-frame 2D image sequence. pcSIR-1x1 needed only 2.3 seconds to accomplish the same task at the expense of a 2.4% smaller accuracy. pcSIR-2x2 completed the task in 5.3 seconds with a 5.8% better tracking accuracy than SIR. This improvement stems from the fact that for some posterior distributions, the piecewise constant likelihood approximate of pcSIR may be a more regular representation than that generated by SIR. This is a known phenomenon [198]. The relative speedups of the two pcSIR variants over classical SIR were *130*-fold and *57*-fold for 12'800 particles, respectively. For larger numbers of particles, we expect even larger speedups.

For small-object tracking, both pcSIR variants showed an average 5-fold improvement in execution time for the largest tested particle number. However, the tracking accuracy of pcSIR-1x1 is greatly reduced, since the likelihood function has a narrow support that is not well sampled by the coarse bins. While the errors are in the range of 150%, they are barely visible in the final trajectories since the average RMSE is only about 0.27 pixels. Interestingly, pcSIR-2x2 shows improvements both in overall runtime (5-fold) and in tracking accuracy (1%), which suggests that pcSIR-2x2 may be a good algorithm for tracking small objects.

We believe that pcSIR can be used in many PF applications that require large numbers of particles, costly likelihood evaluations, or real-time performance. When tracking accuracy is not critical, pcSIR-1x1 can offer orders of magnitude speedup in image-processing applications. If a loss in tracking accuracy is undesired, pcSIR-2x2 still offers significant speedups while in some cases even improving accuracy over SIR. In other applications, one can adjust the size of the averaging bins according to the desired accuracy. Future improvements could involve adaptive bin sizes.

CHAPTER

FOUR

Distributed Resampling Algorithms and Parallel Particle Filtering Library

4.1 Introduction

[1]A main drawback of the PF is its inherently high computational cost, which scales with the number of particles (i.e., samples) used to approximate the posterior distribution. In the past, high-precision real-time applications have been a great challenge to the PF algorithms. Due to their computational cost, many PF applications remain limited to small problems or require long execution times. To address this problem, several algorithmic improvements have been proposed [1, 199, 200].

As computer clusters became more accessible, redesigning PF algorithms to take advantage of such parallel computing systems appeared a natural progress in PF research. The parallelization of PF codes is achieved via *non-geographic domain decomposition* approach where each PE is given only a portion of the available workload (i.e., particles). Thus, the overall computational cost per PE is reduced and the execution of the PF application is accelerated. Faster PF codes enable developers to tackle problems of larger size in real time, which may have a big impact in many research

[1]This work has been done in collaboration with Dr. Ihor Smal who co-designed and co-developed the Parallel Particle Filtering library.

and application areas. As a major attempt in that direction, Bolić *et al.* introduced two distributed algorithms in their seminal work [201]: (i) The distributed resampling algorithm with proportional allocation (RPA) and (ii) the distributed resampling algorithm with non-proportional allocation (RNA). These algorithms enabled the development of PF applications that efficiently use modern multi-core and multi-processor hardware, such as computer clusters.

In this chapter, we recapitulate recent developments in parallel PF research with a particular focus on DRAs, namely, RPA and RNA. We give a detailed explanation of the distributed design of these algorithms. Following the summary, we introduce the `PPF` library [2], which provides efficient implementations of these DRAs on hybrid shared-/distributed-memory architectures based on non-geographic domain decomposition.

4.2 Parallel Particle Filtering Algorithms

A generic PF algorithm consists of two parts: (i) SIS and (ii) resampling [202]. A popular combined implementation of these two parts is the SIR algorithm [202].

Recursive Bayesian importance sampling [203] of an unobserved and discrete Markov process $\{\mathbf{x}_t\}_{t=1,\dots,T}$ is based on three components: (i) the measurement vector $\mathcal{Y}_t = \{\mathbf{y}_1, \dots, \mathbf{y}_t\}$, (ii) the dynamics (i.e., state-transition) model, which is given by a probability distribution $\mathrm{p}(\mathbf{x}_t|\mathbf{x}_{t-1})$, and (iii) the likelihood (i.e., observation model) $\mathrm{p}(\mathbf{y}_t|\mathbf{x}_t)$. Then, the state posterior $\mathrm{p}(\mathbf{x}_t|\mathcal{Y}_t)$ at time t is recursively computed as:

$$\underbrace{\mathrm{p}(\mathbf{x}_t|\mathbf{y}_t)}_{\text{posterior}} = \frac{\overbrace{\mathrm{p}(\mathbf{y}_t|\mathbf{x}_t)}^{\text{likelihood}} \overbrace{\mathrm{p}(\mathbf{x}_t|\mathcal{Y}_{t-1})}^{\text{prior}}}{\underbrace{\mathrm{p}(\mathbf{y}_t|\mathcal{Y}_{t-1})}_{\text{normalization}}}, \tag{4.1}$$

where the prior is defined as:

$$\mathrm{p}(\mathbf{x}_t|\mathcal{Y}_{t-1}) = \int \mathrm{p}(\mathbf{x}_t|\mathbf{x}_{t-1})\, \mathrm{p}(\mathbf{x}_{t-1}|\mathcal{Y}_{t-1})\, \mathrm{d}\mathbf{x}_{t-1}. \tag{4.2}$$

PFs approximate the posterior at each time point t by N weighted samples (i.e., particles) $\{\mathbf{x}_t^{(i)}, w_t^{(i)}\}_{i=1,\ldots,N}$. 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) [202]. However, SIS suffers from *weight degeneracy*, whereby small particle weights become successively smaller and do not contribute to the posterior any more. To address this problem, a *resampling* step is performed [202] whenever the number of particles with relatively high weights falls below a specified threshold. The complete SIR algorithm is given in Algorithm 9.

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. Here, the particle data is parallelized and each PE work on the same input. An example from biological image processing and how the particle data is distributed across PEs is shown in Fig. 4.1.

Algorithm 9 Sequential Importance Resampling (SIR)

1: (P) Propagate all particles according to the transition prior: $\mathbf{x}_t^{(i)} \sim \mathrm{p}(\mathbf{x}_t|\mathbf{x}_{t-1}^{(i)})$, $i = \{1, \ldots, N\}$
2: (U) Update the weights taking into account the measurements at time t, $\mathbf{y}_t$, as $\tilde{w}_t^{(i)} = \mathrm{p}(\mathbf{y}_t|\mathbf{x}_t^{(i)}) w_{t-1}^{(i)}$
3: Renormalize the weights as $w_t^{(i)} = \tilde{w}_t^{(i)} / \sum_{j=1}^{N} \tilde{w}_t^{(j)}$
4: Compute the estimate $\hat{\mathbf{x}}_t = \sum_{i=1}^{N} w_t^{(i)} \mathbf{x}_t^{(i)}$
5: Compute $N_{\text{eff}} = (\sum_{i=1}^{N} (w_t^{(i)})^2)^{(-1)}$
6: Resample if $N_{\text{eff}} < N_{\text{thresh}}$ using Systematic Resampling [177]

4.3 Distributed Resampling Algorithms

Existing DRAs can be classified into three groups: The first one includes *multiple particle filter* (MPF) algorithms [204], which is a bank of truly independent PFs with no communication during and/or after local resampling. This renders MPF an embarrassingly parallel approach, in which

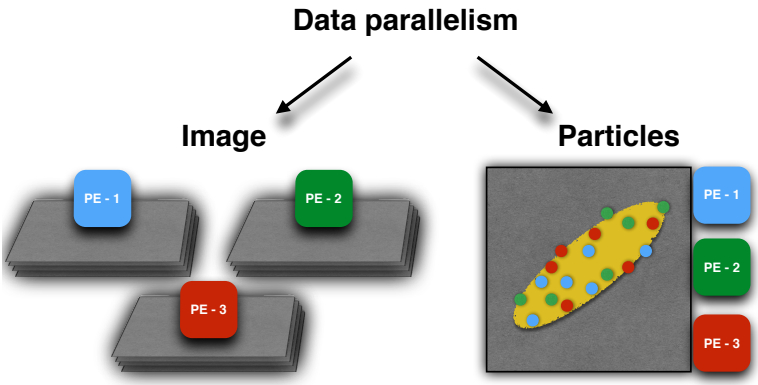


Figure 4.1: Two possible ways of data parallelism is shown for a target tracking application from biology. One possible way to provide data parallelism is to divide the *input data* into smaller chunks and assign them to PEs (left). In DRAs however, *particle data* is parallelized. All PEs work on the same image (right). A combination of both data parallelism is also possible but not included in this work.

every process essentially runs a separate PF. The only communication in this case is sending the local estimates of the state to a master node that forms a combined estimate. However, this low communication complexity comes at the cost of a reduced statistical accuracy due to problems when, for example, some of the independent PFs diverge during the sequential estimation and introduce large errors into the combined estimate. If some of the independent PFs happen to explore the same area of state space, the approach is also computationally wasteful. Using a global resampling scheme, which incurs global communication between the processes, accuracy and efficiency can be improved.

4.3.1 Classical RPA

The second class of DRAs contains RPAs [201]. They are based on stratified sampling [9, 177] with proportional allocation, which means that particles with larger weights are resampled more often. Similarly, PEs with low-weight particles will have even fewer particles after resampling. There is no difference between SIR and RPA except the fact that the resampling step is done in parallel on every PE.

Initially, particle data is divided into K disjoint strata and each stratum is given to a PE. However, each PE generates different number of particles after resampling and therefore, a *particle imbalance* occurs. To overcome this problem, the classical RPA requires adaptive DLB schemes for particle routing (Fig. 4.2), where the number of communication links is minimized in order to reduce the latency in the network.

4.3.2 Classical RNA

The third DRA category includes RNA [201] and *local selection* (LS) algorithms [205]. These algorithms are designed to minimize inter-process communication. In RNA, the sample space is divided into disjoint strata, and each of them is assigned to a different process. The number of particles per PE to partially approximate the posterior is fixed. In a distributed-memory computer system with M PEs, the resampling step in RNA is performed locally by each PE. Thus, the number of particles per PE hence

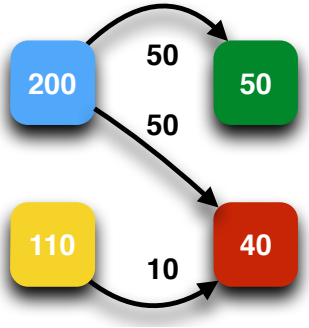


Figure 4.2: A particle imbalance situation in RPA is depicted. A dynamic particle balancing protocol is required to balance the number of particles among PEs.

remains constant and ensures perfect particle balance. However, the weight distribution across PEs can become unbalanced leading to a *particle weight imbalance* across PEs. This requires particle routing (i.e., 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 popular choice is to migrate 10%–50% of the particles of each process to the neighboring process after resampling [206, 207]. A pseudocode for RNA is shown in Algorithm 10 and an exemplary particle routing is given in Fig. 4.3. The neighborhood between processes is defined by a process topology, which usually is taken to be a ring. Using such a simple, static DLB scheme shortens application development time.

Bolić *et al.* [201] describe three methods for particle routing in RNA. *Regrouping* method creates groups of PEs, in which PEs use the resampling algorithm with proportional allocation (RPA) to balance their particle weights. Once a load balance in the groups are achieved, new groups are formed where the PEs belonging previously to different groups are put into the same one. While this method is easy to implement, it does not benefit from the knowledge about particle weights in groups.

Adaptive regrouping improves over the previous method by grouping the most overloaded PE with the most underloaded one. As in the *regrouping* method, the RPA method is employed. The problem of *adaptive regroup-*

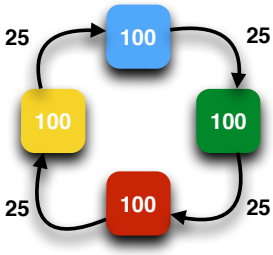


Figure 4.3: Particle weight imbalance situation in RNA is shown. RNA ensures that all PEs hold the same number of particles. However, total particle weights on PEs may differ greatly and a dynamic weight balancing protocol is typically required to balance the particle weights among PEs. Here, a fixed percentage (e.g., 25%) of particles are circled in each time step.

ing method is that it does not utilize PE locality.

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 $\mathrm{p}(\mathbf{x}_k|\mathcal{Y}_t)$ is recovered. Detailed information on RNA with LS can be found in Ref. [206].

The *local exchange* method with a particle-exchange ratio of 10% or 50% is a popular choice when implementing RNA [201, 206, 207]. 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. Moreover, the PEs are arranged in a ring and only communicate with their adjacent neighbors. A processing element P_m randomly selects N_{ex} (out of its N_p) particles and sends them to P_{m+1} where $1 < m < M$. Concurrently, it receives N_{ex} new particles from P_{m-1}.

In the following, we present our novel software library that implements

Algorithm 10 Resampling with Non-proportional Allocation (RNA)

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

RNA and RPA efficiently such that PF applications can benefit from computer clusters effectively.

4.4 Parallel Particle Filtering Library

PFs have been applied to a wide spectrum of signal-processing applications. Despite their advantages, the inherently high computational cost of PF limits their practical application, especially in real-time problems. This challenge has been addressed by algorithmic improvements [200], efficient shared-memory [208] and many-core [209–211] implementations, and scalable distributed-memory solutions [207, 212].

Here, we present a parallel software library for particle filtering, called the `PPF` library, which aims at providing application developers with an easy-to-use and scalable platform to develop PF-based parallel solutions for their applications. The main contributions of the `PPF` library are a highly optimized implementation and the extension of distributed resampling algorithms [201] to hybrid shared-/distributed-memory systems.

The library is written in Java and has an object-oriented architecture. It exploits a hybrid model of parallelism where the MPI [213] and Java threads are combined. The framework relies on the recent Java bindings of Open MPI [65, 214] for inter-process communication, and on Java threads for intra-process parallelism. Java is an emerging language in HPC offering new research opportunities [215].

The `PPF` library includes implementations of different strategies for DLB across processes, and a checkerboard-like thread balancing scheme within processes. Inter-process DLB is used in the resampling phase of a PF, whereas thread balancing (i.e., intra-process balancing) is used throughout the entire library. Non-blocking point-to-point MPI operations are exploited wherever the chosen DLB strategy allows for them. Furthermore, the framework has interfaces for ImageJ [216], Fiji [217], and `imagescience` [218], allowing these popular image-processing applications to directly access `PPF`'s application programming interface (API). The proposed framework can be used to facilitate implementation of other computationally demand-

ing PF based techniques such as pMCMC [219], SMC^2 [220], and SMS-(C)PHD [221].

The `PPF` library uses MPI [213] for inter-process communication, which is the *de facto* standard for parallel high-performance computing. The Open MPI team [65] recently started providing a Java interface based on `mpiJava` [222], which covers all MPI 1.2 functions and is currently maintained on provisional basis in the development trunk of Open MPI [223][2]. While there exist several Java bindings for OpenMP, such as `JOMP` [224] or `JaMP` [225], they come either as an additional compiler on top of the Java Virtual Machine or require preprocessing before the actual Java code is produced. Moreover, they do not provide the full flexibility one can have with Java threads. Therefore, we employ Java threads for shared-memory parallelization. The `PPF` library also provides built-in support for ImageJ [216], Fiji [217], and `imagescience` [218] for file I/O, image analysis, and image editing. A schematic of the structure of the `PPF` library is shown in Fig. 4.4.

The library consists of five modules: (i) actors, (ii) models, (iii) particle, (iv) tools, and (v) interfaces. The *actors* module encapsulates functionality that is common to PF algorithms (e.g., resampling) and provides support for parallel PFs via its communication, data distribution, and DLB sub-modules. The *models* module includes dynamics and observation models. By default, the library includes simple standard models that can be sub-classed to include application-specific models. The *particle* module contains the particle data structure and related methods (e.g., particle generation). The *tools* module contains a set of helper methods for sorting, statistical calculations, efficient particle neighbor lists, etc. The *interfaces* module provides APIs to link the `PPF` library to ImageJ [216], Fiji [217], and `imagescience` [218]. This allows ImageJ/Fiji plugins to access the functionality provided by the `PPF` library, but it also allows `PPF` methods to use functions provided by ImageJ/Fiji, such as functions for image processing, file I/O, and graphical user-interface building.

A client code that implements a parallel PF application can directly call

[2]While writing this thesis Open MPI released v1.8 stable series that contain also Java bindings based on MPI 1.2 standard.

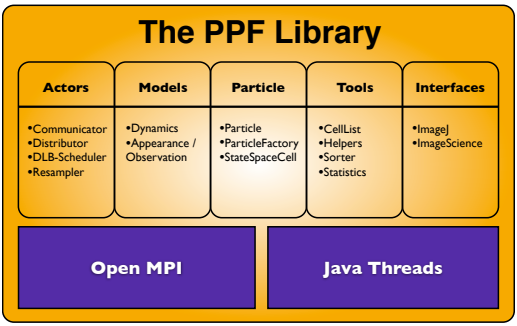


Figure 4.4: Software structure of the `PPF` library. The library is divided into five modules and hides the complexity of MPI and multithreading from the application programmer. It also supports using `imagescience` classes, as well as functions from ImageJ and Fiji.

the `PPF` API. Most of the intricacies arising from parallel programming and code optimization are hence hidden from the application programmer. Sitting as a middleware on top of MPI and Java threads, the `PPF` library makes the parallelization of PF applications on shared- and distributed-memory systems easier. Below, we highlight some of the features of the `PPF` library.

4.4.1 Multi-level Hybrid Parallelism

As HPC systems grow in size, multi-level hybrid parallelization techniques emerge as a viable tool to achieve high parallel efficiency in scalable scientific applications. Many applications [226] realize hybrid parallelization strategies by combining MPI with OpenMP [227]. In order to have full thread control and also enable job-level multi-threading, we here follow the trend of combining MPI for inter-process communication with native threads for intra-process parallelism. We employ Java's native thread concurrency model in the `PPF` library, which provides full thread control and avoids additional software/compiler installation and maintenance. Furthermore, we provide an intra-process load-balancing scheme for threads specifically designed for PF applications, whose implementation using Java threads is straightforward and easy to extend.

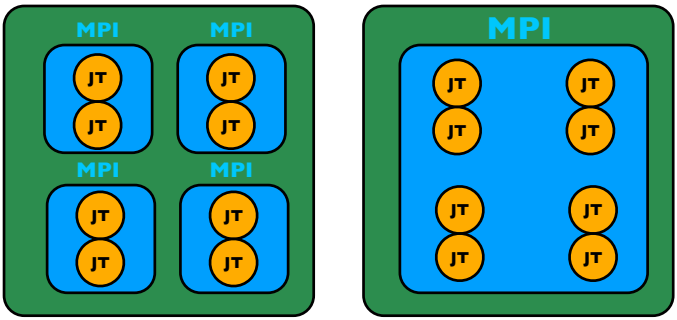


Figure 4.5: Two possible ways to use hybrid parallelism with MPI and Java threads (JT) in PPF: On the left, each MPI process is bound to a *CPU* (blue squares), and each JT is assigned to a *core* (orange circles). On the right, there is one MPI process per *node/computer* (green rectangle), and again one JT per *core*. The PPF library implements both models and lets the application developer choose which one to use for a specific application.

The PPF library lets the user choose between two different concurrency models, as illustrated in Fig. 4.5: In the first model (left panel), the number of MPI processes is equal to the number of available CPU chips, and the number of threads per process is equal to the number of cores per CPU. Compared to an all-MPI paradigm, this allows benefitting from shared caches between cores and reduces communication latency.

The second model (right panel) uses a single MPI process per *node/computer* and and one Java thread for each core. This hybrid model keeps the shared memory on the node coherent and causes a lower memory latency than an all-MPI implementation [228]. Which model one chooses for a particular application depends on the specific hardware (cache size, number of memory banks, memory bus speed, etc.) and application (data size, computational cost of likelihood evaluation, etc.)

4.4.2 Non-blocking MPI Operations

During execution of DLB strategies in RPA, we use non-blocking point-to-point MPI operations in order to overlap communication with computa-

tion. This is especially useful during the DLB phase, where a sender sends its message to a receiver and then immediately carries on with generating new local particles.

4.4.3 Input-space Domain Decomposition

In the `PPF` library, MPI-level parallelization is done at the particle-data level. This means that each MPI process has full knowledge of the input, e.g. the image to be processed, but only knows a part of the particles in state space. At the thread level, we then also decompose the input (e.g., image) into smaller subdomains. This provides a convenient way of introducing thread-level parallelism. For images, the pixels containing particles are directly assigned to local threads. Thus, both the state space is distributed via MPI, and the input space is distributed across threads.

4.4.4 Thread Balancing

Thread-level load balancing is as important as process-level DLB. When using PFs for object tracking, for example, once an object is found and locked onto, many particles are drawn to the vicinity of that object. This worsens thread load-balance, since if consecutive pixels belong to the same thread, that thread becomes overloaded while others are idle.

The `PPF` library hence uses an adaptive checkerboard-like load balancing scheme for threads, as illustrated in Fig. 4.6. Depending on the number of threads and the support of the posterior distribution, the size of the checkerboard patch is automatically adjusted.

4.4.5 Image Patches

When using PFs for image processing, the likelihood computation involves image data and may be computationally costly due to, e.g., invoking a numerical simulation of the image-formation process. Performance improvements in the likelihood calculation hence have the largest impact on the overall performance of a PF in an image-processing application.

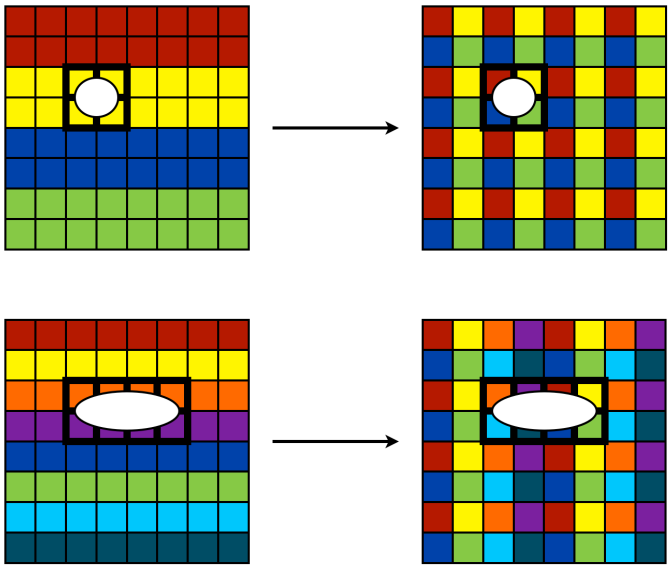


Figure 4.6: Big boxes represent the binned state space (where in the case of images the pixels can be used as bins) and the colors represent which part of the state space will be processed by which thread. Once particles concentrate around an object to be tracked, i.e. the posterior approximation converged, thread balancing accelerates the computations. The `PPF` library implements a checkerboard-like thread balancing scheme with patch sizes that depend on the support of the posterior distribution and on the number of threads. Two examples are shown here: In the 2×2 scheme (upper row), the area covered by the posterior (white circle) is distributed across four threads (upper right corner). Similarly, the 2×4 thread-balancing scheme (lower row) distributes a larger posterior (white ellipse) across eight threads.

In many image-based applications, a separate likelihood estimation is carried out for each particle, where the full image is loaded and then the likelihood kernel is applied. In image-based likelihood computations, these kernels are typically symmetric (e.g., Gaussian) and local (i.e., span only a few pixels). Thus, it is sufficient to visit pixels one by one and only load the *image patch* centered at the visited pixel. The size of the patch is given by the likelihood kernel support, which is typically much smaller than the whole image. Once a patch is loaded, computing the particle weights in the central pixel requires less time. In fact, if there are N_{pix} pixels in the image and N particles in state space, the overall computational complexity of the likelihood calculation is reduced from $\mathrm{O}(NN_{\text{pix}})$ to $\mathrm{O}(N)$, which is usually a significant reduction.

Since threads are distributed in a checkerboard-like fashion, only one thread needs to load an image patch and all neighboring threads can simply access the patch data from shared cache. This results in better cache efficiency.

4.4.6 Piecewise Constant Sequential Importance Resampling

The `PPF` library also provides an implementation of a fast approximate SIR algorithm that uses a piecewise constant approximation of the likelihood function to estimate the posterior distribution faster. This pcSIR algorithm can offer significant speedups [1].

4.5 Conclusions

We presented two classical distributed resampling algorithms (i.e., RNA and RPA) and a novel `PPF` library that enables parallel particle filtering applications on commodity as well as on high-performance parallel computing systems. The library uses multi-level hybrid parallelism combining Open MPI with native Java threads. The `PPF` library reduces parallel runtimes of the RNA and RPA methods by integrating dynamic load balancing with thread balancing and also implements PF-specific algorithmic improvements such as domain decomposition, image patches, and piece-

wise constant sequential importance resampling (pcSIR). The `PPF` library renders using parallel computer systems easier for application developers by hiding the intricacies of parallel programming and providing a simple API to design parallel PF applications.

As a proof of concept for the recently developed Open MPI Java bindings, we created the `PPF` library to enable parallel computing in PF application. In the next chapter, we show the capability of the `PPF` library in a biological imaging application. There, PF is used to track sub-cellular objects in 2D images and videos. This example compares the performances of traditional DRAs with our newly introduced DRAs, which are all implemented as a part of the `PPF` library.

CHAPTER

FIVE

New and Improved Algorithms for Parallel Particle Filtering

[1]Traditional distributed resampling algorithms such as RPA and RNA enable parallel processing of particle filtering algorithms. This is accomplished by parallelizing the resampling step and introducing an additional information exchange phase. This information exchange step is important since both RNA and RPA cause some type of *load imbalance* amongst PEs. In RPA, a *particle imbalance* occurs, meaning that PEs generate uneven number of new particles in resampling phase. Since the overall runtime of a PF algorithm depends on the number of particles, this particle imbalance affects the parallel performance of RPA. On the other hand, in RNA, the load imbalance is due to *particle weight imbalance*. While the number of particles on each PE is fixed in RNA, the weights may differ a lot and PF accuracy may degrade greatly.

In this chapter, we discuss how these DRAs can be further improved by accounting different DLB schemes that deal with *particle imbalance* and *particle weight imbalance*. Our first effort leads to the *adaptive RNA* (ARNA), which improves the tracking accuracy and runtime performance of RNA. Further, we analyze different DLB schemes one can employ for particle

[1]This work has been done in collaboration with Dr. Ihor Smal who co-designed and co-developed DLB algorithms for RNA, RPA, and BEM.

balancing in RPA. Finally, we introduce a new DRA called Box Exchange Method (BEM) that establishes itself as a new class of DRA algorithms and provides favorable parallel PF performance over other DRAs. All these methods are implemented in our `PPF` library and compared with each other in an application where PF is used to track sub-cellular objects in bio-images.

5.1 Tracking sub-cellular objects

We demonstrate the capabilities of the `PPF` library and the implemented DRAs by using them to implement a PF application for tracking sub-cellular objects imaged by fluorescence microscopy [187, 188]. In this example from biological microscopy imaging, sub-cellular structures such as endosomes, vesicles, mitochondria, or viruses are fluorescently labeled and imaged over time with a microscope. Many biological studies start from analyzing the dynamics of those structures and extracting parameters that characterize their behavior, such as average velocity, instantaneous velocity, spatial distribution, motion correlations, etc. First, we describe the dynamics and appearance models implemented in the `PPF` library for this biological application, and then we explain the technical details of the experimental setup.

5.1.1 Dynamics Model

The motion of sub-cellular objects can be represented using a variety of motion models, ranging from random walks to constant-velocity models to more complex dynamics where switching between motion types occurs [56, 192].

Here, we use a near-constant-velocity model, which is frequently used in practice [57, 193]. 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.

5.1.2 Observation Model

Many sub-cellular objects are smaller than what can be resolved by a visible-light microscope, making them appear in a fluorescence image as diffraction-limited bright spots, where the intensity profile is given by the impulse-response function of the microscope, the so-called point-spread function (PSF) [56, 57, 189, 190].

In practice, the PSF of a fluorescence microscope is well approximated by a 2D Gaussian [194]. The object appearance in a 2D image is hence modeled as:

$$I(x, y; x_0, y_0) = I_0 \exp\left(-\frac{(x - x_0)^2 + (y - y_0)^2}{2\sigma_{\text{PSF}}^2}\right) + I_{\text{bg}}, \tag{5.1}$$

where (x_0, y_0) is the position of the object, I_0 is its intensity, I_{bg} is the background intensity, and σ_{PSF} is the standard deviation of the Gaussian PSF. Typical microscope cameras yield images with pixel edge lengths corresponding to 60 to 200 nm physical length in the specimen. For the images used here, the pixel size is 67 nm, and the microscope has $\sigma_{\text{PSF}} = 78$ nm (or 1.16 pixels). During image acquisition, the "ideal" intensity profile $I(x, y)$ is corrupted by measurement noise, which in the case of fluorescence microscopy has mixed Gaussian-Poisson statistics. For the resulting noisy image $\mathbf{y}_t = Y_t(x, y)$ at time point t, the likelihood $\text{p}(\mathbf{y}_t|\mathbf{x}_t)$ is:

$$\text{p}(\mathbf{y}_t|\mathbf{x}_t) \propto \exp\left(-\frac{1}{2\sigma_\xi^2} \sum_{(x_i, y_i) \in \mathcal{S}_\mathbf{x}} [Y_t(x_i, y_i) - I(x_i, y_i; \hat{x}, \hat{y})]^2\right), \tag{5.2}$$

where σ_ξ controls the peakiness of the likelihood, (x_i, y_i) are the integer coordinates of the pixels in the image, $(\hat{x}, \hat{y})$ are the spatial components of the state vector $\mathbf{x}_t$, and $\mathcal{S}_\mathbf{x}$ defines a small region in the image centered at the location specified by the state vector $\mathbf{x}_t$. Here, $\mathcal{S}_\mathbf{x} = [\hat{x} - 3\sigma_{\text{PSF}}, \hat{x} + 3\sigma_{\text{PSF}}] \times [\hat{y} - 3\sigma_{\text{PSF}}, \hat{y} + 3\sigma_{\text{PSF}}]$.

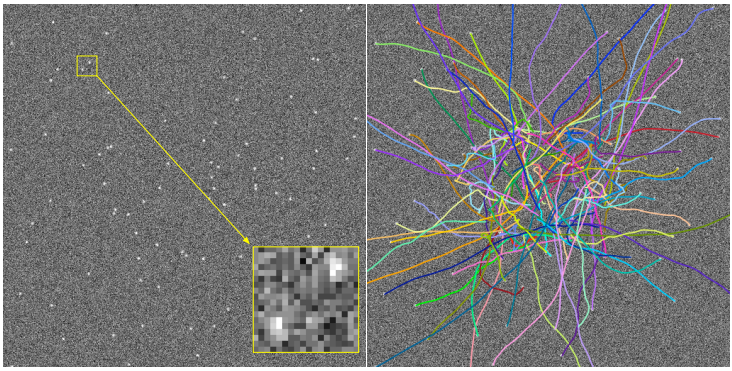


Figure 5.1: Examples of low SNR synthetic images used in the experiments. Left: The first 512×512 frame from a movie sequence of 50 frames, showing the typical object appearance due to the Gaussian PSF model. Right: Trajectories of the moving bright objects, generated using the nearly-constant-velocity dynamics model, overlaid with the first image frame.

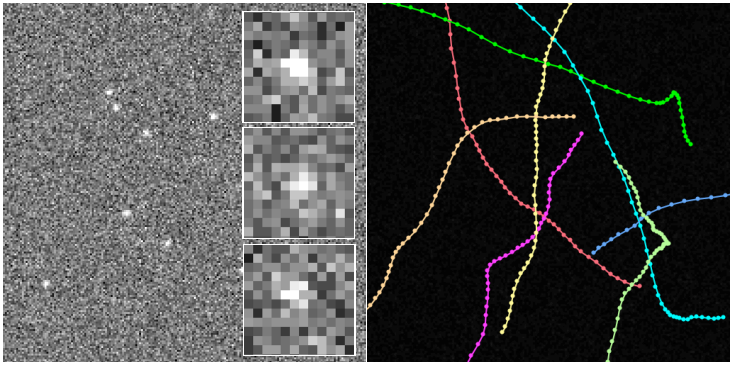


Figure 5.2: Another view at the examples of synthetic images used in the benchmarks. Left: One frame of a typical 2D image sequence with SNR=2, containing the small, bright objects of interest. Zoomed insets show noisy object appearance, modeled using a 2D Gaussian intensity profile corrupted with Poisson noise. Right: Typical object trajectories, generated according to the nearly-constant-velocity model.

5.1.2.1 Experimental Setup

We consider a single-object tracking problem where $\sigma_{\mathrm{PSF}} = 1.16$ pixels and the images have an SNR of 2 (equivalent to 6 dB). An example of an input image is shown in Fig. 5.1 (left). It is a synthetic image showing a number of bright PSF spots moving according to the above-described dynamics and appearance models. The task considered here for the PF is to detect the spots and track their motion over time, reconstructing their trajectories. The ground-truth trajectories used to generate the synthetic movies are shown in Fig. 5.1 (right). Comparing PF tracking results with them allows quantifying the tracking error. For other applications, the `PPF` library can easily be extended to include also other dynamics and observation models.

Using double-precision arithmetics, a single particle requires 52 B (i.e., six doubles and one integer) of computer memory. The particles are initialized uniformly at random and all tests are conducted with different random synthetic movies on the MadMax computer cluster of MPI-CBG, which consists of 44 nodes each having two 6-core Intel® Xeon® E5-2640 2.5 GHz CPU with 128 GB DDR3 800 MHz memory. All algorithms are implemented in Java (v. 1.7.0_13), and Open MPI (v. 1.9a1r28750) is used for inter-process communication.

We test RNA, ARNA, and RPA algorithms with different problem sizes, DLB strategies, and computer system sizes up to 384 cores. Next, we discuss improved versions of RNA and RPA.

5.2 Adaptive RNA

We propose the ARNA algorithm [3], which improves classical RNA by using dynamically adaptive particle-exchange ratios and randomized ring topologies. These two features help reducing *particle weight imbalance* among PEs faster and increase tracking efficiency of the parallel PF.

5.2.1 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:

$$N_{\text{ex}} = N_p \left[0.5 - \frac{0.5(\text{PE}_{\text{eff}} - 1)}{M - 1} \right] , \tag{5.3}$$

where N_p is the number of total particles on a PE. Hence, N_{ex} is negatively correlated with the tracking efficiency PE_{eff}, which is defined as:

$$\text{PE}_{\text{eff}} = \frac{\left(\sum_{m=1}^{M} \sum_{i=1}^{N} w_t^{(m,i)} \right)^2}{\sum_{m=1}^{M} \sum_{i=1}^{N} (w_t^{(m,i)})^2} , \tag{5.4}$$

where $w_t^{(m,i)}$ 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, where the PF is converged to its target and tracks it through input space. In such *tracking* case, all local PFs have converged to an equally good approximation of the posterior. The exchange of particles between PEs would then not be necessary anymore, but is still wastefully performed in RNA. It is also not clear what the optimal percentage of migrating particles should be. For applications requiring high precision, the number of particles that need to be communicated may also become too large, limiting the scalability of RNA.

5.2.2 Randomized Ring Topology

While the ring topology leads to a simple communication schedule in RNA, 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 degrades as the number of PEs increases [229].

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 [230] 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. ARNA only requires a few minor modifications to RNA in steps 1 and 2. A pseudocode for ARNA is given in Algorithm 11.

5.2.3 Benchmarks

We benchmark improvements of the proposed ARNA over RNA using an application from object tracking in fluorescence microscopy imaging [187, 231]. 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 [190], motion correlations, etc.

Algorithm 11 Adaptive RNA (ARNA)

1: Randomize the PE topology using Fisher-Yates shuffle [230]
2: Update the particle-exchange ratio N_{ex}/N_p according to Eq. 5.3. This requires a global communication in order to compute PE_{eff}.
3: Exchange N_{ex} of particles with neighboring PEs
4: Renormalize weights as $w_{t-1}^{(m,i)} = w_{t-1}^{(m,i)}/W_{t-1}$
5: Perform (P) and (U) steps of SIR to get $\mathbf{s}_t^{(m)}$
6: Compute the estimate $\hat{\mathbf{x}}_t^m$, and the sum of unnormalized weights $W_t^{(m)}$
7: Resample $\mathbf{s}_t^{(m)}$ using the locally normalized weights $\tilde{w}_t^{(m,i)} = w_t^{(m,i)}/W_t^{(m)}$
8: Set the i-th weight to $w_t^{(m,i)} = W_t^{(m)}$
9: Send $\hat{\mathbf{x}}_t^{(m)}$ and $W_t^{(m)}$ to the master PE
10: The master PE computes $\hat{\mathbf{x}}_t$ and W_t and broadcasts the result to all PEs

We use the same previous sequential implementation of SIR [56, 57] 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[56, 57]. 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 object tracking in fluorescence microscopy imaging is shown in Fig. 5.2.

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.

We compare ARNA against RNA with 0%, 10%, and 50% particle-exchange ratios. The memory footprint of a single particle is 52 B (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 `PPF` library [232]. We use Open MPI's Java bindings (v. 1.9a1r28750) for inter-process communication [223].

5.2.3.1 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 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. 5.3. 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.

5.2.3.2 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 5.4 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. A theoretical anal-

ysis of the tracking efficiency recovery is carried out in Appendix E.

5.3 DLB strategies for RPA

In RPA, after resampling every PE may generate uneven number of particles, which causes a particle imbalance. To rebalance the workload (i.e., number of particles) on each process, one has to use a DLB scheme. We discuss three DLB protocols, which all start by labeling the processes as either *senders* or *receivers*. A good DLB scheduler then minimizes the communication overhead required for routing particles from the *senders* to the *receivers*.

5.3.1 Greedy Scheduler

The Greedy Scheduler (GS) matches the first sender with the first receiver and then iterates through the senders. For each sender S_i with particle surplus N_{S_i}, it moves as many particles as possible to receiver R_j. Once a receiver is full, it moves on to the next R_{j+1} until the sender is empty. The procedure guarantees that at the end each process has the same number of particles. The pseudocode of GS is given in Algorithm 12.

5.3.2 Sorted Greedy Scheduler

The Sorted Greedy Scheduler (SGS) [94, 204, 233] first sorts the senders in S by their N_{S_i} and the receivers in R by their N_{R_j}, both in descending order. This sorting reduces the number of required communication links. The rest of the SGS algorithm is identical with GS, as seen in Algorithm 13.

5.3.3 Largest Gradient Scheduler

While GS and SGS aim to balance the loads perfectly, one may be interested in a faster scheduler that causes less communication overhead, but does not guarantee optimal particle balancing. Here, we introduce the

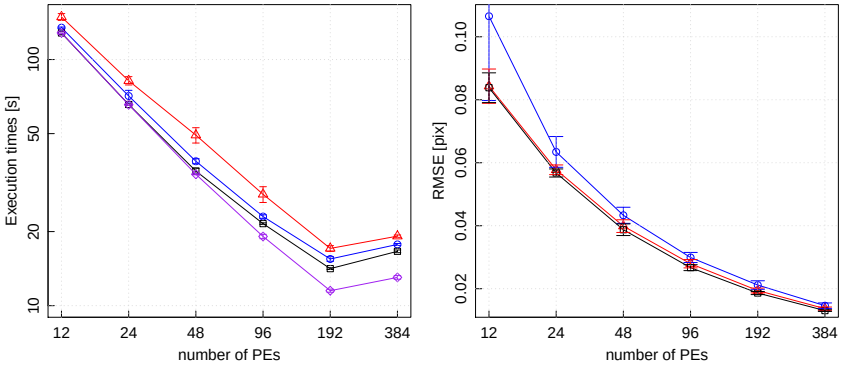


Figure 5.3: Left: Execution times of RNA with 50% exchange (red), 10% exchange (blue), and 0% exchange (purple) compared with the timings for ARNA (black). A fixed total number of 19.2 million particles is distributed over an increasing number of PEs (strong scaling). ARNA is faster than RNA with 10% 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.

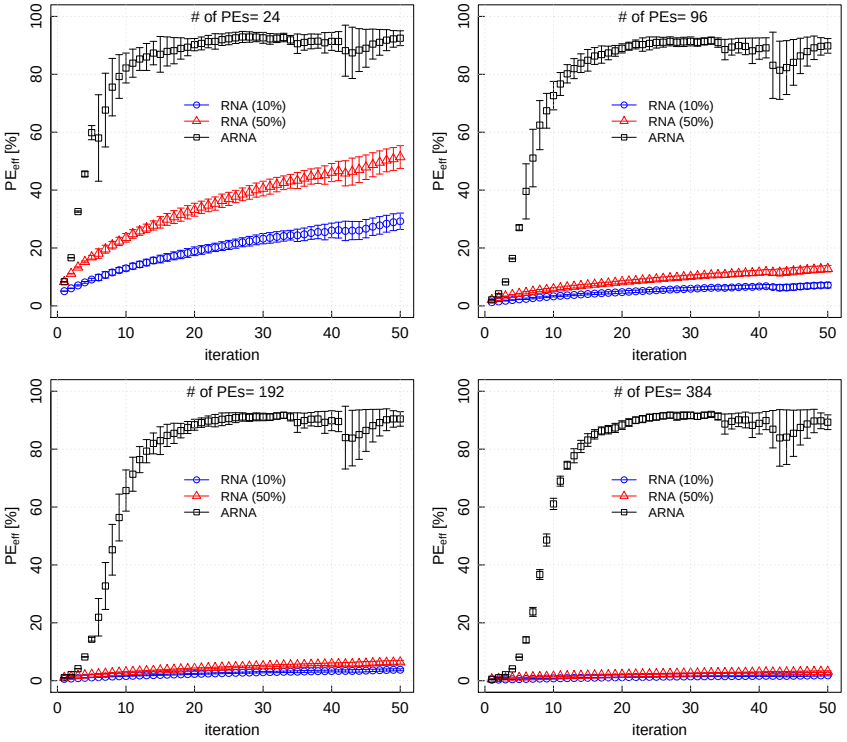


Figure 5.4: Percentage of PEs engaged in successful tracking of the target (PE_{eff}) as a function of iteration number during the information sharing phase: ARNA (black), RNA with 10% particle exchange (blue), and RNA with 50% particle exchange (red) on 24, 96, 192, and 384 PEs. The randomized ring topology of ARNA leads to a faster spread of information and hence a higher tracking efficiency. An upper

Algorithm 12 Greedy Scheduler

Require: S:=the list of senders, R:=the list of receivers.
Ensure: schedule:=the list of matchings between the elements of S and R including the number of particles to be routed.

```
1: procedure GreedyScheduler(S, R)
2:     j ← 0
3:     while S ≠ ∅ do
4:         while N_{S_i} ≠ 0 do
5:             if N_{S_i} ≥ N_{R_j} > 0 then
6:                 schedule ← {S_i, R_j, N_{R_j}}
7:                 N_{S_i} ← N_{S_i} − N_{R_j}
8:                 N_{R_j} ← 0
9:                 j ← j + 1
10:            else if N_{R_j} > N_{S_i} ≥ 0 then
11:                schedule ← {S_i, R_j, N_{S_i}}
12:                N_{R_j} ← N_{R_j} − N_{S_i}
13:                N_{S_i} ← 0
14:                j ← 0
15:            end if
16:        end while
17:        i ← i + 1
18:    end while
19:    return schedule
20: end procedure
```

Algorithm 13 Sorted Greedy Scheduler

Require: S:=the list of senders, R:=the list of receivers.
Ensure: schedule:=the list of matchings between the elements of S and R including the number of particles to be routed.

```
1: procedure SortedGreedyScheduler(S, R)
2:     S' ← sort(S)                          ▷ in descending order
3:     R' ← sort(R)                          ▷ in descending order
4:     return GreedyScheduler(S', R')
5: end procedure
```

Largest Gradient Scheduler (LGS), which is such a sub-optimal heuristic. Similar to SGS, LGS first sorts S and R such that $N_{S_1} > N_{S_2} > \ldots > N_{S_{|S|}}$ and $N_{R_1} > N_{R_2} > \ldots > N_{R_{|R|}}$. After that, each sender is paired with the corresponding receiver of same rank:

$$\begin{aligned} S_1 &\rightarrow R_1, \\ S_2 &\rightarrow R_2, \\ &\vdots \\ S_{\min(|S|,|R|)} &\rightarrow R_{\min(|S|,|R|)}. \end{aligned}$$

LGS thus finds the *largest gradients* between S and R and limits the number of communication links to

$$C = \min(|S|, |R|).$$

The pseudocode of LGS is given in Algorithm 14.

5.3.4 Results with RPA

For RPA, we compare three different DLB schemes. The tracking accuracy is measured by RMSE and was the same for all tests (about 0.063 pixels). All DLB schemes hence lead to results of equal quality. We use six Java threads per MPI process, since each CPU of the benchmark machine has six cores, and one MPI process per CPU. The wall-clock times are shown in Fig. 5.5 for a weak scaling with 60'000 particles per process. The corresponding parallel efficiency is shown in Fig. 5.6. Overall, LGS provides the best scalability, due to its linear communication complexity. Nevertheless, RPA scales less well than RNA and ARNA. For all RPA tests, we use a hybrid parallelism model defined in Fig. 4.5 (right) where each MPI process is assigned six Java Threads within the `PPF` library.

5.4 The Box Exchange Method

So far, all discussed DRAs exchange particles to sort out the *particle imbalance* or the *particle weight balance*. As the number of particles N increases,

Algorithm 14 Largest Gradient Scheduler

Require: S:=the list of senders, R:=the list of receivers.
Ensure: schedule:=the list of matchings between the elements of S and R including the number of particles to be routed.

1: **procedure** LargestGradientScheduler(S, R)
2: $S' \leftarrow \text{sort}(S)$ ▷ in descending order
3: $R' \leftarrow \text{sort}(R)$ ▷ in descending order
4: **for** $i = 1 \rightarrow \min(|S|, |R|)$ **do**
5: **if** $N_{S_i} \geq N_{R_i}$ **then**
6: $N_{S_i} \leftarrow N_{S_i} - N_{R_i}$
7: schedule $\leftarrow \{S_i, R_i, N_{R_i}\}$
8: **else**
9: $N_{S_i} \leftarrow 0$
10: schedule $\leftarrow \{S_i, R_i, N_{S_i}\}$
11: **end if**
12: **end for**
13: **return** schedule
14: **end procedure**

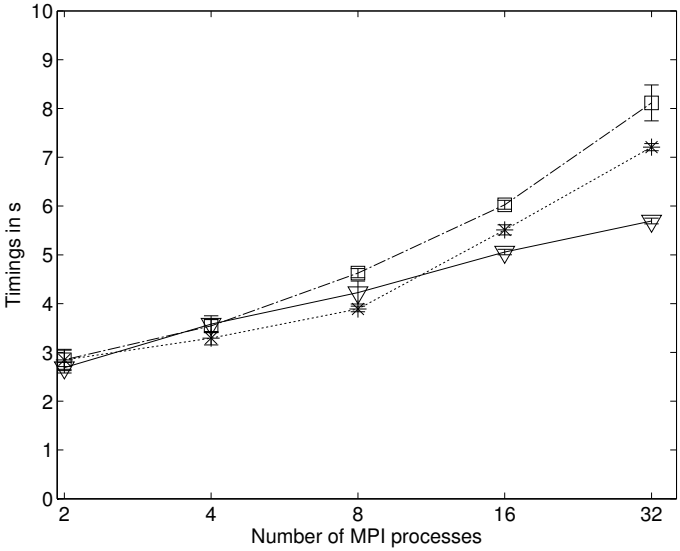


Figure 5.5: Weak scaling runtime results with less than a second standard deviation of a RPA run with 60K particles per MPI process. Each MPI processes is mapped onto a single CPU with six logical cores. There are six Java threads per MPI process. Three DLB schemes are used: Greedy ($\square$), SortedGreedy ($\star$), and LGS ($\triangledown$).

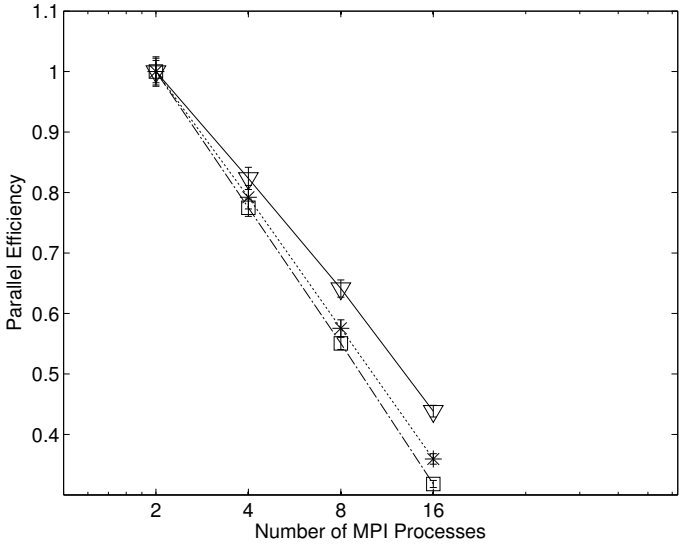


Figure 5.6: Strong-scaling parallel efficiencies of RPA with a constant number of 3.84 million particles distributed across an increasing number of processes. Three different DLB schemes are compared: Greedy ($\square$), SortedGreedy ($\star$), and LGS ($\triangledown$). Each MPI process is pinned to one of the two available CPUs on each node, each running six Java threads.

these DRAs become slower since the communication volume increases proportionally with N. This may limit the scalability of DRAs on large scale problems. Here, we introduce a novel particle exchange method to balance *particle weights* among PEs.

To obtain a better scalable parallel PF algorithm, one has to tackle the problem where N_{ex} scales up with the total number of particles N. In the Box Exchange Method (BEM), the image domain is divided into smaller d-dimensional subdomains where d is the dimensionality of the state space. In each small subdomain (i.e., box) a number of particles is contained. Similar to RNA, the user defines N_{ex} but instead of sending real particles, we send the d-dimensional boxes and the number of particles needed to be generated by the receiving PE in these boxes. The boxes are chosen smartly so that a minimum number of boxes are sent to match the required N_{ex}.

Each box has d dimensions where the first two or three dimensions depending on the input (i.e., 2D or 3D image) are the spatial information. In this study, we decompose the 2D or 3D image into smaller, non-overlapping boxes, which encompass some particles in themselves. For simplicity, we choose square boxes for 2D and cubes for 3D input data. For the remaining $d-2$ (or $d-3$) dimensions, the box acts like a bounding box where each particle in a box is visited and the minimum and maximum values for each dimension in that box is computed. Once the boxes are created, each PE starts preparing the message that needs to be sent to a neighboring PE. To further reduce the size of the message, we sort the boxes by the number of particles they contain and add the most populated boxes to the send message buffer. A schematic example of particle exchange in BEM is presented in Fig. 5.7.

The message buffer contains the information on the total number of boxes being sent and for each box: The ID of the box, the number of particles in that box, minimum and maximum values of $d-2$ (or $d-3$) dimensions and the average particle weight. So, assuming B boxes are being sent, the total message size for a 2D input is $B*(d+1)+1$, whereas in RNA it is significantly more: $N_{ex}*d$. By doing all of these, the message size is reduced from $\mathrm{O}(N)$ down to $\mathrm{O}(B)$, which is a significant algorithmic improvement in massively parallel PF applications. The algorithm for BEM

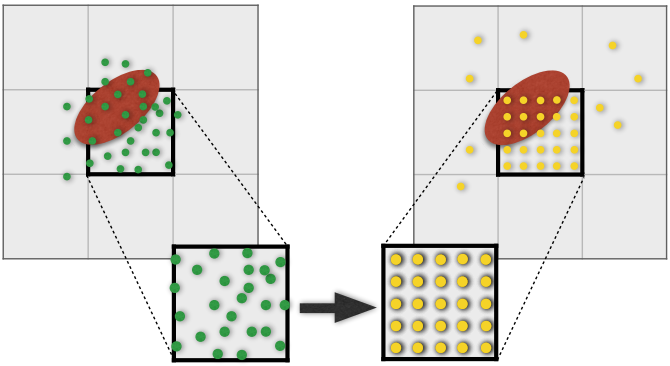


Figure 5.7: Instead of sending actual particles, PE (left) sends the dimensions and coordinates of the bounding box to the PE (right). Receiving PE resamples new particles (uniform) randomly in the box. Thus, the communication volume is greatly reduced.

is given in Algorithm 15.

5.4.1 Benchmarks

We compare BEM with RNA using the same synthetic images, dynamics model, and observation model described in Sec. 5.2.3. We use five different images and each method is run 30 times on each image using the `PPF` library. The RMSE values obtained by RNA and BEM are almost identical (below 0.1 pixels in all cases) and visually there is no difference in the results. BEM shows significant -up to 2.3x as seen in Fig. 5.8- performance enhancement over RNA and thus, it is a very good alternative to RNA for parallel PF.

5.5 Conclusions

In this chapter, we presented several algorithmic improvements over existing DRAs. ARNA adopts adaptive particle exchange ratio and randomizes inter-PE communication pattern to increase the tracking accuracy and to handle *particle weight imbalance* efficiently. In our tests, ARNA showed up to 20x better tracking efficiency and 9% faster execution over the traditional RNA.

On the RPA side, we discussed three dynamic load balancing algorithms for solving *particle imbalance* problem, which happens after resampling step. Largest Gradient Scheduler (LGS) outperforms other greedy DLB algorithms and provides a better scalable RPA. Nevertheless, RPA has a much lower parallel efficiency compared to RNA.

Furthermore, we introduced a novel method called Box Exchange Method (BEM), which offers a much faster way to balance the particle weights among PEs. The communication complexity of the RNA is reduced from $\mathrm{O}(N)$ down to $\mathrm{O}(B)$, where B is the number of boxes. By selecting each pixel as a box, we keep the tracking accuracy intact and accelerate the parallel performance more than 2x in a biological target tracking problem.

Algorithm 15 Box Exchange Method (BEM)

1: Decompose the image domain into B square boxes (done only once), b={1,...,B}
2: **for** each particle i **do**
3: Find out which box b they belong to, increase the number of particles N_b by 1
4: Update the total particle weight in b: $w_{t-1}^{(b)} = w_{t-1}^{(b)} + w_{t-1}^{(m,i)}$
5: **end for**
6: Sort all boxes by N_b in descending order as `sortedBoxes`
7: `nextBox` $\leftarrow 0$, `numBoxes` $\leftarrow 0$
8: **while** $N_{ex} > 0$ **do**
9: Compute the average particle weight in `sortedBoxes[nextBox]`
10: Add the `sortedBoxes[nextBox]`, number of particles in this box N_b, min and max values of $d-2$ (or $d-3$) dimensions and the average particle weight
11: `nextBox` $\leftarrow$ `nextBox` $+ 1$
12: $N_{ex} \leftarrow N_{ex} - N_b$
13: **end while**
14: Exchange boxes with neighboring PEs
15: Uniformly sample a total of N_{ex} particles in received boxes
16: Renormalize weights as $w_{t-1}^{(m,i)} = w_{t-1}^{(m,i)} / W_{t-1}$
17: Perform (P) and (U) steps of SIR to get $\mathbf{s}_t^{(m)}$
18: Compute the estimate $\hat{\mathbf{x}}_t^{(m)}$ and the sum of unnormalized weights $W_t^{(m)}$
19: Resample $\mathbf{s}_t^{(m)}$ using the locally normalized weights $\tilde{w}_t^{(m,i)} = w_t^{(m,i)} / W_t^{(m)}$
20: Set the i-th weight to $w_t^{(m,i)} = W_t^{(m)}$
21: Send $\hat{\mathbf{x}}_t^{(m)}$ and $W_t^{(m)}$ to the master PE
22: The master PE computes $\hat{\mathbf{x}}_t$ and W_t and broadcasts the result to all PEs

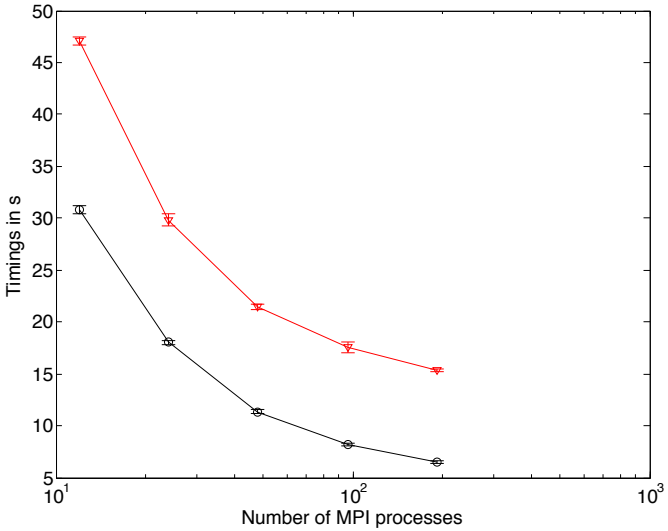


Figure 5.8: Runtime comparison of RNA (red) and BEM (black) both with 10% particle exchange ratio is shown. BEM outperforms RNA by up to 2.3x on a 192-PE simulation setup with 5.76 million particles while the tracking accuracy remains intact.

CHAPTER

SIX

Conclusions

Distributed DLB in massively parallel scientific numerical simulations We highlighted the importance of distributed DLB algorithms for parallel numerical simulations on petascale machines and beyond. We briefly described how load imbalance situations in a numerical simulation based on a domain decomposition may occur during the course of a simulation. We argued that for efficient solutions to such load imbalance problems one can employ distributed DLB protocols by keeping the original inter-process communication topology intact.

Following that, we showed a theoretical analysis of several distributed DLB protocols for *indivisible, real-valued* loads. We argued why these *fixed loads* represent the load imbalance situation in parallel simulations better. We formulated and compared three distributed DLB protocols, namely `Greedy`, `SortedGreedy`, and `Gradient` based on their merits to balance loads in a Balanced Circuit Model (BCM). After this assessment, we tested `SortedGreedy` and `Gradient` in realistic simulation test cases. We considered three prototypical IPC topologies as found in domain-decomposition numerical simulations of two kinds (linear flow and circular shock-wave propagation). We analyzed simulations running on more than one million PEs. Based on our findings, we developed a `HybridBalancer`, which is a good DLB candidate that can reduce initial load imbalance up to three-fold in a few DLB rounds requiring only local communication among neighbor-

ing PEs in a setup with more than a million PEs.

We did not consider the specifics of the computer system and implementation of the parallel application. Our results are therefore general. We also did not distinguish between communication latency and bandwidth, but simply assigned a unit communication cost to each load transfer.

The described DLB methods are designed to be employed on large scale systems with more than several thousand PEs. On smaller systems many communication libraries (e.g., Open MPI) provide fast execution of global communications. Moreover, by using a centralized DLB scheme, one can compute a new load assignment very quickly. This eliminates the need for distributed DLB protocols for simulations on small and mid-sized systems.

Piecewise constant sequential importance resampling (pcSIR) We proposed a fast approximate PF algorithm called pcSIR. pcSIR is based on spatially binning particles in *cells* and representing each cell by a single dummy particle at the center of mass of the cell's particle distribution, carrying the average state vector of all particles in that cell. With this zeroth-order approximation, pcSIR significantly reduces the computational cost of SIR and enables tackling larger problems, as well as mid-sized problems in real time. We discussed two variants, pcSIR-1x1 and pcSIR-2x2, which showed speedups up to several orders of magnitude while keeping the tracking accuracy on par with traditional SIR in synthetic test cases mimicking sub-cellular object tracking in bio-imaging. Both pcSIR variants help reduce the execution time of an application a lot if the likelihood update step is very costly.

In addition to the empirical tests, we performed both theoretical and experimental error analysis of pcSIR. We showed that the convergence rate of pcSIR is the same as SIR, since the Monte-Carlo sampling error dominates the general error term in the application and the error from the function approximation does not play a big role. Moreover, we presented theoretical upper bounds on the likelihood approximation error depending on the cell size.

The pcSIR methods were described with a specific focus on image-based applications from biology. If the input data is not an image, one cannot use

pcSIR directly, since it uses (sub-)pixels as cells for likelihood approximation. In that case, an equivalent strategy (e.g., creating cells in the state space) needs to be considered. Further, if an application is not likelihood-intensive, meaning that likelihood computations does not constitute the bottleneck of the application, using pcSIR may not result in large speedups as presented in this thesis. However, typical target tracking applications benefit from the approximation approach presented in this thesis.

We believe that pcSIR can be used in many PF applications that require large numbers of particles, costly likelihood evaluations, or real-time performance. When tracking accuracy is not critical, pcSIR-1x1 can offer orders of magnitude speedup in image-processing applications. If a loss in tracking accuracy is undesired, pcSIR-2x2 still offers significant speedups while in some cases even improving accuracy over SIR. Depending on the application in hand, one may need to try both pcSIR variants and then pick one that is best suited to the needs.

Improved distributed resampling algorithms (DRAs) We revisited existing DRAs and investigated how DLB can be used to improve these algorithms. For RPA, we designed and compared several DLB algorithms to tackle *particle imbalance* problem among PEs. Our *largest gradient scheduler* outperformed greedy DLB approaches. On the RNA side, we introduced an adaptive particle exchange ratio and randomized inter-PE communication patterns to tune the communicated particle data volume depending on the tracking efficiency. These improvements gave ARNA up to 20x better tracking efficiency and 9% faster execution over the traditional RNA while also improving the tracking accuracy.

Despite the fact that ARNA randomizes inter-PE communication in every iteration, it still uses a ring topology regardless of how the PEs are physically connected to each other. ARNA can be further advanced by knowing the hardware specifications of the computer cluster and designing a hardware-aware communication topology from the start. In line with the *implementation engineering* principle discussed in the introduction, having hardware-optimized ARNA would perform much better on a cluster.

Fast parallel particle filtering via a Box Exchange Method (BEM)

We introduced a novel parallel particle filtering algorithm called BEM, which offers a faster way to balance the particle weights among PEs. The communication complexity of the original RNA is reduced from O(N) to O(B), where N is the number of particles and B is the number of boxes. By selecting each pixel as a box, we keep the tracking accuracy intact and increased the parallel performance more than 2x in a biological target tracking problem.

Similar to ARNA, the BEM method also uses a ring topology and may benefit as well from a better physical network-matching communication topology, which would reduce the time to needed to communicate data with neighboring PEs. Also, more advanced methods for defining boxes can be researched and further computational improvements may be achieved.

The Parallel Particle Filtering (`PPF`) library We presented the `PPF` library that enables parallel particle filtering applications on commodity and on high-end parallel computing systems. The library uses multi-level hybrid parallelism combining Open MPI with native Java threads. The `PPF` library reduces parallel runtimes of all DRAs described in this thesis by integrating dynamic load balancing with thread balancing, and it also implements PF-specific algorithmic improvements such as domain decomposition, image patches, and piecewise constant sequential importance resampling (pcSIR). The `PPF` library hides the intricacies of parallel programming by providing a simple API to design parallel PF applications.

The library currently supports 2D image sequences fully and requires an additional effort to also process 3D image sequences (i.e., 4D movies). Additionally, the library cannot optimally track multiple targets all at the same time since particle filters are designed to optimally track only a single target [234].

CHAPTER

SEVEN

Outlook

We present ongoing work and outline future directions of the research topics discussed in this thesis.

DLB for domain-decomposition-based parallel numerical simulations Future work is concerned with the development of a parametrizable DLB performance virtualization platform based on performance models, such as LogGP [235], whose parameters can be tuned to reflect the specifics of the hardware, communication costs, number of DLB rounds, etc. Such a platform would enable computational scientists to model the expected performance of their parallel numerical simulations, and to optimize DLB schedules in advance. Moreover, the theoretical analysis of the presented DLB algorithms can be extended to heterogeneous systems with different clock rates and other hardware specifications. In addition to theoretical work, the performance of `HybridBalancer` should be verified in real-world applications. This work would see `HybridBalancer` implemented in the PPM library and tested

Parallel Particle Filtering (`PPF`) library Java bindings of Open MPI are relatively new and there is an ongoing project to adopt the new MPI-3 standard in Java bindings as well. MPI-3 offers new features such as sparse collectives and non-blocking collectives, which can be investigated how they can be incorporated into `PPF`. Additionally, GPUs are common

in laptop computers as well as in HPC clusters. We would like to extend the PPF library to take advantage of GPUs and/or other accelerators, and thus enable even faster parallel execution. Further, the library is currently a stand-alone software package. Next, we would like to integrate it into Fiji [217] and ImageJ [216]. This would encourage Fiji/ImageJ users to benefit from high-performance computing in their research. Since the library is written in Java and already interfaced to both libraries, converting the PPF library into a Fiji/ImageJ plug-in would be rather straightforward.

A further task would be extending and generalizing the PPF library such that it can be used as a unified framework for a large class of Monte Carlo simulations. PFs constitute only a single branch of Monte Carlo methods. However, PFs use particles as the main data structure, which is common to all Monte Carlo methods. By redefining some classes and reimplementing some modules, the PPF library could be extended to a unifying framework also for other types of Monte Carlo simulations, e.g., particle swarm optimization algorithms [236]. This would broaden the scope of scientists, who may use the PPF library to execute their codes on big computing systems and obtain results quicker.

The task of object tracking is further complicated if there are multiple objects interacting with each other and/or forming clutters. Current PF algorithms cannot solve tracking problem optimally for all targets since PF is not designed to handle such cases [234]. Revisiting the general PF algorithm and designing several algorithmic and theoretical improvements that would allow the PPF library resolve multi-target tracking problems at occlusions and clutters. This work would entail solving ambiguities outside the PF framework and then incorporating the findings back to the PF framework. With these enchancements the PPF library would enable high-performance multi-target tracking.

APPENDIX

A

Bounds for balancing different types of loads in arbitrary networks

A.0.1 Notation

Let $G = (V, E)$ be an undirected and connected IPC graph consisting of n vertices V. Following established notation [86, 93], we denote an edge $[u : v]$, where $\{u, v\} \in E$ with $u < v$. Each *matching* in round t of a BCM is represented by an $n \times n$ matrix $\mathbf{M}^{(t)} \succeq 0$. Moreover, if $[u : v] \in \mathbf{M}^{(t)}$, then $\mathbf{M}^{(t)}_{u,u} = \mathbf{M}^{(t)}_{v,v} = \mathbf{M}^{(t)}_{u,v} = \mathbf{M}^{(t)}_{v,u} = 1/2$. If u is not matched in round t, then $\mathbf{M}^{(t)}_{u,u} = 1$ and $\mathbf{M}^{(t)}_{u,v} = 0$ for all $v \neq u$. In BCM, two matched nodes u and v try to balance their loads as evenly as possible in round t. L is the total number of loads or subdomain costs.

A.0.2 Balancing Circuit Model

In BCM, a pre-determined sequence of d matchings $\mathbf{M}^{(1)}, \ldots, \mathbf{M}^{(d)}$ is sequentially applied such that all edges in the graph are visited at least once. The resulting *round matrix* $\mathbf{M}$ [86] is defined as $\mathbf{M} := \prod_{s=1}^{d} \mathbf{M}^{(s)}$. The n eigenvalues of $\mathbf{M}$ are denoted $\lambda_1(\mathbf{M}) \geq ... \geq \lambda_n(\mathbf{M})$. Moreover, $\lambda(\mathbf{M}) := \max\{|\lambda_2(\mathbf{M})|, |\lambda_n(\mathbf{M})|\}$. We denote the product of a sequence of matching matrices between rounds t_1 and t_2 by $\mathbf{M}^{[t_1,t_2]} := \prod_{s=t_1}^{t_2} \mathbf{M}^{(s)}$,

where $t_2 \geq t_1$. We also require the Markov chain with transition matrix $\mathbf{M}$ to be ergodic, i.e., $\lambda(\mathbf{M}) < 1$. Matching matrix sequences that satisfy this condition can be obtained by an edge coloring algorithm [237, 238], which also provides the optimal communication schedule across PEs. The results we show here for BCM can be extended to the *random matching model*, where the matching matrices are realizations of a stochastic process.

A.0.3 Bounds for Balancing Different Types of Loads in Arbitrary Networks

We use the theoretical analysis introduced by Sauerwald and Sun [93] to show that similar asymptotic bounds can be derived for the *invidisible, real-valued* load model. In the continuous case, where loads are arbitrarily divisible, the number of rounds needed by a BCM to balance the load in an arbitrary graph with a discrepancy of ϵ is less than or equal to $\frac{4d}{1-\lambda(\mathbf{M})} \log\left(\frac{Kn}{\epsilon}\right)$, where K is the discrepancy in the initial load assignment and d is the number of matchings $\mathbf{M}^{(1)}, \ldots, \mathbf{M}^{(d)}$ ([86], Theorem 1; [93], Theorem 2.2).

If the loads are indivisible, unit-sized tokens, the discrepancy cannot be made arbitrarily small. Using BCM on an arbitrary graph, a discrepancy of $\sqrt{12 \log n} + 1$ is reached after $\mathrm{O}\left(d \cdot \frac{\log(Kn)}{1-\lambda(\mathbf{M})}\right)$ rounds with probability at least $1 - 2n^{-2}$ ([93], Theorem 2.14). In the present case, each load is defined by a constant real number. Loads cannot be modified or subdivided, but only moved from one PE to another.

A.0.3.1 Continuous Case

Let ξ denote the load vector and $\xi^{(0)}$ be initial load vector at time $t = 0$. In this setting, loads can be divided arbitrarily. Thus, a perfect balance between matching nodes u and v can be established in each round t using the corresponding matching matrix $\mathbf{M}^{(t)}_{u,v}$. As shown in [93], the evolution of the load vector is a linear system and can be written as $\xi^{(t)} = \xi^{(t-1)}\mathbf{M}^{(t)}$.

Further, the evolution of the loads on node u can be formulated as follows:

$$\xi_u^{(t)} = \xi_u^{(t-1)} + \sum_{v:\{u,v\}\in E} \left(\xi_v^{(t-1)}\mathbf{M}_{v,u}^{(t)} - \xi_u^{(t-1)}\mathbf{M}_{u,v}^{(t)}\right) \quad (A.1)$$

$$= \xi_u^{(t-1)} + \sum_{v:\{u,v\}\in \mathbf{M}^{(t)}} \left(\frac{1}{2}\xi_v^{(t-1)} - \frac{1}{2}\xi_u^{(t-1)}\right). \quad (A.2)$$

The evolution of the load vector is a Markov chain process and its convergence speed is closely related to its spectral discrepancy $(1 - \lambda(\mathbf{M}))$.

Theorem 1 *([86], Theorem 1; [93], Theorem 2.2) Let G be any graph. Consider the balancing circuit model with d matchings $\mathbf{M}^{(1)},\ldots,\mathbf{M}^{(d)}$. Then, for any $\epsilon > 0$, $\tau_{cont}(K,\epsilon) \leq d \cdot \frac{4}{1-\lambda(\mathbf{M})} \cdot \log\left(\frac{Kn}{\epsilon}\right)$ where $\tau_{cont}(K,\epsilon)$ is the minimum number of rounds in the continuous case to reach a discrepancy ϵ for any initial load vector $\xi^{(0)}$ with discrepancy at most K ([93], Definition 2.1).*

A.0.3.2 Discrete Case

In case of loads being indivisible, unit-size tokens the evolution of the load vector includes an error term $e_{u,v}^{(t)}$ for each matching $\mathbf{M}^{(t)}$ in round t. The error stems from the rounding of the real number down to the nearest integer and can be written as follows:

$$e_{u,v}^{(t)} := \frac{1}{2}\texttt{Odd}(x_u^{(t-1)} + x_v^{(t-1)}) \cdot \Phi_{u,v}^{(t)}, \quad (A.3)$$

where $\texttt{Odd}(x) := x \mod 2$ and $\Phi_{u,v}^{(t)} = 1$, if the excess token is given to u. If it is assigned to v, then $\Phi_{u,v}^{(t)} = -1$. Thus, the true load vector $x^{(t)}$ in round t can be formulated as follows:

$$x^{(t)} = x^{(t-1)}\mathbf{M}^{(t)} + e^{(t)}. \quad (A.4)$$

After solving this recursion (cf. [86]) and using the same notation as in [93], we get:

$$x^{(t)} = x^{(0)}\mathbf{M}^{(t)} + \sum_{s=1}^{t} e^{(s)}\mathbf{M}^{[s+1,t]} = \xi^{(t)} + \sum_{s=1}^{t} e^{(s)}\mathbf{M}^{[s+1,t]}, \quad (A.5)$$

where $\xi^{(t)}$ is the load vector in continuous case and initially $\xi^{(0)} = x^{(0)}$. The tight bounds on the discrepancy between the *discrete* and *continuous* cases can be found in [93].

Theorem 2 *([93], Theorem 2.14) Let G be any graph. Then, the following statements hold:*

- *Let $\mathcal{M} = \langle \mathbf{M}^{(1)}, \mathbf{M}^{(2)}, .. \rangle$ be any sequence of matchings. If $x^{(0)} = \xi^{(0)}$, then for any round t and any $\delta \geq 1$, it holds that*

$$\Pr\left[\max_{w \in V} \left| x_w^{(t)} - \xi_w^{(t)} \right| \geq \sqrt{4\delta \cdot \log n}\right] \leq 2n^{-\delta+1}. \tag{A.6}$$

- *Using the balancing circuit model, a discrepancy of $\sqrt{12 \log n} + 1$ is reached after $\tau_{cont}(K, 1) = \mathrm{O}\left(d \cdot \frac{\log(Kn}{1-\lambda(\mathbf{M})}\right)$ with probability at least $1 - 2n^{-2}$.*

A.0.3.3 Fixed-load Case

In *fixed-load model*, each load is defined by a constant real number, i.e., different loads can have different sizes and there is no unit token or quantum, but loads are indivisible and retain their size throughout the entire DLB process. We show the relation between the present case and the continuous case by using a slightly modified version of the theorems from Ref. [93]. In order for the analysis to be valid, all of the following have to be satisfied[1]:

1. The maximum load is non-increasing and the minimum load is non-decreasing during the entire DLB process.

2. The load difference between two nodes is minimized as much as possible in each matching.

3. The expected error $\mathbb{E}[e_{u,v}]$ on every matching edge $[u : v] \in \mathbf{M}^{(t)}$ is zero.

4. Lemma 2.12 of Ref. [93] holds:

[1] Personal communication with Dr. Thomas Sauerwald

Lemma 1 ([93], Lemma 2.12) *Fix two rounds $t_1 < t_2$ and the load vector $x^{(t_1)}$ at the end of round t_1. For any family of non-negative numbers $g_{u,v}^{(s)}$ ($[u:v] \in \mathbf{M}^{(s)}$, $t_1 + 1 \leq s \leq t_2$), define the random variable Z by $Z := \sum_{s=t_1+1}^{t_2} \sum_{[u:v]\in\mathbf{M}^{(s)}} g_{u,v}^{(s)} \cdot e_{u,v}^{(s)}$. Then, $\mathbb{E}[Z] = 0$ and for any $\delta > 0$ it holds that*

$$\Pr\left[|Z - \mathbb{E}[Z]| \geq \delta\right] \leq 2 \cdot \exp\left(\frac{\delta^2}{2\sum_{s=t_1+1}^{t_2} \sum_{[u:v]\in\mathbf{M}^{(s)}} \left(g_{u,v}^{(s)}\right)^2}\right). \tag{A.7}$$

This lemma requires condition (3): $\mathbb{E}\left[e_{u,v}^{(s)}\right] = 0, \forall\{u,v\} \in \mathbf{M}^{(s)}$, which ensures $\mathbb{E}[Z] = 0$.

Under these conditions, the upper bound on the discrepancy that can be reached by a BCM with fixed, real-valued loads is the same as the upper bound already derived for indivisible, unit-sized tokens (Theorem 2.14 in Ref. [93]):

Theorem 3 *([93], Theorem 2.14) Let G be any graph. Then, the following statements hold:*

- *Let $\mathcal{M} = \langle \mathbf{M}^{(1)}, \mathbf{M}^{(2)}, \ldots \rangle$ be any sequence of matchings in a BCM. If $x^{(0)} = \xi^{(0)}$, then for any round t and any $\delta \geq 1$*

$$\Pr\left[\max_{w\in V} \left|x_w^{(t)} - \xi_w^{(t)}\right| \geq \sqrt{4\delta \cdot \log n}\right] \leq 2n^{-\delta+1}. \tag{A.8}$$

- *Using BCM, a discrepancy of $\sqrt{12\log n}+1$ is reached after $\tau_{cont}(K,1) \in \mathrm{O}\left(d \cdot \frac{\log(Kn}{1-\lambda(\mathbf{M})}\right)$ with probability at least $1 - 2n^{-2}$.*

We prove the expected performance of a `SortedGreedy`-based DLB algorithm working on indivisible real-valued loads under the above conditions. We need the following lemmata:

Lemma 2 *The error e_c in every matching $[u:v]$ is always zero in the continuous case.*

Proof. Let ξ_u and ξ_v be the local load vectors on u and v, respectively. The evolution of the load vector is a linear system and can be written as $\xi^{(t)} = \xi^{(t-1)}\mathbf{M}^{(t)}$. Further, the evolution of the loads on node u can be formulated as:

$$\xi_u^{(t)} = \xi_u^{(t-1)} + \sum_{v:\{u,v\}\in E} \left(\xi_v^{(t-1)}\mathbf{M}_{v,u}^{(t)} - \xi_u^{(t-1)}\mathbf{M}_{u,v}^{(t)}\right) \quad \text{(A.9)}$$

$$= \xi_u^{(t-1)} + \sum_{v:\{u,v\}\in \mathbf{M}^{(t)}} \left(\frac{1}{2}\xi_v^{(t-1)} - \frac{1}{2}\xi_u^{(t-1)}\right). \quad \text{(A.10)}$$

The evolution of the load vector is a Markov chain and its convergence speed is closely related to its spectral gap $(1 - \lambda(\mathbf{M}))$. In the continuous case after a matching $[u : v]$ both ξ_u and ξ_v will be the same. Since $e_c = |\xi_u - \xi_v| = 0$, we will always have a perfectly balanced state after each matching. ■

Lemma 3 *Let e_f denote the load imbalance in the fixed load case after balancing local loads ξ_u and ξ_v on nodes u and v, respectively. The difference d between e_f and e_c after balancing a matched edge equals to e_f.*

Proof. From Lemma 2 we have $e_c = 0$ for every matching, hence $d = |e_f - e_c| = e_f$. ■

Lemma 4 *Let the load vector $l := \{l_1, l_2, \ldots, l_n\}$ with $l_1 \geq l_2 \geq, \ldots, \geq l_n$. The maximum difference $|d_{max}| := max(|e_f - e_c|)$ obtained by* `SortedGreedy` *is $|d_{max}| \leq \frac{l_1}{2}$.*

Proof. Consider the worst case where all loads are equal to each other, $l_1 = l_2 = \ldots = l_l = \mathcal{L}$. In this case, the minimum discrepancy achieved by `SortedGreedy` is maximized. This is due to the fact that all loads carry maximum possible weight compared to each other. The algorithm places the first load on processor A, which is chosen arbitrarily. The total weights of processors A and B hence are $\mathcal{L}$ and 0, respectively, for any B. The ideal load distribution would correspond to $\mathcal{L}/2$ on each processor. Thus, the discrepancy is $\mathcal{L}/2$ and it will remain at most $\mathcal{L}/2$ until all loads are placed. ■

Now, we prove that the present case and `SortedGreedy` fulfill all requirements stated above:

Proof of condition 1: By definition, the load weights do not change during the DLB process, since they are indivisible. Only their host PEs change. ■

Proof of condition 2: By construction, `SortedGreedy` balances the loads as evenly as possible. ■

Proof of condition 3: To show that $\mathbb{E}[e_{u,v}^{(t)}] = 0$, we can look at the two-bin case between u and v. Due to the symmetry $e_{u,v}^{(t)} = -e_{v,u}^{(t)}$, the expected error on an edge is always zero. ■

Proof of condition 4: We need to prove that Z is concentrated around its mean by applying an appropriate concentration inequality theorem. We closely follow the proof given in Ref. [93], but we have to adjust the concentration bounds for the error. In Ref. [93], unit loads are considered, hence $e_{u,v}^{(t)} \in \{-1/2, 0, 1/2\}$, and errors on different edges are independent of each other. In the present case of indivisible real-valued loads, $e_{u,v}^{(t)}$ is also independent of errors on other edges and, due to Lemma 4, $\{-\frac{l_{\max}}{2} \leq e_{u,v}^{(s)} \leq \frac{l_{\max}}{2}\}$, where $l_{\max}$ is the largest load in the entire network. In words, the maximum error on any edge is bounded by the largest load in the network. This enables us to use Lemma 2.13 from Ref. [93]:

Lemma 5 ([93], Lemma 2.13) *Fix an arbitrary load vector $x^{(0)}$. Consider two rounds $t_1 \leq t_2$ and assume that the time-interval $[0, t_1]$ is $(K, 1/(2n))$-smoothing. Then, for any node $w \in V$ and $\delta > 1/n$, it holds that*

$$\Pr\left[\left|x_w^{(t_1)} - \bar{x}\right| \geq \delta\right] \leq 2 \cdot \exp\left(-\left(\delta - \frac{1}{2n}\right)^2 / 4\right). \tag{A.11}$$

Using Lemmata 1, 4, and 5, and following the same derivation as in Ref. [93] (Lemmata 2.12 and 2.13 therein), it follows that Theorem 3 holds for a BCM with indivisible, real-valued loads. ■

APPENDIX

B

Analogy to Balls-into-bins Problem

[1]We analyze the expected decrease in discrepancy per iteration of a two-bin balls-into-bins model and derive a lower bound on the final discrepancy. Let $W_1 > W_2 > \ldots > W_m$ be the weights of the m balls, and S the set of all balls assigned to bin k. The total weight of bin k after assigning i balls then is

$$U_i^{(k)} := \sum_{j \in S} W_j \,. \tag{B.1}$$

We rewrite Eq. B.1 as:

$$U_i^{(k)} := i \cdot \bar{W}, \tag{B.2}$$

where $\bar{W}$ is the mean ball weight. Further, the discrepancy after placing i out of m balls is:

$$G_i = \max_k U_i^{(k)} - \min_k U_i^{(k)}. \tag{B.3}$$

We put the tag "heaviest" on the heaviest bin $U_{\texttt{heaviest}}$ and a "switch" happens if after assigning the next ball, another bin takes the tag "heaviest". If no switch occurs, the heaviest bin remains the same, but the discrepancy is reduced by the weight of the next ball W_{i+1}. In the "switch" case G_i can change at most by W_{i+1}.

We start our analysis after putting the first ball at random into either bin

[1]This analysis is developed with the help of Dr. Erdem Yörük.

$U^{(1)}$ or $U^{(2)}$. Without loss of generality, we assume that $U^{(1)}$ is heavier than $U^{(2)}$ after assigning the i^{th} ball. The discrepancy is $G_i = U_i^{\texttt{heaviest}} - U_i^{(2)}$. After assigning the next ball, the discrepancy becomes:

- "no switch": $G_{i+1} = U_{i+1}^{\texttt{heaviest}} - U_{i+1}^{(2)}$.
- "switch": $G_{i+1} = U_{i+1}^{\texttt{heaviest}} - U_{i+1}^{(1)}$.

We are interested in the decrease in discrepancy $\Delta G_{i+1} = G_i - G_{i+1}$ in both cases:

1) *"No-switch" case*: We put the next ball W_{i+1} in $U^{(2)}$. Thus, the total weight of $U^{(1)}$ does not change but the discrepancy is reduced by W_{i+1}:

$$\Delta G_{i+1} = W_{i+1}. \tag{B.4}$$

2) *"Switch" case*: Let us assume that $U_i^{(1)}$ has J balls in it, whereas $U_i^{(2)}$ contains K balls such that $J + K = i$. We put the next ball W_{i+1} in $U^{(2)}$, which newly becomes the heavier bin. Now $U^{(2)}$ contains $K + 1$ balls. The total weight of $U^{(1)}$ did not change, $U_i^{(1)} = U_{i+1}^{(1)}$ and the discrepancy difference is:

$$\begin{aligned}
\Delta G_{i+1} &= U_i^{(1)} - U_i^{(2)} - \left| U_{i+1}^{(1)} - U_{i+1}^{(2)} \right| && \text{(B.5)} \\
&= U_i^{(1)} - U_i^{(2)} + U_{i+1}^{(1)} - U_{i+1}^{(2)} \\
&= 2 \cdot U_i^{(1)} - U_i^{(2)} - U_{i+1}^{(2)} \\
&= 2 \cdot U_i^{(1)} - 2 \cdot U_i^{(2)} - W_{i+1} \\
&\leq W_{i+1}. && \text{(B.6)}
\end{aligned}$$

In terms of the average ball weight, this is:

$$\begin{aligned}
\Delta G_{i+1} &= 2 \cdot U_i^{(1)} - U_i^{(2)} - U_{i+1}^{(2)} \\
&\simeq 2J \cdot \bar{W} - K \cdot \bar{W} - (K+1) \cdot \bar{W} \\
&\simeq (2J - 2K + 1) \cdot \bar{W}. && \text{(B.7)}
\end{aligned}$$

If the ball weights are sampled from a uniform distribution, and m is large

enough (s.t. $J \approx K$), we have:

$$\Delta G_{i+1} \simeq \bar{W} \leq W_{i+1}. \tag{B.8}$$

For other distributions, the relationship between J and K depends on the standard deviation of $\mathcal{D}$. ■

B.0.4 n-BIN CASE

The extension of two-bin problem to n-bin problem is straightforward. We add an additional tag "lightest." In two-bin case, the bin with the "lightest" tag is trivial and the tag is not used. Yet, here we take advantage of having this second tag. One important fact to consider is the existence of other *intermediate* bins whose total weights lie between the heaviest and lightest bin. The "switch" and "no-switch" of the "heaviest" tag can be written as follows:

1) *"No-switch" case*: We put the next ball W_{i+1} into the lightest bin $U_i^{\texttt{lightest}}$. Since we have n bins, an intermediate bin might become the lightest bin after $i+1$ balls or $U_i^{\texttt{lightest}}$ is increased by W_{i+1}. Nevertheless, regardless of the value of W_{i+1} it holds that $U_{i+1}^{\texttt{lightest}} > U_i^{\texttt{lightest}}$. On the other hand, $U_i^{\texttt{heaviest}} = U_{i+1}^{\texttt{heaviest}}$ since a "switch" does not occur. Thus, the discrepancy difference is written as follows:

$$\begin{aligned}
\Delta G_{i+1} &= U_i^{\texttt{heaviest}} - U_i^{\texttt{lightest}} - (U_{i+1}^{\texttt{heaviest}} - U_{i+1}^{\texttt{lightest}}) \\
&= U_{i+1}^{\texttt{lightest}} - U_i^{\texttt{lightest}} \\
&\leq W_{i+1},
\end{aligned} \tag{B.9}$$

since the maximum ΔG is achieved only if the previous lightest bin gets W_{i+1} and is still the lightest. For large enough i, we can reformulate equation B.9 by introducing a statistical upper bound on the discrepancy difference ΔG_{i+1}

$$\begin{aligned}\Delta G_{i+1} &= U_{i+1}^{\texttt{lightest}} - U_i^{\texttt{lightest}} \\ &\simeq (K+1)\cdot \bar{W} - K \cdot \bar{W} \\ &\simeq \bar{W}, \end{aligned} \tag{B.10}$$

where $K < i$ is the number of balls in $U_i^{\texttt{lightest}}$. We do a similar combination as in the two-bin case and obtain using equations B.9 and B.10:

$$\Delta G_{i+1} \simeq \bar{W} \leq W_{i+1}. \tag{B.11}$$

2) *"Switch" case*: Switching the "heavier" tag to another bin states that

$$U_{i+1}^{\texttt{heaviest}} > U_i^{\texttt{heaviest}},$$

and

$$U_{i+1}^{\texttt{heaviest}} = U_i^{\texttt{lightest}} + W_{i+1}. \tag{B.12}$$

Moreover, a previously intermediate bin becomes the lightest bin:

$$U_{i+1}^{\texttt{lightest}} \geq U_i^{\texttt{lightest}}.$$

Yet, the relation is now

$$U_{i+1}^{\texttt{lightest}} \neq U_i^{\texttt{lightest}} + W_{i+1}.$$

Hence, the discrepancy difference is:

$$\begin{aligned}\Delta G_{i+1} &= U_i^{\texttt{heaviest}} - U_i^{\texttt{lightest}} - (U_{i+1}^{\texttt{heaviest}} - U_{i+1}^{\texttt{lightest}}) \\ &= U_i^{\texttt{heaviest}} - U_{i+1}^{\texttt{heaviest}} + U_{i+1}^{\texttt{lightest}} - U_i^{\texttt{lightest}}. \end{aligned} \tag{B.13}$$

We cannot further reduce the equation B.13 since each term therein depends on the specific weight sampling scored from $\mathcal{D}$. However, we can tightly bound the maximum ΔG_{i+1} by considering all intermediate bins having the same total weight as $U_i^{\texttt{lightest}}$ after i^{th} ball. This way, we

imply:

$$U_{i+1}^{\texttt{lightest}} = U_i^{\texttt{lightest}}, \tag{B.14}$$

$$W_{i+1} \geq U_i^{\texttt{heaviest}} - U_i^{\texttt{lightest}}. \tag{B.15}$$

Thus, substituting equations B.12, B.14 and B.15 in equation B.13, ΔG is upper bounded as follows:

$$\begin{aligned}\Delta G_{i+1} &= U_i^{\texttt{heaviest}} - U_{i+1}^{\texttt{heaviest}} + U_{i+1}^{\texttt{lightest}} - U_i^{\texttt{lightest}} \\ &= U_i^{\texttt{heaviest}} - U_{i+1}^{\texttt{lightest}} \\ &\leq W_{i+1}. \end{aligned} \tag{B.16}$$

A statistical investigation of the upper bound on ΔG_{i+1} makes a randomly selected $W_{i+1} \to \bar{W}$. Therefore,

$$\Delta G_{i+1} \simeq \bar{W} \leq W_{i+1}. \qquad \blacksquare \tag{B.17}$$

Lemma 6 *Consider two samples of weighted balls: sample A of size m and sample B of size $m+1$. All balls in A and B sample their weights from the same uniform distribution $\mathcal{U} \in [0,1]$. Then,* $\Pr[\min(A) \geq \min(B)] = \frac{m}{m+1}$. *Thus,* $\min(A) \geq \min(B)$ *with high probability.*

Proof. Let A_i denote a random sample in A. Then, $\Pr\left[A_i \leq \frac{1}{m}\right] = \frac{1}{m}$ and thus, $\Pr\left[\min(A) \leq \frac{1}{m}\right] = 1$. Similarly, $\Pr\left[\min(B) \leq \frac{1}{m+1}\right] = 1$. Moreover, we can say $\min(A) \leq \frac{1}{m}$ and $\min(B) \leq \frac{1}{m+1}$ in $\mathcal{D}$. Using these, let us device another uniform distribution $\mathcal{D}' \in [0, \frac{1}{m}]$ from $\mathcal{D}$, which includes both $\min(A)$ and $\min(b)$. Then, $\Pr[\min(A) \geq \min(B)] = \frac{\min(B)}{\min(A)} = \frac{m}{m+1}$. Taking the limit as m goes to infinity yields $\min(A) \geq \min(B)$. $\blacksquare$

Using Lemma 6 and Eq. B.17 we can bound ΔG_m by

$$\Delta G_m \leq W_m \leq \frac{1}{m}. \tag{B.18}$$

Since $1 \geq W_1 \geq \frac{m-1}{m}$, the final discrepancy becomes

$$\begin{aligned} G_m &\geq W_1 - \sum_{i=2}^{m} W_i \\ &\geq 2W_1 - \sum_{i=1}^{m} W_i \\ &\geq 2W_1 - \sum_{i=1}^{m} \frac{1}{i}. \end{aligned} \tag{B.19}$$

B.0.5 Lower Bound on G_m

Deriving an upper bound for the final discrepancy G_m is hard. However, we can derive a non-trivial lower bound for both cases; namely, when each thrown ball triggered a "switch" and where none of the balls caused a "switch":

$$\left.\begin{aligned} \Delta G_2 = G_1 - G_2 \leq W_2 \\ \Delta G_3 = G_2 - G_3 \leq W_3 \\ \cdots \\ \Delta G_m = G_{m-1} - G_m \leq W_n \end{aligned}\right\} \text{Add all } \Delta G_i.$$

This gives

$$G_1 - G_m \leq \sum_{i=2}^{m} W_i,$$

where $G_1 = |W_1|$ and thus,

$$G_m \geq W_1 - \sum_{i=2}^{m} W_i \geq 0. \tag{B.20}$$

For statistically large m, we rewrite Eq. B.20 as

$$G_m \geq 2W_1 - \bar{W}m \geq 0. \qquad \blacksquare \tag{B.21}$$

APPENDIX

C

State-space Representation

The term "state-space" was mentioned for the first time in the area of control engineering [60]. A state-space representation is a mathematical model that describes a dynamic system by a set of input, output, and state variables, which are typically represented by vectors. This representation allows us to show the interdependencies between input, output, and state variables as well as their dependencies on time. An exemplary state-space representation of a continuous, time-invariant linear system can be given as follows:

$$\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + B\mathbf{v}(t)\,,$$
$$\mathbf{y}(t) = C\mathbf{x}(t) + D\mathbf{v}(t)\,,$$

where $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{v}$ are the state, output and input vectors, respectively. Their corresponding matrices A, B, C, and D are the state, input, measurement, and feed-through matrices, respectively. Moreover, $\dot{\mathbf{x}}$ denotes the time derivative of $\mathbf{x}$. Here, all matrices are time-invariant, meaning their values do not change with time.

State-space models can easily be generalized to also represent nonlinear, time-varying systems. In its generic form, the state-space representation

can be formulated as follows:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{v}(t), t)\,,$$
$$\mathbf{y}(t) = \mathbf{g}(\mathbf{x}(t), \mathbf{v}(t), t)\,,$$

where $\mathbf{f}$ and $\mathbf{g}$ are arbitrary (e.g., non-/linear) functions . The first equation is called the *state equation* and the second one is called the *observation equation*.

With the natural inclusion of time in the representation, the state-space model provides a general framework for the analysis of deterministic and stochastic dynamical systems. Such systems can be investigated successfully by Bayesian filters, which employ a *probabilistic* state-space model:

$$\mathbf{x}(0) \sim \mathrm{p}(\mathbf{x}(0))\,,$$
$$\mathbf{x}(t) \sim \mathrm{p}(\mathbf{x}(t)|\mathbf{x}(t-1))\,,$$
$$\mathbf{y}(t) \sim \mathrm{p}(\mathbf{y}(t)|\mathbf{x}(t))\,.$$

APPENDIX

D

Approximation Error of pcSIR in 2D

Approximating integrable functions by piecewise constant functions is well understood in mathematics on the basis of Riemann integral theory [181, 182]. We formulate the approximation error $\mathtt{E}_{\mathrm{pcSIR}}(l_x, l_y)$ of pcSIR with rectangular cells and then simplify it to $\mathtt{E}_{\mathrm{pcSIR}}(l)$ for square cells. All results can be extended to higher-dimensional settings.

Let the likelihood $\mathrm{p}(\mathbf{y}_t|\mathbf{x}_t)$ be a twice continuously differentiable function $f(x, y)$ within a domain $D \in \mathbb{R}^{[x_0,x_n]\times[y_0,y_m]}$, which is divided into $B = n \times m$ non-overlapping rectangular cells. Further, l_{k_i} and l_{k_j} denote the width (i.e., in x-direction) and the height (i.e., in y-direction) of cell I_k in D, where $D = \bigcup_{k=1}^{B} I_k$. The indices i and j are given by $i = 1, \ldots, n$ and $j = 1, \ldots, m$, and the maximum side lengths in both dimensions are defined as $l_x = \max_{k_i}(l_{k_i})$ and $l_y = \max_{k_j}(l_{k_j})$ where $l_{k_i} = x_k - x_{k-1}$ and $l_{k_j} = y_k - y_{k-1}$. Then, the total approximation error $\mathtt{E}_{\mathrm{pcSIR}}(l_x, l_y)$ of the likelihood in D obtained by pcSIR (Algorithm 8) is bounded by

$$\mathtt{E}_{\mathrm{pcSIR}}(l_x, l_y) \leq \frac{1}{24}\left[\max_{[D]}|f_{xx}|\, l_x^3\, l_y + \max_{[D]}|f_{yy}|\, l_x l_y^3\right],$$

where $\max_{[x_0,x_n]}|f_{xx}|$ and $\max_{[y_0,y_m]}|f_{yy}|$ are the maxima of the absolute values of $\frac{\partial^2 f}{\partial x^2}$ and $\frac{\partial^2 f}{\partial y^2}$ in D, respectively.

This result can be derived by mid-point Riemann-sum approximation of an integral. While the dummy particle does not have to be located at the center of a cell, for the sake of simplicity of the derivation, we assume that pcSIR uses the mid-point for piecewise constant likelihood approximation. Assume that $f(x, y)$ is twice continuously differentiable in region $D \in \mathbb{R}^{[x_0,x_n]\times[y_0,y_m]}$ and the following partial derivatives are defined: $\frac{\partial^2 f}{\partial x^2} = f_{xx}$, $\frac{\partial^2 f}{\partial y^2} = f_{yy}$ and $\frac{\partial^2 f}{\partial x \partial y} = f_{xy}$. The approximation error $\mathtt{E}_{I_k}(l_x, l_y)$ can be calculated by integrating the multivariate Taylor approximation

$$\begin{aligned}\mathtt{E}_{I_k}(l_x, l_y) &= f(x, y) - f(a, b)\\ &= f_x(a, b)(x - a) + f_y(a, b)(y - b)\\ &+ \frac{1}{2!}[f_{xx}(a, b)(x - a)^2 + f_{yy}(a, b)(y - b)^2\\ &+ 2f_{xy}(a, b)(x - a)(y - b)]\end{aligned} \tag{D.1}$$

over the two-dimensional interval $I_k = [x_{k-1}, x_k] \times [y_{k-1}, y_k]$, where $a = \frac{x_{k-1}+x_k}{2}$, $b = \frac{y_{k-1}+y_k}{2}$, and $D = \bigcup_{k=1}^{B} I_k$, hence:

$$\begin{aligned}\mathtt{E}_{I_k}(B) =& f_x(a, b) \iint_{I_k} (x - a)\,\mathrm{d}x\,\mathrm{d}y\\ &+ f_y(a, b) \iint_{I_k} (y - b)\,\mathrm{d}x\,\mathrm{d}y\\ &+ \frac{1}{2!}\left[f_{xx}(a, b) \iint_{I_k} (x - a)^2\,\mathrm{d}x\,\mathrm{d}y\right.\\ &+ f_{yy}(a, b) \iint_{I_k} (y - b)^2\,\mathrm{d}x\,\mathrm{d}y\\ &\left.+ 2f_{xy}(a, b) \iint_{I_k} (x - a)(y - b)\,\mathrm{d}x\,\mathrm{d}y\right].\end{aligned} \tag{D.2}$$

Substituting a and b in Eq. (D.2), we find:

$$\begin{aligned}\mathtt{E}_{I_k}(l_x, l_y) =& \frac{1}{2!}\left[f_{xx}(a, b) \iint_{I_k} \left(x - \frac{x_{k-1} + x_k}{2}\right)^2 \mathrm{d}x\,\mathrm{d}y\right.\\ &\left.+ f_{yy}(a, b) \iint_{I_k} \left(y - \frac{y_{k-1} + y_k}{2}\right)^2 \mathrm{d}x\,\mathrm{d}y\right].\end{aligned} \tag{D.3}$$

We substitute $l_{k_i} = x_k - x_{k-1}$, $l_{k_j} = y_k - y_{k-1}$ and evaluate the integrals. Then, Eq. (D.3) becomes

$$\mathtt{E}_{I_k}(l_x, l_y) = \frac{1}{24}\left[f_{xx}(a,b)\, l_{k_i}^3\, l_{k_j} + f_{yy}(a,b)\, l_{k_j}^3\, l_{k_i}\right]. \tag{D.4}$$

Next, we sum the absolute values of the partial errors in all I_k regions in order to provide an upper bound on the total approximation error in the closed region $[D]$ as:

$$\begin{aligned}\mathtt{E}(l_x, l_y) \leq \frac{1}{24}\Bigg[&\max_{[D]}|f_{xx}|\ \max_{k_i}(l_{k_i})^3\ \max_{k_j}(l_{k_j}) \\ &+ \max_{[D]}|f_{yy}|\ \max_{k_j}(l_{k_j})^3\ \max_{k_i}(l_{k_i})\Bigg].\end{aligned} \tag{D.5}$$

By substituting l_x and l_y into Eq. (D.5), we find the total error in the closed region $[D]$:

$$\mathtt{E}(l_x, l_y) \leq \frac{1}{24}\left[\max_{[D]}|f_{xx}|\, l_x^3\, l_y + \max_{[D]}|f_{yy}|\, l_x l_y^3\right]. \tag{D.6}$$

For equi-sized square cells, a tighter bound for the approximation error can be derived by repeating the steps that lead to Eq. (D.4). The derivation diverges here by taking the minimum possible value for the side lengths l_{k_i} and l_{k_j} of the small interval I_k in region $D \in \mathbb{R}^{[x_0,x_n]\times[y_0,y_m]}$, where $D = \bigcup_{k=1}^{B} I_k$. When I_k is a square with $l = l_{k_i} = x_k - x_{k-1} = l_{k_j} = y_k - y_{k-1}$, we obtain the bound on $\mathtt{E}_{I_k}(l)$ as:

$$\mathtt{E}_{l_k}(l) \leq \frac{l^4}{24}\left[f_{xx} + f_{yy}\right]. \tag{D.7}$$

By summing the absolute values of the errors in all I_k regions, we get the total error in closed region $[D]$:

$$\mathtt{E}(l) \leq \frac{l^4}{24}\left[\max_{[D]}|f_{xx}| + \max_{[D]}|f_{yy}|\right]. \tag{D.8}$$

APPENDIX

E

Tracking Efficiency Recovery in ARNA

In *information sharing performance* tests, we allow only a single PE to track the object effectively. In other words, only that PE has particles with heavy importance weights and these "heavy" particles need to be shared with other PEs as quickly as possible. In a perfect randomization setting for information sharing, each heavy PE communicates with a PE, which does not yet have heavy particles. In this ideal setting, the number of PEs that have heavy particles grows exponentially with each round of ARNA. Assuming a network size of M PEs, the minimum number of required iterations k to reach 100% tracking efficiency can be calculated as follows:

$$2^k = M \tag{E.1}$$

$$k = \frac{\log M}{\log 2} \tag{E.2}$$

From the above equation, we conclude that ARNA's parallel tracking efficiency scales with $\mathrm{O}(\log M)$ and thus, it outperforms RNA, whose parallel tracking efficiency complexity is $\mathrm{O}(M)$ since the locations of the PEs are not randomized in each iteration. In RNA, heavy particles travel the whole ring topology one neighboring PE at a time to reach perfect tracing efficiency, which can be achieved only slowly.

Bibliography

[1] Ö. Demirel, I. Smal, W. Niessen, E. Meijering, and I. F. Sbalzarini, "Piecewise constant sequential importance sampling for fast particle filtering," in *Data Fusion and Target Tracking Conference*, (Liverpool, UK), April 2014.

[2] Ö. Demirel, I. Smal, W. Niessen, E. Meijering, and I. F. Sbalzarini, "PPF - a parallel particle filtering library," in *Data Fusion and Target Tracking Conference*, (Liverpool, UK), April 2014.

[3] Ö. Demirel, I. Smal, W. Niessen, E. Meijering, and I. F. Sbalzarini, "Adaptive distributed resampling algorithm with non-proportional allocation," in *IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, May 4-9 2014.

[4] Ö. Demirel and I. F. Sbalzarini, "Distributed dynamic load balancing in massively parallel scientific numerical simulations," *(in prepation)*, 2014.

[5] I. F. Sbalzarini, "Abstractions and middleware for petascale computing and beyond," *International Journal of Distributed Systems and Technologies (IJDST)*, vol. 1, no. 2, pp. 40–56, 2010.

[6] C. Xu, F. Lau, B. Monien, and R. Lüling, "Nearest-neighbor algorithms for load-balancing in parallel computers," *Concurrency: Practice and Experience*, vol. 7, no. 7, pp. 707–736, 1995.

[7] N. J. Gordon, D. J. Salmond, and A. F. Smith, "Novel approach to nonlinear/non-Gaussian Bayesian state estimation," in *IEE Proceedings F (Radar and Signal Processing)*, vol. 140, pp. 107–113, IET, 1993.

[8] A. Doucet, S. Godsill, and C. Andrieu, "On sequential Monte Carlo sampling methods for Bayesian filtering," *Statistics and computing*, vol. 10, no. 3, pp. 197–208, 2000.

[9] A. Doucet, N. De Freitas, N. Gordon, *et al.*, *Sequential Monte Carlo methods in practice*, vol. 1. Springer New York, 2001.

[10] A. Doucet and A. M. Johansen, "A tutorial on particle filtering and smoothing: Fifteen years later," *Handbook of Nonlinear Filtering*, vol. 12, pp. 656–704, 2009.

[11] C. P. Robert and G. Casella, *Monte Carlo statistical methods*. Springer, 1999.

[12] N. Metropolis and S. Ulam, "The Monte Carlo method," *Journal of the American statistical association*, vol. 44, no. 247, pp. 335–341, 1949.

[13] W. K. Hastings, "Monte Carlo sampling methods using markov chains and their applications," *Biometrika*, vol. 57, no. 1, pp. 97–109, 1970.

[14] R. Rojas and U. Hashagen, *The first computers: history and architectures*. 2002.

[15] V. A. Voelz, G. R. Bowman, K. Beauchamp, and V. S. Pande, "Molecular simulation of ab initio protein folding for a millisecond folder ntl9 (1- 39)," *Journal of the American Chemical Society*, vol. 132, no. 5, pp. 1526–1528, 2010.

[16] N. A. Kaib, S. N. Raymond, and M. Duncan, "Planetary system disruption by galactic perturbations to wide binary stars," *Nature*, vol. 493, no. 7432, pp. 381–384, 2013.

[17] G. E. Moore *et al.*, "Cramming more components onto integrated circuits," 1965.

[18] G. Moore, "Excerpts from a conversation with gordon moore: Moore's law," *Video Transcript, Intel*, vol. 54, 2005.

[19] G. De Michell and R. K. Gupta, "Hardware/software co-design," *Proceedings of the IEEE*, vol. 85, no. 3, pp. 349–365, 1997.

[20] S. Aleksic, "Analysis of power consumption in future high-capacity network nodes," *Optical Communications and Networking, IEEE/OSA Journal of*, vol. 1, no. 3, pp. 245–258, 2009.

[21] P. K. Pepeljugoski, J. A. Kash, F. Doany, D. M. Kuchta, L. Schares, C. Schow, M. Taubenblatt, B. J. Offrein, and A. Benner, "Low power and high density optical interconnects for future supercomputers," in *Optical Fiber Communication Conference*, p. OThX2, Optical Society of America, 2010.

[22] Y.-C. Chow and W. H. Kohler, "Models for dynamic load balancing in a heterogeneous multiple processor system," *Computers, IEEE Transactions on*, vol. 100, no. 5, pp. 354–361, 1979.

[23] I. F. Sbalzarini, "Modeling and simulation of biological systems from image data," *Bioessays*, vol. 35, no. 5, pp. 482–490, 2013.

[24] G. E. Forsythe and W. R. Wasow, "Finite-difference methods for partial differential equations," 1960.

[25] O. C. Zienkiewicz and P. Morice, *The finite element method in engineering science*, vol. 1977. McGraw-hill London, 1971.

[26] R. J. LeVeque, *Finite volume methods for hyperbolic problems*, vol. 31. Cambridge university press, 2002.

[27] T. Belytschko, Y. Krongauz, D. Organ, M. Fleming, and P. Krysl, "Meshless methods: an overview and recent developments," *Computer methods in applied mechanics and engineering*, vol. 139, no. 1, pp. 3–47, 1996.

[28] G.-R. Liu, *Meshfree methods: moving beyond the finite element method*. CRC press, 2010.

[29] G. Ala, E. Francomano, A. Tortorici, E. Toscano, and F. Viola, "Corrective meshless particle formulations for time domain Maxwell's equations," *Journal of Computational and Applied Mathematics*, vol. 210, no. 1, pp. 34–46, 2007.

[30] R. Dalrymple and B. Rogers, "Numerical modeling of water waves with the sph method," *Coastal engineering*, vol. 53, no. 2, pp. 141–147, 2006.

[31] D. C. Rapaport, *The art of molecular dynamics simulation*. Cambridge university press, 2004.

[32] K. Binder and D. W. Heermann, *Monte Carlo simulation in statistical physics: an introduction*. Springer, 2010.

[33] J. Deutscher, A. Blake, and I. Reid, "Articulated body motion capture by annealed particle filtering," in *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, vol. 2, pp. 126–133, IEEE, 2000.

[34] L. Bretzner, I. Laptev, and T. Lindeberg, "Hand gesture recognition using multi-scale colour features, hierarchical models and particle filtering," in *Automatic Face and Gesture Recognition, 2002. Proceedings. Fifth IEEE International Conference on*, pp. 423–428, IEEE, 2002.

[35] P. Pérez, C. Hue, J. Vermaak, and M. Gangnet, "Color-based probabilistic tracking," in *Computer vision—ECCV 2002*, pp. 661–675, Springer, 2002.

[36] C. Shan, Y. Wei, T. Tan, and F. Ojardias, "Real time hand tracking by combining particle filtering and mean shift," in *Automatic Face and Gesture Recognition, 2004. Proceedings. Sixth IEEE International Conference on*, pp. 669–674, IEEE, 2004.

[37] H. Tamimi, H. Andreasson, A. Treptow, T. Duckett, and A. Zell, "Localization of mobile robots with omnidirectional vision using particle filter and iterative sift," *Robotics and Autonomous Systems*, vol. 54, no. 9, pp. 758–765, 2006.

[38] C. Choi and H. I. Christensen, "Robust 3d visual tracking using particle filtering on the special Euclidean group: A combined approach of keypoint and edge features," *The International Journal of Robotics Research*, vol. 31, no. 4, pp. 498–519, 2012.

[39] F. Caballero, L. Merino, I. Maza, and A. Ollero, "A particle filtering method for wireless sensor network localization with an aerial robot beacon," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pp. 596–601, IEEE, 2008.

[40] J.-M. Valin, F. Michaud, and J. Rouat, "Robust localization and tracking of simultaneous moving sound sources using beamforming and particle filtering," *Robotics and Autonomous Systems*, vol. 55, no. 3, pp. 216–228, 2007.

[41] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," *Robotics, IEEE Transactions on*, vol. 23, no. 1, pp. 34–46, 2007.

[42] L. Liao, D. J. Patterson, D. Fox, and H. Kautz, "Learning and inferring transportation routines," *Artificial Intelligence*, vol. 171, no. 5, pp. 311–331, 2007.

[43] S. Das, D. Lawless, B. Ng, and A. Pfeffer, "Factored particle filtering for data fusion and situation assessment in urban environments," in *Information Fusion, 2005 8th International Conference on*, vol. 2, pp. 8–pp, IEEE, 2005.

[44] L. Mihaylova, R. Boel, and A. Hegyi, "Freeway traffic estimation within particle filtering framework," *Automatica*, vol. 43, no. 2, pp. 290–300, 2007.

[45] J. Fernández-Villaverde and J. F. Rubio-Ramírez, "Estimating macroeconomic models: a likelihood approach," *The Review of Economic Studies*, vol. 74, no. 4, pp. 1059–1087, 2007.

[46] H. F. Lopes and R. S. Tsay, "Particle filters and Bayesian inference in financial econometrics," *Journal of Forecasting*, vol. 30, no. 1, pp. 168–209, 2011.

[47] T. Flury and N. Shephard, "Bayesian inference based only on simulated likelihood: particle filter analysis of dynamic economic models," *Econometric Theory*, vol. 27, no. 5, p. 933, 2011.

[48] D. Lautier, A. Javaheri, and A. Galli, "Filtering in finance," 2003.

[49] P. M. Djuric, M. Khan, and D. E. Johnston, "Particle filtering of stochastic volatility modeled with leverage," *Selected Topics in Signal Processing, IEEE Journal of*, vol. 6, no. 4, pp. 327–336, 2012.

[50] Y. Eroğlu and S. U. Seçkiner, "Wind farm layout optimization using particle filtering approach," *Renewable Energy*, vol. 58, pp. 95–107, 2013.

[51] M. D. Breitenstein, F. Reichlin, B. Leibe, E. Koller-Meier, and L. Van Gool, "Robust tracking-by-detection using a detector confidence particle filter," in *Computer Vision, 2009 IEEE 12th International Conference on*, pp. 1515–1522, IEEE, 2009.

[52] E. Meijering, I. Smal, and G. Danuser, "Tracking in molecular bioimaging," *Signal Processing Magazine, IEEE*, vol. 23, no. 3, pp. 46–53, 2006.

[53] Z. Khan, T. Balch, and F. Dellaert, "MCMC-based particle filtering for tracking a variable number of interacting targets," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 27, no. 11, pp. 1805–1819, 2005.

[54] M. Sanjeev Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking," *Signal Processing, IEEE Transactions on*, vol. 50, no. 2, pp. 174–188, 2002.

[55] Y. Rathi, N. Vaswani, A. Tannenbaum, and A. Yezzi, "Tracking deforming objects using particle filtering for geometric active contours," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 29, no. 8, pp. 1470–1475, 2007.

[56] I. Smal, E. Meijering, K. Draegestein, N. Galjart, I. Grigoriev, A. Akhmanova, M. Van Royen, A. B. Houtsmuller, and W. Niessen, "Multiple object tracking in molecular bioimaging by

rao-blackwellized marginal particle filtering," *Medical Image Analysis*, vol. 12, no. 6, pp. 764–777, 2008.

[57] 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," *Medical Imaging, IEEE Transactions on*, vol. 27, no. 6, pp. 789–804, 2008.

[58] F. Gustafsson, F. Gunnarsson, N. Bergman, U. Forssell, J. Jansson, R. Karlsson, and P.-J. Nordlund, "Particle filters for positioning, navigation, and tracking," *Signal Processing, IEEE Transactions on*, vol. 50, no. 2, pp. 425–437, 2002.

[59] E. Meijering, O. Dzyubachyk, I. Smal, *et al.*, "Methods for cell and particle tracking," *Methods Enzymol*, vol. 504, no. 9, pp. 183–200, 2012.

[60] R. E. Kalman *et al.*, "A new approach to linear filtering and prediction problems," *Journal of basic Engineering*, vol. 82, no. 1, pp. 35–45, 1960.

[61] E. A. Wan and R. Van Der Merwe, "The unscented Kalman filter for nonlinear estimation," in *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*, pp. 153–158, IEEE, 2000.

[62] S. Thrun, W. Burgard, D. Fox, *et al.*, *Probabilistic robotics*, vol. 1. MIT press Cambridge, 2005.

[63] L. Dagum and R. Menon, "OpenMP: an industry standard API for shared-memory programming," *Computational Science & Engineering, IEEE*, vol. 5, no. 1, pp. 46–55, 1998.

[64] D. Buttlar and J. Farrell, *Pthreads programming: A POSIX standard for better multiprocessing.* " O'Reilly Media, Inc.", 1996.

[65] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. S. Woodall, "Open MPI: Goals, concept, and design of a next generation MPI

implementation," in *Proceedings, 11th European PVM/MP Users' Group Meeting*, (Budapest, Hungary), pp. 97–104, September 2004.

[66] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[67] I. Sbalzarini, J. H. Walther, M. Bergdorf, S. Hieber, E. Kotsalis, and P. Koumoutsakos, "PPM–a highly efficient parallel particle–mesh library for the simulation of continuum systems," *Journal of Computational Physics*, vol. 215, no. 2, pp. 566–588, 2006.

[68] O. Awile, Ö. Demirel, and I. F. Sbalzarini, "Toward an object-oriented core of the PPM library," in *AIP Conference Proceedings*, vol. 1281, p. 1313, 2010.

[69] S. Balay, J. Brown, K. Buschelman, V. Eijkhout, W. Gropp, D. Kaushik, M. Knepley, L. C. McInnes, B. Smith, and H. Zhang, "PETSc users manual revision 3.4," 2013.

[70] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, *et al.*, "An overview of the trilinos project," *ACM Transactions on Mathematical Software (TOMS)*, vol. 31, no. 3, pp. 397–423, 2005.

[71] J. V. Reynders, P. J. Hinker, J. C. Cummings, S. R. Atlas, S. Banerjee, W. F. Humphrey, S. R. Karmesin, K. Keahey, M. Srikant, and M. D. Tholburn, "Pooma: A framework for scientific simulations on parallel architectures," *Parallel Programming in C+*, pp. 547–588, 1996.

[72] M. Valiev, E. J. Bylaska, N. Govind, K. Kowalski, T. P. Straatsma, H. J. Van Dam, D. Wang, J. Nieplocha, E. Apra, T. L. Windus, *et al.*, "Nwchem: a comprehensive and scalable open-source solution for large scale molecular simulations," *Computer Physics Communications*, vol. 181, no. 9, pp. 1477–1489, 2010.

[73] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kale, and K. Schulten, "Scalable

molecular dynamics with namd," *Journal of computational chemistry*, vol. 26, no. 16, pp. 1781–1802, 2005.

[74] N. Minar, R. Burkhart, C. Langton, and M. Askenazi, "The swarm simulation system: A toolkit for building multi-agent simulations," Santa Fe Institute Santa Fe, 1996.

[75] M. Frigo and S. G. Johnson, "Fftw: An adaptive software architecture for the fft," in *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, vol. 3, pp. 1381–1384, IEEE, 1998.

[76] B. Moon and J. Saltz, "Adaptive runtime support for direct simulation Monte Carlo methods on distributed memory architectures," in *Proceedings of the IEEE Scalable High-Performance Computing Conference*, pp. 176–183, IEEE, 1994.

[77] L. M. Ni and K. Hwang, "Optimal load balancing in a multiple processor system with many job classes," *Software Engineering, IEEE Transactions on*, no. 5, pp. 491–496, 1985.

[78] I. Ahmad and A. Ghafoor, "A semi distributed task allocation strategy for large hypercube supercomputers," in *Supercomputing'90. Proceedings of*, pp. 898–907, IEEE, 1990.

[79] J. Lifflander, S. Krishnamoorthy, and L. V. Kale, "Work stealing and persistence-based load balancers for iterative overdecomposed applications," in *Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing*, pp. 137–148, ACM, 2012.

[80] G. Zheng, A. Bhatelé, E. Meneses, and L. V. Kalé, "Periodic hierarchical load balancing for large supercomputers," *International Journal of High Performance Computing Applications*, vol. 25, no. 4, pp. 371–385, 2011.

[81] B. Yagoubi and Y. Slimani, "Task load balancing strategy for grid computing," *Journal of Computer Science*, vol. 3, no. 3, p. 186, 2007.

[82] H. Menon and L. Kalé, "A distributed dynamic load balancer for iterative applications," in *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*, p. 15, ACM, 2013.

[83] G. Cybenko, "Dynamic load balancing for distributed memory multiprocessors," *Journal of parallel and distributed computing*, vol. 7, no. 2, pp. 279–301, 1989.

[84] J. E. Boillat, "Load balancing and poisson equation in a graph," *Concurrency: Practice and Experience*, vol. 2, no. 4, pp. 289–313, 1990.

[85] B. Ghosh and S. Muthukrishnan, "Dynamic load balancing by random matchings," *Journal of Computer and System Sciences*, vol. 53, no. 3, pp. 357–370, 1996.

[86] Y. Rabani, A. Sinclair, and R. Wanka, "Local divergence of markov chains and the analysis of iterative load-balancing schemes," in *Foundations of Computer Science, 1998. Proceedings. 39th Annual Symposium on*, pp. 694–703, IEEE, 1998.

[87] G. O. Roberts, A. Gelman, and W. R. Gilks, "Weak convergence and optimal scaling of random walk metropolis algorithms," *The annals of applied probability*, vol. 7, no. 1, pp. 110–120, 1997.

[88] T. C. Aysal, M. E. Yildiz, A. D. Sarwate, and A. Scaglione, "Broadcast gossip algorithms for consensus," *Signal Processing, IEEE Transactions on*, vol. 57, no. 7, pp. 2748–2761, 2009.

[89] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms," *Information Theory, IEEE Transactions on*, vol. 52, no. 6, pp. 2508–2530, 2006.

[90] S. Muthukrishnan, B. Ghosh, and M. H. Schultz, "First-and second-order diffusive methods for rapid, coarse, distributed load balancing," *Theory of computing systems*, vol. 31, no. 4, pp. 331–354, 1998.

[91] M. Mitzenmacher, "The power of two choices in randomized load balancing," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 12, no. 10, pp. 1094–1104, 2001.

[92] T. Friedrich and T. Sauerwald, "Near-perfect load balancing by randomized rounding," in *Proceedings of the 41st annual ACM symposium on Theory of computing*, pp. 121–130, ACM, 2009.

[93] T. Sauerwald and H. Sun, "Tight bounds for randomized load balancing on arbitrary network topologies," in *Foundations of Computer Science (FOCS), 2012 IEEE 53rd Annual Symposium on*, pp. 341–350, IEEE, 2012.

[94] Ö. Demirel and I. F. Sbalzarini, "Balancing indivisible real-valued loads in arbitrary networks," *arXiv preprint arXiv:1308.0148*, 2013.

[95] D. Brélaz, "New methods to color the vertices of a graph," *Commun. ACM*, vol. 22, no. 4, pp. 251–256, 1979.

[96] M. Raab and A. Steger, ""balls into bins"—a simple and tight analysis," in *Randomization and Approximation Techniques in Computer Science*, pp. 159–170, Springer, 1998.

[97] P. Berenbrink, A. Czumaj, A. Steger, and B. Vöcking, "Balanced allocations: The heavily loaded case," *SIAM Journal on Computing*, vol. 35, no. 6, pp. 1350–1385, 2006.

[98] N. L. Johnson and S. Kotz, *Urn models and their application: an approach to modern discrete probability theory.* Wiley New York, 1977.

[99] V. F. Kolchin, B. A. Sevastianov, and V. P. Chistiakov, *Random allocations.* Vh Winston New York, 1978.

[100] Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal, "Balanced allocations," in *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pp. 593–602, ACM, 1994.

[101] Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal, "Balanced allocations," *SIAM journal on computing*, vol. 29, no. 1, pp. 180–200, 1999.

[102] P. Berenbrink, T. Friedetzky, Z. Hu, and R. Martin, "On weighted balls-into-bins games," *Theoretical Computer Science*, vol. 409, no. 3, pp. 511–520, 2008.

[103] K. Talwar and U. Wieder, "Balanced allocations: the weighted case," in *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pp. 256–265, ACM, 2007.

[104] Y. Peres, K. Talwar, and U. Wieder, "The (1+ β)-choice process and weighted balls-into-bins," in *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1613–1619, Society for Industrial and Applied Mathematics, 2010.

[105] S. Dutta, S. Bhattacherjee, and A. Narang, "Perfectly balanced allocation with estimated average using approximately constant retries," *CoRR, vol. abs/1111.0801*, 2011.

[106] A. Czumaj, C. Riley, and C. Scheideler, "Perfectly balanced allocation," in *Approximation, Randomization, and Combinatorial Optimization.. Algorithms and Techniques*, pp. 240–251, Springer, 2003.

[107] T. A. Standish, *Data structures in Java*. Addison-Wesley Longman Publishing Co., Inc., 1997.

[108] K.-D. Neubert, ""flashsort"," *Dr. Dobbs Journal*, 1998.

[109] C. A. Hoare, "Quicksort," *The Computer Journal*, vol. 5, no. 1, pp. 10–16, 1962.

[110] J. N. Hooker, "A quantitative approach to logical inference," *Decision Support Systems*, vol. 4, no. 1, pp. 45–69, 1988.

[111] V. Chandru and J. Hooker, *Optimization methods for logical inference*, vol. 34. John Wiley & Sons, 2011.

[112] M. Bayes and M. Price, "An essay towards solving a problem in the doctrine of chances. by the late rev. mr. bayes, frs communicated by mr. price, in a letter to john canton, amfrs," *Philosophical Transactions*, vol. 53, pp. 370–418, 1763.

[113] P. S. marquis de Laplace, *Essai philosophique sur les probabilités: précédé d'une notice historique sur le calcul des probabilités*, vol. 1. Chez H. Remy, 1829.

[114] J. M. Bernardo and A. F. Smith, *Bayesian theory*, vol. 405. John Wiley & Sons, 2009.

[115] S. J. Press, *Subjective and objective Bayesian statistics: principles, models, and applications*, vol. 590. John Wiley & Sons, 2009.

[116] G. Casella and R. L. Berger, *Statistical inference*, vol. 70. Duxbury Press Belmont, CA.

[117] H. M. Walker and J. Lev, "Statistical inference.," 1953.

[118] Y.-C. Ho and R. Lee, "A Bayesian approach to problems in stochastic estimation and control," *Automatic Control, IEEE Transactions on*, vol. 9, no. 4, pp. 333–339, 1964.

[119] J. Spragins, "A note on the iterative application of bayes' rule," *Information Theory, IEEE Transactions on*, vol. 11, no. 4, pp. 544–549, 1965.

[120] Z. Chen, "Bayesian filtering: From Kalman filters to particle filters, and beyond," *Statistics*, vol. 182, no. 1, pp. 1–69, 2003.

[121] J. V. Candy, *Bayesian signal processing: Classical, modern and particle filtering methods*, vol. 54. John Wiley & Sons, 2011.

[122] S. Särkkä, *Bayesian filtering and smoothing*, vol. 3. Cambridge University Press, 2013.

[123] W. R. Gilks, *Markov chain Monte Carlo*. Wiley Online Library, 2005.

[124] R. M. Neal, "Probabilistic inference using markov chain Monte Carlo methods," 1993.

[125] B. D. Anderson and J. B. Moore, *Optimal filtering*. Courier Dover Publications, 2012.

[126] R. E. Kalman and R. S. Bucy, "New results in linear filtering and prediction theory," *Journal of Basic Engineering*, vol. 83, no. 3, pp. 95–108, 1961.

[127] A. H. Jazwinski, "Stochastic processes and filtering theory," *New York: Academic*, 1970.

[128] B. Stenger, P. R. Mendonça, and R. Cipolla, "Model-based hand tracking using an unscented Kalman filter.," in *BMVC*, vol. 1, pp. 63–72, 2001.

[129] R. J. Qian, M. I. Sezan, and K. E. Matthews, "A robust real-time face tracking algorithm," in *Image Processing, 1998. ICIP 98. Proceedings. 1998 International Conference on*, vol. 1, pp. 131–135, IEEE, 1998.

[130] G. M. Siouris, G. Chen, and J. Wang, "Tracking an incoming ballistic missile using an extended interval Kalman filter," *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 33, no. 1, pp. 232–240, 1997.

[131] A. C. Harvey, *Forecasting, structural time series models and the Kalman filter*. Cambridge university press, 1990.

[132] C. Doz, D. Giannone, and L. Reichlin, "A two-step estimator for large approximate dynamic factor models based on Kalman filtering," *Journal of Econometrics*, vol. 164, no. 1, pp. 188–205, 2011.

[133] M. St-Pierre and D. Gingras, "Comparison between the unscented Kalman filter and the extended Kalman filter for the position estimation module of an integrated navigation information system," in *Intelligent Vehicles Symposium, 2004 IEEE*, pp. 831–835, IEEE, 2004.

[134] Q. Gan and C. J. Harris, "Comparison of two measurement fusion methods for Kalman-filter-based multisensor data fusion," *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 37, no. 1, pp. 273–279, 2001.

[135] S.-L. Sun and Z.-L. Deng, "Multi-sensor optimal information fusion Kalman filter," *Automatica*, vol. 40, no. 6, pp. 1017–1023, 2004.

[136] L. Matthies, T. Kanade, and R. Szeliski, "Kalman filter-based algorithms for estimating depth from image sequences," *International Journal of Computer Vision*, vol. 3, no. 3, pp. 209–238, 1989.

[137] D. J. MacKay, "Choice of basis for Laplace approximation," *Machine learning*, vol. 33, no. 1, pp. 77–86, 1998.

[138] D. J. MacKay, *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[139] J. F. De Freitas, *Bayesian methods for neural networks*. PhD thesis, Citeseer, 2003.

[140] H. J. Kushner and A. S. Budhiraja, "A nonlinear filtering algorithm based on an approximation of the conditional distribution," *Automatic Control, IEEE Transactions on*, vol. 45, no. 3, pp. 580–585, 2000.

[141] R. Bucy, "Bayes theorem and digital realizations for non-linear filters(one dimensional discrete time nonlinear filters synthesis using bayes theorem and digital techniques)," *Journal of the Astronautical Sciences*, vol. 17, pp. 80–94, 1969.

[142] G. Kitagawa, "Non-Gaussian state—space modeling of nonstationary time series," *Journal of the American statistical association*, vol. 82, no. 400, pp. 1032–1041, 1987.

[143] A. Pole, M. West, and J. Harrison, *Applied Bayesian forecasting and time series analysis*. CRC Press, 1994.

[144] R. S. Bucy and K. D. Senne, "Digital synthesis of non-linear filters," *Automatica*, vol. 7, no. 3, pp. 287–298, 1971.

[145] S. C. Kramer and H. W. Sorenson, "Recursive Bayesian estimation using piece-wise constant approximations," *Automatica*, vol. 24, no. 6, pp. 789–801, 1988.

[146] V. Peterka, "Bayesian system identification," *Automatica*, vol. 17, no. 1, pp. 41–53, 1981.

[147] K. Srinivasan, "State estimation by orthogonal expansion of probability distributions," *Automatic Control, IEEE Transactions on*, vol. 15, no. 1, pp. 3–10, 1970.

[148] A. C. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss, "Fast, small-space algorithms for approximate histogram maintenance," in *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pp. 389–398, ACM, 2002.

[149] P. Muldowney, "Topics in probability using generalised riemann integration," in *MATHEMATICAL PROCEEDINGS-ROYAL IRISH ACADEMY*, vol. 99, pp. 39–50, 1999.

[150] H. W. Sorenson and D. L. Alspach, "Recursive Bayesian estimation using Gaussian sums," *Automatica*, vol. 7, no. 4, pp. 465–479, 1971.

[151] D. L. Alspach and H. W. Sorenson, "Nonlinear Bayesian estimation using Gaussian sum approximations," *Automatic Control, IEEE Transactions on*, vol. 17, no. 4, pp. 439–448, 1972.

[152] E. A. Wan and R. Van Der Merwe, "The unscented Kalman filter," *Kalman filtering and neural networks*, pp. 221–280, 2001.

[153] S. J. Julier and J. K. Uhlmann, "A new extension of the Kalman filter to nonlinear systems," in *Int. symp. aerospace/defense sensing, simul. and controls*, vol. 3, pp. 3–2, Orlando, FL, 1997.

[154] H. Niederreiter, *Random number generation and quasi-Monte Carlo methods*, vol. 63. SIAM, 1992.

[155] I. H. Sloan and H. Woźniakowski, "When are quasi-Monte Carlo algorithms efficient for high dimensional integrals?," *Journal of Complexity*, vol. 14, no. 1, pp. 1–33, 1998.

[156] R. E. Caflisch, "Monte Carlo and quasi-Monte Carlo methods," *Acta numerica*, vol. 7, pp. 1–49, 1998.

[157] W. Fong, S. J. Godsill, A. Doucet, and M. West, "Monte Carlo smoothing with application to audio signal enhancement," *Signal Processing, IEEE Transactions on*, vol. 50, no. 2, pp. 438–449, 2002.

[158] M. H. Kalos and P. A. Whitlock, *Monte Carlo methods*. John Wiley & Sons, 2008.

[159] M. E. Newman, G. T. Barkema, and M. Newman, *Monte Carlo methods in statistical physics*, vol. 13. Clarendon Press Oxford, 1999.

[160] N. Hansen, S. D. Müller, and P. Koumoutsakos, "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es)," *Evolutionary Computation*, vol. 11, no. 1, pp. 1–18, 2003.

[161] E. Aarts and J. Korst, "Simulated annealing and boltzmann machines: a stochastic approach to combinatorial optimization and neural computing," 1988.

[162] A. Y. Khintchine, "Limit laws of sums of independent random variables," *ONTI, Moscow,(Russian)*, 1938.

[163] F. Daum and J. Huang, "Curse of dimensionality and particle filters," in *Aerospace Conference, 2003. Proceedings. 2003 IEEE*, vol. 4, pp. 4_1979–4_1993, IEEE, 2003.

[164] C. Snyder, T. Bengtsson, P. Bickel, and J. Anderson, "Obstacles to high-dimensional particle filtering.," *Monthly Weather Review*, vol. 136, no. 12, 2008.

[165] A. W. Marshall, "The use of multi-stage sampling schemes in Monte Carlo computations," tech. rep., DTIC Document, 1954.

[166] J. Handschin and D. Q. Mayne, "Monte Carlo techniques to estimate the conditional expectation in multi-stage non-linear filtering," *International journal of control*, vol. 9, no. 5, pp. 547–559, 1969.

[167] P. W. Glynn and D. L. Iglehart, "Importance sampling for stochastic simulations," *Management Science*, vol. 35, no. 11, pp. 1367–1392, 1989.

[168] J. S. Liu, R. Chen, and T. Logvinenko, "A theoretical framework for sequential importance sampling with resampling," in *Sequential Monte Carlo methods in practice*, pp. 225–246, Springer, 2001.

[169] S. Geman and D. Geman, "Stochastic relaxation, gibbs distributions, and the Bayesian restoration of images," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, no. 6, pp. 721–741, 1984.

[170] G. Casella and E. I. George, "Explaining the gibbs sampler," *The American Statistician*, vol. 46, no. 3, pp. 167–174, 1992.

[171] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *The journal of chemical physics*, vol. 21, no. 6, pp. 1087–1092, 2004.

[172] S. Chib and E. Greenberg, "Understanding the metropolis-hastings algorithm," *The American Statistician*, vol. 49, no. 4, pp. 327–335, 1995.

[173] A. Kong, J. S. Liu, and W. H. Wong, "Sequential imputations and Bayesian missing data problems," *Journal of the American statistical association*, vol. 89, no. 425, pp. 278–288, 1994.

[174] J. Handschin, "Monte Carlo techniques for prediction and filtering of non-linear stochastic processes," *Automatica*, vol. 6, no. 4, pp. 555–563, 1970.

[175] S. N. MacEachern, M. Clyde, and J. S. Liu, "Sequential importance sampling for nonparametric bayes models: The next generation," *Canadian Journal of Statistics*, vol. 27, no. 2, pp. 251–267, 1999.

[176] J. S. Liu, R. Chen, and W. H. Wong, "Rejection control and sequential importance sampling," *Journal of the American Statistical Association*, vol. 93, no. 443, pp. 1022–1031, 1998.

[177] G. Kitagawa, "Monte Carlo filter and smoother for non-Gaussian nonlinear state space models," *Journal of computational and graphical statistics*, vol. 5, no. 1, pp. 1–25, 1996.

[178] J. D. Hol, T. B. Schon, and F. Gustafsson, "On resampling algorithms for particle filters," in *Nonlinear Statistical Signal Processing Workshop, 2006 IEEE*, pp. 79–82, IEEE, 2006.

[179] J. S. Liu and R. Chen, "Blind deconvolution via sequential imputations," *Journal of the American Statistical Association*, vol. 90, no. 430, pp. 567–576, 1995.

[180] L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations," *Journal of computational physics*, vol. 73, no. 2, pp. 325–348, 1987.

[181] P. J. Davis and P. Rabinowitz, *Numerical integration*. Blaisdell Publishing Company London, 1967.

[182] G. B. Thomas, R. L. Finney, and M. D. Weir, *Calculus and analytic geometry*, vol. 7. Addison-Wesley Reading, Massachusetts, 1984.

[183] F. Abdallah, A. Gning, and P. Bonnifait, "Box particle filtering for nonlinear state estimation using interval analysis," *Automatica*, vol. 44, no. 3, pp. 807–815, 2008.

[184] L. Jaulin, *Applied interval analysis: with examples in parameter and state estimation, robust control and robotics*. Springer, 2001.

[185] A. Gning, B. Ristic, L. Mihaylova, and F. Abdallah, "An introduction to box particle filtering [lecture notes]," *Signal Processing Magazine, IEEE*, vol. 30, no. 4, pp. 166–171, 2013.

[186] A. Gning, L. Mihaylova, and F. Abdallah, "Mixture of uniform probability density functions for non linear state estimation using interval analysis," in *Information Fusion (FUSION), 2010 13th Conference on*, pp. 1–8, IEEE, 2010.

[187] A. Akhmanova and C. C. Hoogenraad, "Microtubule plus-end-tracking proteins: Mechanisms and functions," *Current Opinion in Cell Biology*, vol. 17, no. 1, pp. 47–54, 2005.

[188] Y. Komarova, C. O. De Groot, I. Grigoriev, S. M. Gouveia, E. L. Munteanu, J. M. Schober, S. Honnappa, R. M. Buey, C. C. Hoogenraad, M. Dogterom, *et al.*, "Mammalian end binding proteins control persistent microtubule growth," *The Journal of cell biology*, vol. 184, no. 5, pp. 691–706, 2009.

[189] J. A. Helmuth and I. F. Sbalzarini, "Deconvolving active contours for fluorescence microscopy images," in *Advances in Visual Computing*, pp. 544–553, Springer, 2009.

[190] J. Helmuth, G. Paul, and I. Sbalzarini, "Beyond co-localization: inferring spatial interactions between sub-cellular structures from microscopy images," *BMC bioinformatics*, vol. 11, no. 1, p. 372, 2010.

[191] A. Shivanandan, A. Radenovic, and I. F. Sbalzarini, "MosaicIA: an ImageJ/Fiji plugin for spatial pattern and interaction analysis," *BMC Bioinformatics*, vol. 14, p. 349, 2013.

[192] W. J. Godinez, M. Lampe, P. Koch, R. Eils, B. Muller, and K. Rohr, "Identifying virus-cell fusion in two-channel fluorescence microscopy image sequences based on a layered probabilistic approach," *Medical Imaging, IEEE Transactions on*, vol. 31, no. 9, pp. 1786–1808, 2012.

[193] X. Rong Li and V. P. Jilkov, "Survey of maneuvering target tracking. Part I. Dynamic models," *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 39, no. 4, pp. 1333–1364, 2003.

[194] D. Thomann, D. Rines, P. Sorger, and G. Danuser, "Automatic fluorescent tag detection in 3d with super-resolution: application to the analysis of chromosome movement," *Journal of Microscopy*, vol. 208, no. 1, pp. 49–64, 2002.

[195] B. Zhang, J. Zerubia, and J.-C. Olivo-Marin, "Gaussian approximations of fluorescence microscope point-spread function models," *Applied Optics*, vol. 46, no. 10, pp. 1819–1829, 2007.

[196] B. Ristic, S. Arulampalm, and N. J. Gordon, *Beyond the Kalman filter: Particle filters for tracking applications*. Artech House Publishers, 2004.

[197] M. K. Cheezum, W. F. Walker, and W. H. Guilford, "Quantitative comparison of algorithms for tracking single fluorescent particles," *Biophysical journal*, vol. 81, no. 4, pp. 2378–2388, 2001.

[198] E. Koblents and J. Mıguez, "A population Monte Carlo scheme with transformed weights and its application to stochastic kinetic models," in *ICASSP*, 2013.

[199] K. Nummiaro, E. Koller-Meier, and L. Van Gool, "An adaptive color-based particle filter," *Image and Vision Computing*, vol. 21, no. 1, pp. 99–110, 2003.

[200] Z. Wang, X. Yang, Y. Xu, and S. Yu, "Camshift guided particle filter for visual tracking," *Pattern Recognition Letters*, vol. 30, no. 4, pp. 407–413, 2009.

[201] M. Bolic, P. M. Djuric, and S. Hong, "Resampling algorithms and architectures for distributed particle filters," *Signal Processing, IEEE Transactions on*, vol. 53, no. 7, pp. 2442–2450, 2005.

[202] A. Doucet, N. De Freitas, N. Gordon, *et al.*, *Sequential Monte Carlo methods in practice*, vol. 1. Springer New York, 2001.

[203] J. Geweke, "Bayesian inference in econometric models using Monte Carlo integration," *Econometrica: Journal of the Econometric Society*, pp. 1317–1339, 1989.

[204] I. Strid, "Parallel particle filters for likelihood evaluation in dsge models: An assessment," tech. rep., Society for Computational Economics, 2006.

[205] J. Míguez, M. F. Bugallo, and P. M. Djurić, "A new class of particle filters for random dynamic systems with unknown statistics," *EURASIP Journal on Applied Signal Processing*, vol. 2004, pp. 2278–2294, 2004.

[206] J. Míguez, "Analysis of parallelizable resampling algorithms for particle filtering," *Signal Processing*, vol. 87, no. 12, pp. 3155–3174, 2007.

[207] S. 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*, vol. 24, no. 4, pp. 319–333, 2010.

[208] M. A. Goodrum, M. J. Trotter, A. Aksel, S. T. Acton, and K. Skadron, "Parallelization of particle filter algorithms," in *Computer Architecture*, pp. 139–149, Springer, 2012.

[209] J. A. Brown and D. W. Capson, "A framework for 3d model-based visual tracking using a gpu-accelerated particle filter," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 18, no. 1, pp. 68–80, 2012.

[210] G. Hendeby, R. Karlsson, and F. Gustafsson, "Particle filtering: the need for speed," *EURASIP Journal on Advances in Signal Processing*, vol. 2010, p. 22, 2010.

[211] S. Maskell, B. Alun-Jones, and M. Macleod, "A single instruction multiple data particle filter," in *Nonlinear Statistical Signal Processing Workshop, 2006 IEEE*, pp. 51–54, IEEE, 2006.

[212] L. M. Murray, "Bayesian state-space modelling on high-performance hardware using libbi," *arXiv preprint arXiv:1306.3277*, 2013.

[213] MPI Forum, "Message Passing Interface (MPI) Forum Home Page." `http://www.mpi-forum.org/` (Dec. 2009).

[214] O. Vega-Gisbert, J. E. Roman, S. Groß, and J. M. Squyres, "Towards the availability of java bindings in open mpi," in *Proceedings of the 20th European MPI Users' Group Meeting*, pp. 141–142, ACM, 2013.

[215] A. Cheptsov, "Promoting data-centric supercomputing to the www world: Open mpi's java bindings," in *Advanced Information Networking and Applications Workshops (WAINA), 2013 27th International Conference on*, pp. 1417–1422, IEEE, 2013.

[216] M. D. Abràmoff, P. J. Magalhães, and S. J. Ram, "Image processing with imagej," *Biophotonics international*, vol. 11, no. 7, pp. 36–42, 2004.

[217] J. Schindelin, I. Arganda-Carreras, E. Frise, V. Kaynig, M. Longair, T. Pietzsch, S. Preibisch, C. Rueden, S. Saalfeld, B. Schmid, *et al.*, "Fiji: an open-source platform for biological-image analysis," *Nature methods*, vol. 9, no. 7, pp. 676–682, 2012.

[218] "Imagescience.org," 10 2013.

[219] C. Andrieu, A. Doucet, and R. Holenstein, "Particle Markov chain Monte Carlo methods," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 72, no. 3, pp. 269–342, 2010.

[220] N. Chopin, P. E. Jacob, and O. Papaspiliopoulos, "Smc2: an efficient algorithm for sequential analysis of state space models," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 2012.

[221] B.-N. Vo, S. Singh, and A. Doucet, "Sequential Monte Carlo implementation of the phd filter for multi-target tracking," in *Proc. Int'l Conf. on Information Fusion*, pp. 792–799, 2003.

[222] M. Baker, B. Carpenter, G. Fox, S. H. Ko, and S. Lim, "mpijava: An object-oriented java interface to MPI," in *Parallel and Distributed Processing*, pp. 748–762, Springer, 1999.

[223] "Open MPI: Trunk nightly snapshot tarballs," 09 2013.

[224] J. M. Bull and M. E. Kambites, "Jomp—an OpenMP-like interface for java," in *Proceedings of the ACM 2000 conference on Java Grande*, pp. 44–53, ACM, 2000.

[225] M. Klemm, M. Bezold, R. Veldema, and M. Philippsen, "Jamp: An implementation of OpenMP for a java dsm," *Concurrency and Computation: Practice and Experience*, vol. 19, no. 18, pp. 2333–2352, 2007.

[226] M. Lange, G. Gorman, M. Weiland, L. Mitchell, and J. Southern, "Achieving efficient strong scaling with PETSc using hybrid MPI/OpenMP optimisation," in *Supercomputing*, pp. 97–108, Springer, 2013.

[227] L. Dagum and R. Menon, "OpenMP: an industry standard API for shared-memory programming," *Computational Science & Engineering, IEEE*, vol. 5, no. 1, pp. 46–55, 1998.

[228] R. V. Workshop, "Why hybrid? or why not?," 07 2013.

[229] B. Ghosh and S. Muthukrishnan, "Dynamic load balancing by random matchings," *Journal of Computer and System Sciences*, vol. 53, no. 3, pp. 357–370, 1996.

[230] R. A. Fisher, F. Yates, *et al.*, "Statistical tables for biological, agricultural and medical research.," *Statistical tables for biological, agricultural and medical research.*, no. Ed. 3., 1949.

[231] Y. Komarova, C. O. de Groot, I. Grigoriev, S. M. 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," vol. 184, no. 5, pp. 691–706, 2009.

[232] Ö. Demirel, I. Smal, W. Niessen, E. Meijering, and I. F. Sbalzarini, "PPF - a parallel particle filtering library," *arXiv preprint arXiv:1310.5045*, 2013.

[233] Ö. Demirel and I. F. Sbalzarini, "Balanced offline allocation of weighted balls into bins," *arXiv preprint arXiv:1304.2881*, 2013.

[234] B. T. Vo, *Random finite sets in multi-object filtering*. Citeseer, 2008.

[235] A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman, "Loggp: incorporating long messages into the logp model—one step closer towards a realistic model for parallel computation," in *Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures*, pp. 95–105, ACM, 1995.

[236] J. Kennedy, R. Eberhart, *et al.*, "Particle swarm optimization," in *Proceedings of IEEE international conference on neural networks*, vol. 4, pp. 1942–1948, Perth, Australia, 1995.

[237] D. Brélaz, "New methods to color the vertices of a graph," *Communications of the ACM*, vol. 22, no. 4, pp. 251–256, 1979.

[238] A. Panconesi and A. Srinivasan, "Randomized distributed edge coloring via an extension of the chernoff–hoeffding bounds," *SIAM Journal on Computing*, vol. 26, no. 2, pp. 350–368, 1997.

Curriculum Vitae

Ömer Demirel
March 20, 1983, Istanbul, Turkey

Hertelstrasse 21
01307 Dresden
Germany

omer@inf.ethz.ch

Education

2010 – 2014	Ph.D. in Computer Science, Institute of Theoretical Computer Science, ETH Zurich, Switzerland
2007 – 2009	M.Sc. in Computational Science and Engineering, Institute of Computer Science, TU München, Germany
2002 – 2007	B.Sc. in Computer Engineering, Koç University, Istanbul, Turkey
2002 – 2007	B.Sc. in Physics, Koç University, Istanbul, Turkey
1994 – 2002	Abitur, German High School, Istanbul, Turkey
1994 – 2002	Turkish HS diploma, German High School, Istanbul, Turkey

Practical Experience

08/2012 – 07/2014	Research assistant, MOSAIC Group, Center of Systems Biology Dresden (CSBD), Max Planck Institute of Molecular Cel Biology and Genetics (MPI-CBG), Dresden, Germany
01/2010 – 07/2012	Research assistant, MOSAIC Group, Institute of Theoretical Computer Science, ETH Zurich, Switzerland
07/2008 – 10/2008	Software engineering intern, TrustYou GmbH, Munich, Germany
08/2006 – 09/2006	Software engineering intern, Gordion Bilgi Hizmet Ltd Sti, Istanbul, Turkey
07/2006 – 08/2006	Software engineering intern, Bilmed Bilgisayar ve Yazılım A.S., Istanbul, Turkey
08/2005 – 09/2005	Software engineering intern, TUBITAK Marmara Research Center, Kocaeli, Turkey
08/2004 – 09/2004	Software engineering intern, Hewlett-Packard (HP), Istanbul, Turkey

Honors & Awards

05/2013 – 08/2013	DAAD Rise award for summer internship project proposal, MPI-CBG
02/2008 – 12/2009	BGCE Honor's program participant, TU München
09/2007 – 12/2009	TEV-DAAD Scholarship, TU München
06/2007	Second place student award in physics, Koç University
09/2002 – 06/2007	Full merit scholarship, Koç University
05/1994	Top 50 in National High School Entrance Exam, Turkey

Scientific Publications

Ö. Demirel, and I. F. Sbalzarini. Distributed dynamic load balancing in massively parallel scientific numerical simulations. Submitted to *Supercomputing*, April 2014.

Ö. Demirel, I. Smal, W. Niessen, E. Meijering, and I. F. Sbalzarini. Adaptive distributed resampling algorithm with non-proportional allocation. *IEEE Int. Conf. on Acoustics, Speech, and Signal Processing* (ICASSP), pp. 1654-1658, Florence, Italy, May 2014.

Ö. Demirel, I. Smal, W. Niessen, E. Meijering, and I. F. Sbalzarini. PPF - a parallel particle filtering framework. *Data Fusion & Target Tracking Conference*, Liverpool, UK, April 30, 2014.

Ö. Demirel, I. Smal, W. Niessen, E. Meijering, and I. F. Sbalzarini. Piecewise constant sequential importance sampling for fast particle filtering. *Data Fusion & Target Tracking Conference*, Liverpool, UK, April 30, 2014.

N. Fiétier, **Ö. Demirel**, and I. F. Sbalzarini. A meshless particle method for Poisson and diffusion problems with discontinuous coefficients and inhomogeneous boundary conditions. *SIAM J. Scientific Computing*, 35(6), 2013.

Ö. Demirel, B. Schrader, and I. F. Sbalzarini. A parallel particle method for solving the EEG source localization forward problem. *In Proc. 6th Intl. Symp. Health Informatics and Bioinformatics (HIBIT)*, pp. 154-158, Izmir, Turkey, May 2011.

O. Awile, **Ö. Demirel**, and I. F. Sbalzarini. Toward an object-oriented core of the PPM library. *In Proc. of the ICNAAM, International Conference on Numerical Analysis and Applied Mathematics*, 2010.

B. Cankurt, **Ö. Demirel**, Y. Li, D. Mudigere, and K. Rahnema. Inverted pendulum control demonstrator. *BGCE Honor's Project Report*, TU München, January 2009.

Ö. Demirel. The art of quantum cryptography. *Bilim ve Teknik*, vol. 477, pp. 64-66, 2007.