Thesis

HPC Multiscale Modelling and Simulation of Volcanic Tephra Related Processes

KUENZLI, Pierre

Abstract

Explosive volcanic eruptions are typically associated with the injection into the atmosphere of large numbers of particles (known as tephra) dispersed over long distances depending on their size, shape and density. In this thesis, we present the MPI parallel Eulerian-Lagrangian Tephra Transport and Deposition Model (TTDM) Tetras. We build on this model a flexible multiscale TTDM, coupled using MUSCLE-HPC. This model accounts for processes at multiple temporal and spatial scales, such as fast rising of particles in the volcanic column, aggregation of particles and transport of particles by wind over very long range. We propose as well a new family of models called Spatially Extended Exit Rate (SEER) models to improve agreement with field data of proximal deposit compared to Tetras modelling strategy. We propose four SEER based models, ranging from an analytical expression to a compute intensive Lagrangian MPI parallel model. Finally, we investigate new inversion strategies based either on a semi analytical SEER model and metaheuristics or on Tetras coupled with the parallel approximate Bayesian computation framework ABCpy. [...]

Reference

KUENZLI, Pierre. *HPC Multiscale Modelling and Simulation of Volcanic Tephra Related Processes*. Thèse de doctorat : Univ. Genève, 2021, no. Sc. 5600

DOI : 10.13097/archive-ouverte/unige:156386
URN : urn:nbn:ch:unige-1563860

UNIVERSITÉ DE GENÈVE
Départment d'informatique

FACULTÉ DES SCIENCES
Professeur Bastien Chopard
Docteur Jean-Luc Falcone

HPC Multiscale Modelling and Simulation of Volcanic Tephra Related Processes

THÈSE

présentée à la Faculté des sciences de l'Université de Genève
pour obtenir le grade de Docteur ès sciences, mention informatique

par
Pierre KÜNZLI
de
Genève (Genève, Suisse)

Thèse N^o 5600

GENÈVE
Repro-Mail - Université de Genève
2021

UNIVERSITÉ
DE GENÈVE

FACULTÉ DES SCIENCES

DOCTORAT ÈS SCIENCES, MENTION INFORMATIQUE

Thèse de Monsieur Pierre KÜNZLI

intitulée :

«HPC Multiscale Modelling and Simulation of Volcanic Tephra Related Processes»

La Faculté des sciences, sur le préavis de Monsieur B. CHOPARD, professeur ordinaire et directeur de thèse (Département d'informatique), Monsieur J.-L. FALCONE, docteur et codirecteur de thèse (Département d'informatique), Madame C. BONADONNA, professeure ordinaire (Département des sciences de la Terre), Monsieur E. ROSSI, docteur (Département des sciences de la Terre), Monsieur D. GROEN, docteur (Department of Computer Science, University of Brunel, London United Kingdom), autorise l'impression de la présente thèse, sans exprimer d'opinion sur les propositions qui y sont énoncées.

Genève, le 13 octobre 2021

Thèse - 5600 -

Le Doyen

N.B. - La thèse doit porter la déclaration précédente et remplir les conditions énumérées dans les "Informations relatives aux thèses de doctorat à l'Université de Genève".

À Vincent.

Remerciements

Tout d'abord merci au professeur Bastien Chopard et au docteur Jean-Luc Falcone, qui m'ont accueilli et suivi au sein du SPC depuis mon master. Merci particulièrement à eux de m'avoir fait confiance et laissé le temps dont j'avais besoin pour arriver au bout de cette démarche.

Merci ensuite à la professeure Costanza Bonadonna et au docteur Eduardo Rossi pour leurs conseils et pour avoir accepté d'écouter les questions et les idées d'un néophite en volcanologie. Merci également au professeur Ritabrata Dutta pour son accueil qui aura débouché sur une fructueuse collaboration dans le domaine de l'inférence bayésienne et de la cartographie de randonnée de l'arrière-pays anglais. Je tiens également à remercier le docteur Derek Groen d'avoir accepté de porter attention à mon travail en faisant partie de mon jury de thèse.

Mon voyage académique n'aurait évidemment pas été le même sans les membres du groupe SPC et ses satellites, que j'ai eu l'occasion de côtoyer au fil du temps avec différents niveaux de proximité scientifique et personnelle. Merci (dans le désordre) à Christophe, Christophe, Xavier, Federico, Aziza, Christos, Francesco, Sha Li, Jonas, Raphaël, Anthony, Jonathan, Franck, Mohamed, Orestis, Rémy, Joël, Petr.

Merci aussi à mes collègues passés et présents d'hepia. Tout cela n'aurait certainement pas été pareil sans eux. Une pensée spéciale pour Adrien, mon ancien compagnon d'infortune de master et à Mickaël, qui a donné l'étincelle dont l'association Informasciences avait besoin pour voir le jour.

Je dois ensuite des remerciements particuliers à mes amis pour avoir été présent avec une infaillible constance. Merci Paul de m'avoir initié aux joies du calcul parallèle, qui constitue le coeur du travail ici présent et pour les années de coaching sportif mutuel. Merci Jon pour avoir tenté de me pousser à tirer le meilleur parti de mes biceps et avoir été un compagnon de jeu infatigable, merci Jay et Guy pour les pérégrinations électroniques, les camps informatiques et les game jam, merci Gaël d'avoir arrêté de boire, merci Massimo pour les inimitables imitations. Merci Amélie pour les festivals poussiéreux et pour m'avoir supporté et supporté pendant tant d'années.

Remerciements

Merci Gregor et Yann pour les conciliabules du coin café, les parties de Go au Nazar et la traversée de la Sibérie. Merci Loïc pour les bons mots. Merci Léa pour le bref accueil en Bretagne, où j'ai bien failli perdre plus d'une oreille. Merci enfin Aurélia pour m'avoir soutenu et accompagné durant les derniers mois de ce parcours, enduré mes sautes d'humeur et poussé à aiguiser encore un peu plus mes papilles. Toutes et tous ont étés, à un moment ou un autre, des appuis d'une importance majeure.

Et pour finir, je dois un immense merci à mes parents, Bernard et Élisabeth. Merci pour le soutien inaltérable et pour avoir toujours cherché à m'offrir un cadre me permettant de réaliser mes projets.

Abstract

Volcanic eruptions are dramatic events that can greatly endanger human lives, disrupt human activities and impact climate on a global scale. Explosive volcanic eruptions are typically associated with the injection into the atmosphere of large numbers of particles (known as tephra) dispersed over long distances depending on their size, shape and density. Accumulation at ground level of few millimetres to tens of centimetres of tephra can cause a wide range of hazards including the collapse of roofs, damage to crops, killing of many grazing animals, contamination of water supplies, disruption of electricity and telecommunication networks and perturbation of ground transportation, while fine volcanic ash significantly threatens aviation operations and human health. Consequently, even though other volcanic phenomena are more likely to directly endanger human life (e.g., pyroclastic flows and lahars), tephra dispersal and sedimentation can seriously impact entire economic sectors and disrupt critical infrastructure services.

Computer models that simulate the transport in the atmosphere and deposition of tephra are called VATDMs (Volcanic Ash Transport and Dispersion Models) or TTDMs (Tephra Transport and Dispersion Models). They typically describe particle motion in a turbulent velocity field. Volcanic particles are advected inside this field from the moment they leave the vent of the volcano until they deposit on the ground. Volcanic eruptions involve phenomena occurring at multiple temporal and spatial scales, such as fast rising of particles inside the volcanic column, aggregation of fine particles into larger ones that tends to fall faster or transport of fine particles over very long range. Volcanic eruptions are thus multiscale phenomena, which needs multiscale-multiscience models to be fully depicted.

MMSF (Multiscale Modelling and Simulation Framework) is a set of tools and techniques that allow to model, describe and simulate multiscale and multiscience phenomena in a standardized way. In this thesis, we focus on the design and implementation of tephra transport models as multiscale applications. Each model is described as a coupling of submodels using MMSF formalism. Tetras (TEphra TRAnsport Simulator) is a flexible simulation tool based on a hybrid Eulerian-Lagrangian numerical model. As this kind of model needs computationally intensive simulations, a parallel version on a distributed memory architecture using MPI was developed. Then, we developed a multiscale implementation of the software. This implementation combines short and long-range transport of particles as well as particle aggregation.

Abstract

A strong plume is a volcanic plume where the volcanic mixture first rises mainly vertically before spreading horizontally to form an umbrella cloud at the level of neutral buoyancy. The region of transition between this mainly vertical and mainly horizontal regime is called the plume corner. A drawback of Tetras modelling strategy is its tendency to underestimate deposits of particles in the plume corner area. Thus, we investigated how to better characterize the source term (the description of the insertion of particles in the atmosphere) of the transport model and propose the Spatially Extended Exit Rate (SEER) model where flux of particles is injected in the atmosphere from the border of a three-dimensional volcanic plume. We built on this model volcano-lagrangian-seer, which is a MPI parallel TTDM based on a Lagrangian numerical scheme and volcano-semianalytical-seer, which is a simplified version of the model offering fast solving.

Forward volcanic ash simulation models take Eruption Source Parameters (ESP) as input such as initial temperature and velocity of the plume, particle size and density distributions and atmospheric conditions and produces a ground deposition pattern and atmospheric concentration of particles. The inverse problem is to consider the observed tephra accumulation to reconstruct ESP. To achieve this, volcanologists usually use optimization algorithms combined with a fast-solving semianalytical forward model. We propose an inversion strategy based on parallel approximate Bayesian computation algorithms and our parallel MPI model Tetras. ABCpy is a parallel Approximate Bayesian Computation framework, which allows running hundreds or thousands of instances of a forward model to infer its parameters. To perform inversion with ABCpy and Tetras, we developed a nested parallelization strategy which allows integrating MPI parallel models in ABCpy. This opens the way of inversion using heavy MPI parallel forward models on high performance computing infrastructure. Finally, we started investigating inversion using volcano-semianalytical-seer and the Nelder-Mead simplex algorithm. This could lead to hybrid inversion strategies, where the parameter space is first largely explored with a fast analytical or semi-analytical model to give a starting point for a more local optimization using a heavy sophisticated parallel model.

Résumé

Les éruptions volcaniques sont des événements spectaculaires qui peuvent grandement mettre en danger des vies humaines, perturber les activités humaines et avoir un impact sur le climat à une échelle globale. Les éruptions volcaniques explosives vont généralement de pair avec l'injection dans l'atmosphère d'un grand nombre de particules (appelées téphra) dispersées sur de longues distances en fonction de leur taille, forme et densité. L'accumulation sur le sol de quelques millimètres à plusieurs dizaines de centimètres de téphra peut causer un grand nombre de dangers comprenant l'effondrement de toits, des dommages aux cultures, la mort d'animaux de pâturage, la contamination de sources d'approvisionnement en eau, la perturbation des réseaux électriques et de communication et la perturbation des transports terrestres, tandis que les fines particules volcaniques menacent fortement l'aviation et la santé humaine. Par conséquent, même si d'autres phénomènes volcaniques sont plus susceptibles de mettre directement en danger des vies humaines (par exemple les nuages pyroclastiques ou les lahars), la dispersion et la sédimentation de téphra peut gravement impacter des secteurs économiques entiers et perturber des infrastructures critiques.

Les modèles informatiques simulant le transport dans l'atmosphère et le dépôt de téphra sont nommés VATDM (Volcanic Ash Transport and Dispersion Models) ou TTDM (Tephra Transport and Dispersion Models). Ils décrivent typiquement le mouvement des particules dans un champ de vitesses turbulent. Les particules volcaniques sont advectées dans ce champ dès le moment où elles s'échappent du cratère jusqu'à ce qu'elles se déposent au sol. Les éruptions volcaniques impliquent des phénomènes se produisant à de multiples échelles de temps spatiales et temporelles, telles que l'ascension rapide des particules dans la colonne volcanique, l'agrégation de particules fines en particules plus grandes qui ont tendance à tomber plus vite ou encore le transport de particules fines sur de très longues distances. Les éruptions volcaniques sont donc des phénomènes multiéchelle, qui nécessitent des modèles multiscience et multiéchelle pour être complètement représentés.

MMSF (Multiscale Modelling and Simulation Framework) est un ensemble d'outils et de techniques permettant de modéliser, décrire et simuler des phénomènes multiscience et multiéchelle de façon standardisée. Dans cette thèse, on se concentre sur la conception et l'implémentation de modèles de transport de téphra en tant qu'applications multiéchelle. Chaque modèle est décrit sous la forme d'un couplage de sous modèles en utilisant le formalisme de MMSF. Tetras (TEphra TRAnsport Simulator) et un outil de simulation souple basé

sur un modèle numérique hybride eulérien-lagrangien. Puisque ce type de modèle nécessite des simulations numériquement intensives, une version parallèle sur architecture à mémoire distribuée utilisant MPI a été développée. Ensuite, nous avons développé une implémentation multiéchelle du logiciel. Cette implémentation combine le transport à courte et longue distance ainsi que l'agrégation de particules.

Un "panache fort" (ou "strong plume") est un panache volcanique ou le mélange volcanique s'élève principalement verticalement avant de s'étendre horizontalement pour former un "nuage parasol" (ou "umbrella cloud") au niveau de poussée neutre (ou "neutral buoyancy level"). La région de transition entre ces deux régimes est appelée le "coin du panache" (ou "plume corner"). Un inconvénient de la stratégie de modélisation de Tetras est sa tendance à sous-estimer les dépôts de particules en dessous de cette région. Ainsi, nous avons cherché à mieux caractériser le terme source (ou "source term", la description de l'insertion des particules dans l'atmosphère) du modèle de transport et nous proposons le modèle "Spatially Extended Exit Rate" (SEER) où des flux de particules sont injectés dans l'atmosphère depuis le bord d'un panache volcanique en trois dimensions. Nous avons construit sur ce modèle volcano-lagrangian-seer, qui est un TTDM parallélisé avec MPI basé sur un modèle numérique lagrangien et volcano-semianalytical-seer, qui est une version simplifiée du modèle qui permet une résolution rapide.

Les modèles de simulation de cendres volcaniques prennent en entrées des paramètres d'éruption appelés "Eruption Source Parameters (ESP)", tels que la température et vitesse initiale du panache, la taille et la densité des particules et les conditions atmosphériques et produisent un motif de dépôt au sol ainsi que des concentrations atmosphériques de particules. On appelle cela le problème direct. Le problème inverse est de considérer l'accumulation de téphra observée et de reconstruire les ESP. Cette tâche est appelée inversion. Pour réaliser cela, les volcanologues utilisent habituellement des algorithmes d'optimisation combinés à des modèles volcanologiques à résolution rapide. On propose une stratégie d'inversion basée sur des algorithmes d'inférence bayésienne parallèles (Approximate Bayesian Computation, ABC) et notre modèle parallèle MPI Tetras. ABCpy est une infrastructure de développement parallèle implémentant des algorithmes ABC qui permet de résoudre des centaines ou des milliers d'instances d'un problème direct en parallèle afin d'en inférer ses paramètres. Afin de réaliser une procédure d'inversion basée sur ABCpy et Tetras, nous avons développé une stratégie de parallélisation imbriquée permettant d'intégrer des modèles MPI dans ABCpy. Cela ouvre la voie à des procédures d'inversion basées sur des simulations MPI parallèles lourdes sur des infrastructures de calcul haute performance. Finalement, nous avons commencé à étudier une stratégie d'inversion utilisant volcano-semianalytical-seer et l'algorithme du simplexe de Nelder-Mead. Cela pourrait conduire à des stratégies d'inversion hybrides, où l'espace de paramètres est d'abord exploré largement avec un modèle analytique ou semi-analytique rapide afin de donner un point de départ pour une optimisation plus locale utilisant un modèle parallèle sophistiqué.

Contents

1 Introduction

Volcanic eruptions can be separated in two main types : effusive and explosive eruptions. Effusive eruptions are the most frequent types of eruption, showing large discharge of lava. On the other hand, explosive volcanic eruptions inject into the atmosphere large amount of pyroclastic material (including tephra), ranging from fine ash that can remain in the atmosphere for days to weeks, to large volcanic bombs.

Explosive volcanic eruptions are typically associated with the injection into the atmosphere of large amounts of tephra dispersed over long distances depending on size, shape and density. Accumulation at ground level of few millimetres to tens of centimetres of tephra can cause a wide range of hazards including the collapse of roofs, damage to crops, killing of many grazing animals, contamination of water supplies, disruption of electricity and telecommunication networks and perturbation of ground transportation, while fine volcanic ash significantly threatens aviation operations and human health [73, 66, 33, 118, 121, 122].

Consequently, even though other volcanic phenomena are more likely to directly endanger human life (e.g., pyroclastic density currents and lahars), tephra dispersal and sedimentation can seriously impact entire economic sectors and disrupt critical infrastructure services, as demonstrated by the eruptions of Eyjafjallajökull (Iceland; 2010 [65]) and Cordón Caulle (Chile; 2011 [120]) volcanoes.

1.1 Modelling and simulation of volcanic tephra dispersal

Mathematical models have been used for centuries in science. More recently, usage of computational models have dramatically increased, and it is now one of the main techniques used to model real world phenomena. Numerical models (also called computer simulations or computational models) are tools that aims to mimic *in silico* (by the use of a numerical computer) real life phenomena. Numerical models are developed with various levels of complexity. More complexity in general involves the necessity of more computing power.

Computers are increasingly powerful, but this increase is now largely dominated by the

increase in the level of parallelism in computer architectures (the number of computing cores in a computer). Thus, to benefit from technological improvement in computer design, software, and specifically simulation software, have to be parallelized in an efficient way. Computer simulations are developed with various goals in mind, which includes a better understanding of the modelled physics, forecasting of natural phenomena, such as weather forecasting, or for engineering purpose, to design new tools and techniques.

Computer models that aim to reproduce tephra transport and deposition in the case of a volcanic eruption are called VATDMs (Volcanic Ash Transport and Dispersal Models) or TTDMs (Tephra Transport and Dispersal Models)[58]. During the past decades, several TTDMs have been developed with variable levels of complexity and different objectives including a better understanding of particle transport and sedimentation dynamics; the compilation of real time and long-term hazard assessment associated with both ground load and atmospheric concentrations of ash and the determination of eruption source parameters through inversion strategies (see [58, 23, 24] for a review).

A classic way of modelling tephra transport and deposition is to apply the advection-diffusion-sedimentation equation to tephra particles released in the atmosphere from a source term (the description of quantity, time and position of injection of particles in the atmosphere) and subject to atmospheric condition, such as wind. Models exist with a variety of semi-analytical models (Tephra 2 [39, 22]), Eulerian numerical models (Fall3D [61, 94], Ash3D [102]), Lagrangian or hybrid Lagrangian models (NAME [74], Hysplit [106], Vol-calpuff [7, 6], ATLAS [97]).

Models describing movement of bulk eruption materials exist for volcanic plumes and umbrella cloud. Such as integral plume models based on buoyant plume theory (BPT [87]) (Woods model [123], Degruyter and Bonadonna model [49, 48], FPLUME [60], PLUME-MoM-TSM [86]). Those models can be used to describe the density of particles along the plume to construct a source term. The difficulty resides on the determination of the position and rate of injection of particles in the atmosphere from a volcanic plume, which has an important impact on proximal (regions close to the vent) deposition of particles. 3D fully resolved model exists as well for volcanic plumes, such as ATHAM [90, 70, 71] and ASHEE [34]. While those models are of great interest to study the physics of eruption columns, they are too computationally heavy to be suitable for real time hazard assessment or eruption source parameters inversion.

1.2 A non-exhaustive review of TTDMs

Fall3D [61, 41, 59] is a three-dimensional parallel Eulerian model developed at the Barcelona Supercomputing Center solving the advection-diffusion-sedimentation (ADS) equation. The source term can be defined as a point, a uniform vertical line, a Suzuki distribution [109], a plume model based on buoyant plume theory or a resuspension scheme. It has been originally designed for volcanic tephra, but has been extended to other types of particles, aerosols or radionuclides and can be coupled to external meteorological models. It can compute

atmosphere particle concentration as well as particles deposits.

ATLAS [97] is a Lagrangian particle transport model. It shares the same physics as Fall3D and can be viewed as a Lagrangian implementation of this model. The source term can be as well defined as a point, a uniform vertical line or a Suzuki distribution. There is also a resuspension scheme. Those different source terms can be combined within a given simulation. Aggregation is handled in a simple way, by modifying the initial TGSD. This code is not parallelized.

Ash3D [102] is an Eulerian model developed by the U.S. Geological Survey that simulates the transport and deposition of tephra. Particles are injected in domain cells, either in a single cell, linearly distributed above the volcano, or following a Suzuki distribution. The model is coupled with external numerical weather prediction models to compute the trajectories of particles. The model can output atmospheric concentration of particles or ground mass load.

NAME (Numerical Atmospheric Dispersion modelling Environment) [74, 12, 11] is a Lagrangian particles model of the London Volcanic Ash Advisory Centre (VAAC). It is used to provide real-time forecasts during volcanic eruptions, such as the 2010 eruption of Eyjafjallajökull volcano. It can be initialized using a variety of source terms, such as a vertical distribution of particles above the vent, or using a BPT model with injection of particles at the top of the plume.

Vol-calpuff [7] is a Lagrangian puff model based on a plume model that accounts for wind entrainment and particle sedimentation from plume border. Particles are injected at the top of the plume in the form of puffs.

Tephra2 [39, 22] is a semi-analytical model where particles are assumed to be distributed in a line above the vent with a given density distribution. The atmosphere is divided in layers that can account for user-submitted wind data. Particles bins are supposed to be released at their given height, and the ADS equation is solved to compute where the particles bins will fall on the ground. The model has been parallelized with MPI, and it's simplicity makes that it's widely used for eruption source parameters (ESP) inversion [38].

1.3 Multiscale modelling

Volcanic eruptions are complex phenomena implying process at multiple time and space scales. When volcanoes erupt, they inject in the atmosphere a wide range of materials, including gas and solid particles which interacts in complex ways. Large particles are ejected from the vent, and follow ballistic trajectories. The mixture of small particles and gas can interact, forming a volcanic plume. Larger particles can fall from plume borders, and then travel distances for hundreds of meters to tenth of kilometres. Fine ash can be transported tens of thousands of kilometres away, or even at global scale, remaining in the atmosphere for weeks to months. They can also aggregate and form larger particles, which falls closer to the vent. A full volcano model is thus a multiscale model, where processes of multiple spatial and

temporal scales interact to form a global process.

Models of multiple levels of complexity have been developed by scientists involved in the research fields of volcanology, atmospheric sciences and physics. Each model relies on different assumptions or simplifications. They are expressed in different forms, ranging from an analytical expression, solved almost instantly, to heavy numerical simulations requiring powerful supercomputers to be solved in a reasonable time, and account for various number of physical processes within each model. But they have the common characteristics, independently from their internal formulation, of taking various number of parameters as input, and output values for various number of variables of interest. Every output variable from a model can be considered as input for another model. The construction of a multiscale model then consists of defining how submodels exchange their data and on which computing resources they are solved.

The problem of designing multiscale models where submodels with various characteristics and scales interact is not limited to earth sciences, but arise to every other modelling science. In the past decade, efforts have been made to formally define interaction of multiscale submodels and develop tools and frameworks which allows for the design and implementation of multi-scale multi-physics applications. This is the case of MMSF (Multiscale Modelling Software and Framework) [27, 35] which we use in this work to design our multiscale volcano application. This methodology allows for a clear view of the different parts of a multiscale model, and make it easier to make submodels evolve without changing the whole structure of the overall model. Moreover, computational load can be very different between submodels, which can raise efficiency problems if submodels are not attributed to computing resources in a clever way. It is thus important to model performances of submodels in order to predict their execution time to maximize the usage efficiency of computing resources.

1.4 Inversion

Inversion is the task of optimizing input parameters of a model (called the forward model) to obtain best fit of the model output with field measurements. For a volcanic eruption, this allows estimating eruptions source parameters (ESP) such as plume height, initial plume speed, radius and temperature, total erupted mass, eruption duration or particle characteristics, or even atmospheric conditions [38]. Inversion is generally done through nonlinear optimization algorithms, such as metaheuristics, or statistical inference algorithms coupled with light numerical model. For example, [38] uses Tephra2 coupled with the Nelder-Mead simplex algorithm and [124] uses Tephra2 as well, but coupled with the Metropolis–Hastings algorithm. Those techniques work by running a lot of instances of the forward model (usually thousands) and optimizing input parameters by exploration of the parameter space.

With modern supercomputers, it becomes possible to use mediumly heavy numerical models as forward model for inversion [55, 91]. This implies either that the forward model is parallelized and can be solved very quickly thanks to a good strong scaling behaviour (i.e.,

that the solving time becomes lower when adding computing resources when the size of the problem remains the same), or that the optimization scheme is itself parallel and allows running hundreds or thousands of instances of the forward model at the same time, which is the case of a number of approximate Bayesian computation algorithms.

1.5 High performance computing systems used within this work

High performance computing systems for research comes in a great range of computing power and availability, ranging from small clusters private to research groups, to large supercomputers often available at a national level. The need for a specific level of computing power depends on the considered problem, and the capacity of the simulations tools to scale with the number of the computing cores[1] (i.e., to be able to efficiently use the computing resources). We give a short description of the three HPC systems that have been used within this work.

Baobab and **Yggdrasil** are the high performance computing facilities of the Geneva's higher education institutions, hosted at the University of Geneva. They are heterogeneous clusters, equipped with public nodes available to any user of the systems and private nodes, accessible with higher priority to the owners. **Baobab** is running since 2013 and **Yggdrasil** since 2021.

Piz Daint is a supercomputer of the Swiss National Supercomputing Centre (CSCS) located in Lugano. It is a Cray machine available since 2012, and has been updated several times since then. It is, at the time of writing (July 2021), ranked 15th in the TOP500 list [2].

- **Baobab** is compounded of 54 public nodes equipped with 16 -core Intel E5-2660V0 @ 2.20 GHz for a total of 864 public cores. There are also approximately 140 private nodes equipped only with CPUs and 20 private nodes equipped with GPUs.
- **Yggdrasil** is compounded of 81 public nodes equipped with 36 -core Intel Xeon Gold 6240 @ 2.60 GHz and 4 public nodes equipped with 16 cores Intel Xeon Gold 6244 @ 3.60Gz. There are as well 7 public nodes equipped with GPUs and 16-core Intel Xeon Silver 4208 @ 2.10 GHz. This makes a total of 3092 publicly available cores. There are also approximately 30 private CPU nodes.
- **Piz Daint** is compounded of 5074 computing nodes equipped with 12-core Intel Xeon E5-2690 v3 @ 2.60 GHz and a GPU NVIDIA Tesla P100 and 1813 nodes equipped with 2×18 -core Intel Xeon E5-2695 v4 @ 2.1GHz for a total of $126'156$ cores.

[1] Modern processor architectures are usually multicore. To avoid ambiguity, we use the term "core" or "computing core" to denote a sequential computing unit.

[2] https://www.top500.org/

1.6 Content of the thesis

In this thesis, we focus on the modelling and simulation of transport of volcanic tephra as a multiscale phenomenon. While tephra accounts for particles of any size, we focus on small sizes particles, for which the movement can be considered has a transport within an ambient environment, rather than ballistic trajectories.

First, we present the TTDM Tetras (TEphra TRANsport Simulator), which is based on a hybrid Lagrangian-Eulerian numerical model. Its hybrid representation allows for particle interaction and outputting of atmospheric ash concentration at any given time. The modelling strategy is based on a point source term located at the vent. Then, particles are transported according to a velocity field described by an integral plume model, an umbrella cloud model, wind velocity and sedimentation velocity. A parallelization on distributed architecture using MPI has been developed and a performance model proposed. We developed the first version of this simulation tool and modelled its performances before the present work. The first version of Tetras integrated only a model for strong plumes, we extend the capacity of the tool by integrating a plume model suited for bent over plumes. We published this work in [80]. We describe as well Tetras in terms of a multiscale model, which allows us to introduce the MMSF formalism.

Then, while the deposit in the plume corner area is underestimated by Tetras modelling strategy, we propose the Spatially Extended Exit Rate source term model. With this model, particles are injected along a three-dimensional volcanic plume and from the base of the umbrella cloud at specific rates which depends on the geometry of the plume. We build on this formulation two TTDMs that aims at reproducing tephra deposits : volcano-semianalytical-seer which is a fast-solving semianalytical model and volcano-lagrangian-seer which is a heavy MPI parallelized Lagrangian particles model. This latter has a good strong scaling behaviour, which makes it potentially usable in sequential inversion schemes.

Then, we investigate ESP inversion strategies using the Approximate Bayesian Computation framework ABCpy coupled with Tetras. For this, we implement a nested parallelism scheme which opens the way of inversion using ABC techniques in conjunction with MPI parallelized models. This work has been published in [91, 55]. We provide as well preliminary inversion results using the Nelder-Mead simplex algorithm applied to volcano-semianalytical-seer.

Finally, we describe a full multiscale tephra transport and aggregation application. We use MMSF tools and techniques to design and implements our application. We couple an aggregation model with the transport model to account for this phenomenon in the volcanic plume, and we couple transport models at multiple scales to account for transport of particles at local and global scales. We observe as well the impact of submodel placements on computing resources and propose a technique based on Discrete Event Simulation to model performances of a multiscale model. This work has been published in [79, 37].

The document is structured as follows :

- **Chapter 1** gives an introduction to the field of numerical simulations and volcanic tephra transport simulations and defines the scope of this work.
- **Chapter 2** present the TTDM Tetras, the related physical models on which it relies as well as the MMSF formalism that is used in the rest of the thesis. We provide comparison with field data as well as validation of the performance model.
- **Chapter 3** present the Spatially Extended Exit Rate (SEER) family of models. We present the TTDMs based on this source term and observe the scaling behaviour of volcano-lagrangian-seer, the Lagrangian TTDM based on SEER source term.
- **Chapter 4** describes the coupling of Tetras with ABCpy to perform ESP inversion. We provide as well preliminary results of inversion using volcano-semianalytical-seer coupled with the Nelder-Mead simplex algorithm.
- **Chapter 5** gives more details about the implementation of Tetras and volcano-lagrangian-seer. We present as well the design and implementation of the full multiscale transport and aggregation model. We describe how the aggregation model is coupled with the transport model and investigates the impact on performances of the multiscale coupling.
- **Chapter 6** summarizes the work presented in this thesis and gives perspective for future work. We discuss how the proposed multiscale model could evolve, and how it could be coupled with cutting edge volcanological models recently published.

A large part of the work presented in this thesis has been published in the following journals and international conferences publications :

- P. Künzli, K. Tsunematsu, P. Albuquerque, J.-L. Falcone, B. Chopard, and C. Bonadonna. “Parallel simulation of particle transport in an advection field applied to volcanic explosive eruptions”. In: *Computers & Geosciences* 89 (2016), pp. 174–185. DOI: https://doi.org/10.1016/j.cageo.2016.02.005
- P. Künzli, J.-L. Falcone, E. Rossi, P. Albuquerque, and B. Chopard. “HPC Multiscale Simulation of Transport and Aggregation of Volcanic Particles”. In: *2018 17th International Symposium on Parallel and Distributed Computing (ISPDC)*. 2018 17th International Symposium on Parallel and Distributed Computing (ISPDC). June 2018, pp. 25–32. DOI: 10.1109/ISPDC2018.2018.00013
- B. Chopard, J.-L. Falcone, P. Kunzli, L. Veen, and A. Hoekstra. “Multiscale modeling: recent progress and open questions”. In: *Multiscale and Multidisciplinary Modeling, Experiments and Design* 1.1 (2018), pp. 57–68. DOI: 10.1007/s41939-017-0006-4

- L. Pacchiardi, P. Künzli, M. Schöngens, B. Chopard, and R. Dutta. “Distance-learning For Approximate Bayesian Computation To Model a Volcanic Eruption”. In: *Sankhya B* (Jan. 2020). DOI: 10.1007/s13571-019-00208-8

- R. Dutta, M. Schoengens, L. Pacchiardi, A. Ummadisingu, N. Widmer, P. Künzli, J.-P. Onnela, and A. Mira. “ABCpy: A High-Performance Computing Perspective to Approximate Bayesian Computation”. In: *arXiv:1711.04694 [stat]* (Feb. 25, 2021)

The content of those papers has sometimes been reorganized for the purpose of this document and enriched with novel results. Details are given at the beginning of each chapter.

2 The tephra transport model Tetras

The work in this chapter has been published in

P. Künzli, K. Tsunematsu, P. Albuquerque, J.-L. Falcone, B. Chopard, and C. Bonadonna. "Parallel simulation of particle transport in an advection field applied to volcanic explosive eruptions". In: *Computers & Geosciences* 89 (2016), pp. 174–185. DOI: https://doi.org/10.1016/j.cageo.2016.02.005.

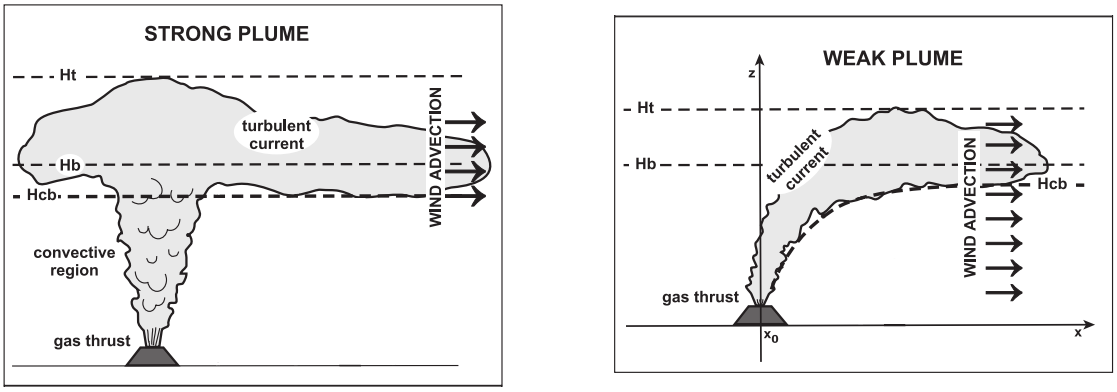
Compared to the published version, the work has been enriched with coupling structure description based on the gMML (Graphical Multiscale Modelling Language) and SSM (Scale Separation Map) formalisms. The physical models are as well described in more details.

2.1 Introduction

Most existing TTDMs are either Eulerian finite difference schemes such as FALL3D [61, 94] or Lagrangian particle-puff schemes such as VOL-CALPUFF [7, 6] or NAME III [74]. Some are pure Lagrangian models such as PUFF [104] or ATLAS [97].

Pure Lagrangian models dealing with point particles have the advantage of allowing the implementation of individual particle behaviour as well as particle interaction. Moreover, they remain unconditionally stable, unlike for example finite difference schemes, and allow monitoring ash concentration in the atmosphere as well as ground mass load at any given time.

The drawback is that such representations require the simulation of large numbers of particles to yield statistically accurate results. This may thus produce very computationally intensive simulations. Although the increasing computational capacities of computers and supercomputers tend to make these simulations affordable, parallelization still remains essential to run them on large numbers of computing cores. However, pure Lagrangian models are difficult to implement on massively multicore systems when particle interactions are involved. This is why we propose an implementation based on a hybrid model.



(a) (b)

Figure 2.1: Representation of the main features of (a) a strong plume and (b) a weak plume (adjusted from [20]). H_b: Neutral buoyancy level; H_t: total plume height; H_{cb}: base of the umbrella cloud.

In this chapter, we present a hybrid Eulerian-Lagrangian model for the dispersal and sedimentation of tephra with an emphasis on computational efficiency and model flexibility. Computational efficiency is particularly important for both real time forecasting and long-term hazard assessments. In our approach, particles remain linked to a site and their exact positions are continuously tracked. This allows for an accurate treatment of diffusion while maintaining a known maximum spatial integration step. The associated data structure facilitates a parallel implementation as well as the addition of complex sedimentation processes, such as particle aggregation [29, 42, 11].

The implementation was designed with a modular approach to permit modification or addition of components to the physical model. The modular design is presented based on the gMML and SSM formalisms, which are part of MMSF, the Multiscale Modelling and Simulation Framework [14, 35, 56]. It is then possible to use this simulation tool both for the study of the dynamics of particle dispersion and as a prediction tool for hazard assessment. As previously said, our model allows to easily monitor both atmospheric ash concentration and ground mass load. In this work, we only focus on the latter.

2.2 Model description

During explosive volcanic eruptions, a hot mixture of particles and volcanic gases is typically ejected with an initial density several times larger than in the atmosphere, and rises due to momentum. As the ejected material entrains ambient air, the mixture density drastically decreases and the eruptive plume starts rising due to buoyancy. If the plume upward velocity is much larger than the horizontal wind velocity (strong plume, Figure 2.1a), the volcanic plume buoyantly rises up to the neutral buoyancy level (H_b) where it starts spreading laterally

as a gravity current (umbrella cloud). In contrast, if the horizontal wind velocity dominates, the plume bends over above the basal jet before spreading laterally around the neutral buoyancy level (weak plume, Figure 2.1b)[25, 31]. In all cases, cloud spreading results from the interplay between buoyancy and wind advection, with the contribution of buoyancy being proportional to plume height [25, 21, 43].

Depending on their size and density, particles are transported upward by the volcanic plume and, if sufficiently small, might be entrained within the umbrella cloud and sediment according to their terminal velocity. In particular, information on the Total Grain-Size Distribution (TGSD), namely the size distribution of particles injected into the atmosphere, combined with particle density is necessary to initialize the model.

Turbulent effects, which play an important role in this model, are represented through a diffusion process. Different diffusion coefficients are applied in the atmosphere (D_a) and in the plume (D_p), the latter coefficient being usually several times larger.

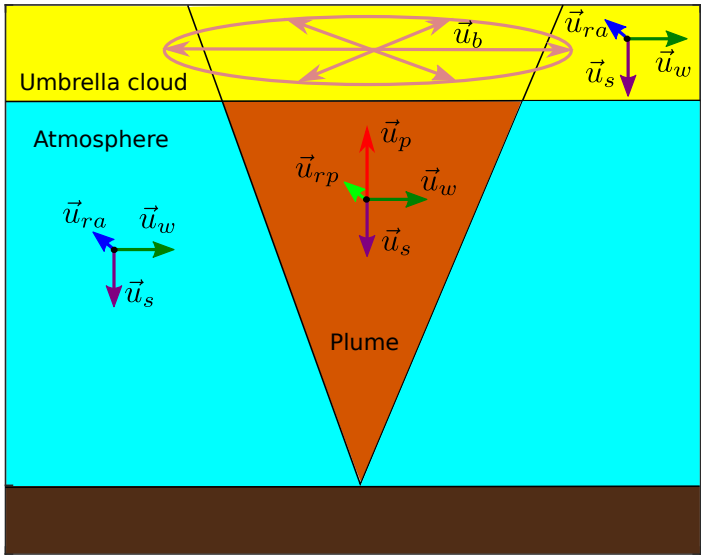


Figure 2.2: Velocity fields considered in the model. Note that velocities are summed to produce the final advection field. For example, in the umbrella cloud, $\vec{u}_b$ and $\vec{u}_w$ will be summed, so the umbrella cloud will be elongated in the downwind direction, as shown in Figure 2.1.

We divide space into three main zones (Figure 2.2), where the velocity fields are described as

- atmosphere : $\vec{u} = \vec{u}_s + \vec{u}_w + \vec{u}_{ra}$;
- volcanic column : $\vec{u} = \vec{u}_s + \vec{u}_w + \vec{u}_p + \vec{u}_{rp}$;
- umbrella cloud : $\vec{u} = \vec{u}_s + \vec{u}_w + \vec{u}_b + \vec{u}_{ra}$;

where

- $\vec{u}_s$ is the settling velocity of the particles [3, 25, 78];
- $\vec{u}_w$ is the wind velocity, retrievable from analytical models [31, 72], weather reanalysis databases or weather forecasting models;
- $\vec{u}_p$ is the plume velocity [49, 123];
- $\vec{u}_b$ is the spreading velocity of the umbrella cloud [100, 25];
- $\vec{u}_{ra}$ and $\vec{u}_{rp}$ are random velocities for simulating diffusion in the atmosphere and in the volcanic column.

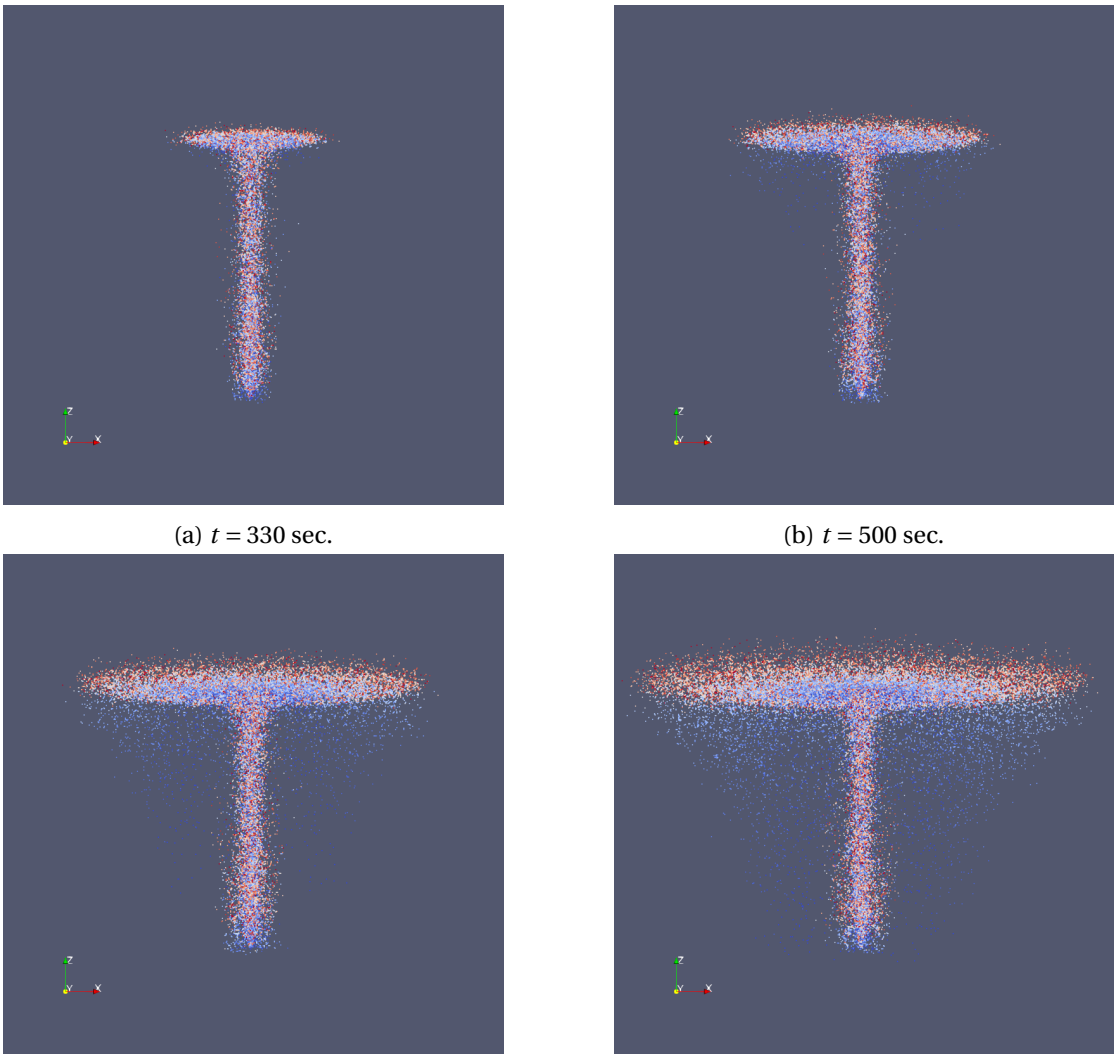
Once a set of parameters is selected, a simulation is run by injecting particles at the vent (we consider a point source term) inside the domain and waiting for particles to either deposits on the ground or leave the domain. Figure 2.3 shows an example of Lagrangian particles during a simulation.

2.2.1 Multiscale formulation

The Multiscale Modelling and Simulation Framework (MMSF) is a methodology and a set of tools to describe multiscale, multiscience phenomena [35, 37, 27, 26]. A multiscale application involves different processes occurring at different spatial and temporal scales. The first step consists of decomposing a phenomenon in multiple single time and space scale processes, called submodels. For example, particles move at the time scale of the second, while atmospheric condition evolves at the time scale of the hour. Here, we make use of the graphical formalisms Scale Separation Map (SSM) and graphical Multiscale Modelling Language (gMML) of the MMSF framework to define the overall model as a set of submodels exchanging data.

The Scale Separation Map

The SSM shows graphically the temporal and spatial scale of each submodel by representing a rectangle whose lower bound is the spatial step, upper bound is the size of the phenomenon, left bound is the time step and right bound is the duration of the phenomenon. Two variations of the transport model have been implemented, whose SSM are shown in Figure 2.4. One version relies on Woods plume model [123] and an analytical approximation for atmospheric



(a) $t = 330$ sec. (b) $t = 500$ sec.

(c) $t = 700$ sec. (d) $t = 930$ sec.

Figure 2.3: A volcanic plume is simulated by the numerical model over 15 minutes (physical time). Colours represent size of particles. Grain size ranges from $\phi = -7$ (blue) to $\phi = 10$ (red), $\phi = -\log_2(d)$ where d is the diameter of the particle in mm.

conditions [123, 72] (variant a, Figure 2.4a). The other one relies on Degruyter plume model [49, 48] and atmosphere data from a reanalysis database (variant b, Figure 2.4b).

Note that while Degruyter plume model is an extension of Woods model accounting for wind, the difference between both versions of the transport model relies on implementation aspects only. Woods model is solved directly within the main transport code written in C++ while Degruyter model is solved by an external Matlab code which results is read by the C++ code. The consequence is that solving of Woods model is much faster than Degruyter, which is important in an inversion context (see Chapters 3 and 4). An improvement would be to write

Degruyter model in C++ in order to integrate it to the main C++ transport code.

The *transport* submodel of our application is the submodel that transport Lagrangian particles in the atmosphere according to an advection field. The time step of this submodel is of the order of the second (left bound of the model in Figure 2.4), and the total duration of the phenomenon is usually days (right bound of the model in Figure 2.4). The lower bound of the spatial scale is defined by the interplay between advection velocities and temporal time step and is limited by the size of the sites of the domain Δx (see Section 2.3, lower bound of the model in Figure 2.4). The upper bound of the model is defined by the size of the simulation domain, which is limited to hundreds of kilometres in this chapter (upper bound of the model in Figure 2.4). Note that the transport of a volcanic ash cloud can span over thousands of kilometres. This case is addressed in Chapter 5 with a multiscale implementation.

In the present model, we identify the following submodels :

Plume Steady state model computing the average velocity of particles in the plume and geometry of the plume.

Transport Transport of Lagrangian particles in the atmosphere according to an advection field.

Advection field Computation of the advection field, based on diffusion coefficients, particle sedimentation velocity, plume velocity, umbrella cloud velocity and wind velocity.

Source term Insert particles at each time step in the transport domain at the vent position.

Atmosphere **(only for variant b)** Atmosphere characteristics from a reanalysis database.

The Submodel Execution Loop

We suppose that each discretized submodel implements a numerical solver in the form of an iterative loop called SEL (Submodel Execution Loop). The steps of such a loop can be abstracted with the following operators:

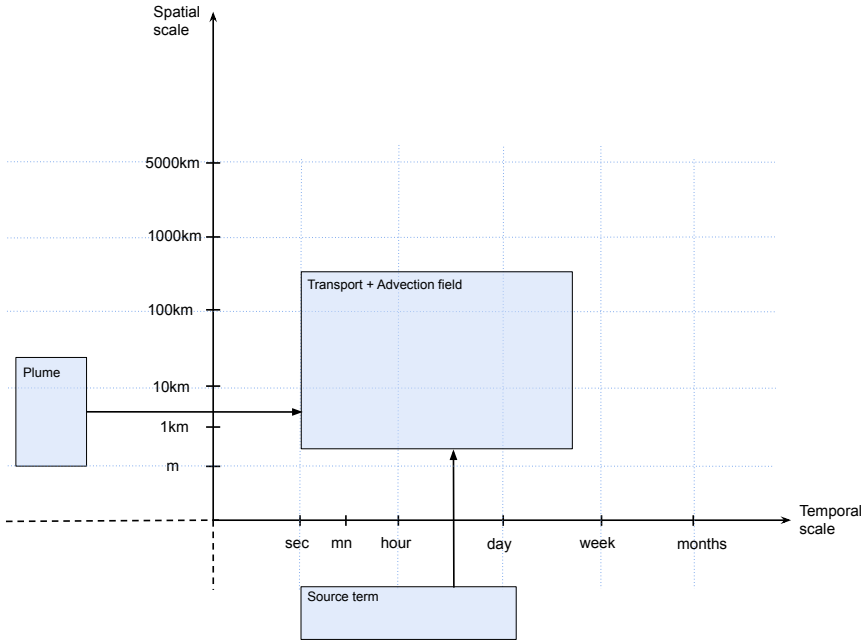
Initialization (F_{init}) This is where the state of the submodel domain is initialized.

Solver (S) This is where the model is applied to the domain.

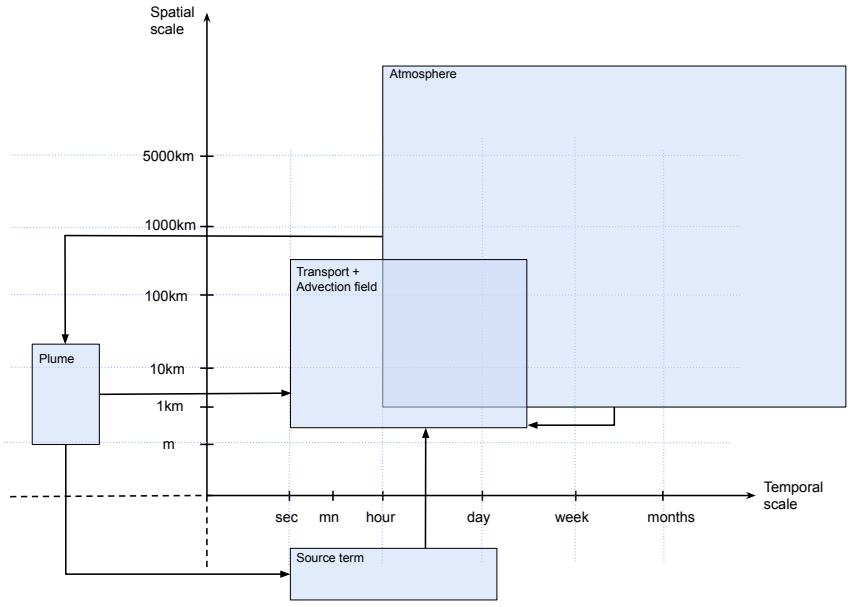
Boundary (B) Is the operator that updates the boundary conditions.

Intermediate observation (O_i) This operator provides the desired physical quantities during the computation.

Final observation (O_f) This operator provides the desired physical quantities at the end of the computation.



(a) SSM showing the submodels when using an analytical approximation for atmosphere conditions. This corresponds to the gMML description in Figure 2.6a.



(b) SSM showing the submodels when using a reanalysis database (here ECMWF-ERA-Interim) for atmosphere conditions. This corresponds to the gMML description in Figure 2.6b.

Figure 2.4: SSM for the tephra transport applications. In both cases, the scale of advection field overlaps with the scale of the transport model. In (a), atmosphere characteristics are embedded in *plume* and *advection field* as a state equation, while in (b) it is a separate submodel. *plume* is a steady-state model, so it does not have a time scale. *source term* is a point in this model, it does not have a spatial scale.

Algorithm 2.1: General pattern of the SEL. f is the state of the submodel domain, F_{init}, S, B, O_i and O_f are the operators of the SEL. F_{init}, S and B operators can modify the state of the domain, while O_i and O_f operators are used to retrieve state of the domain, for example for exchanging data with other submodels through the network, function call or writing results to disk.

```
1 f = f_init();
2 while not finished do
3 |   O_i(f);
4 |   f=S(f);
5 |   f=B(f);
6 end
7 O_f(f);
```

The graphical Multiscale Modelling Language

Submodels can interact through data exchange between SEL operators. The full coupling pattern between submodels is given in a graphical specification language called gMML for Graphical Multiscale Modelling Language. Figure 5.3 shows the components of a gMML specification. The submodels are interconnected by components called conduits, each being connected to an out port from a submodel or mapper to an in port of a submodel or mapper. Ports are software components of a submodel into which we can read or write data. Figures 2.6a and 2.6b shows the gMML descriptions of the two tephra transport application versions.

Coupling templates

MMSF defines various types of coupling templates [27]. By coupling templates, we mean the structure of interaction and thus data exchange between submodels. Figure 2.7 shows an example of the coupling template between *transport* and *advection field* which is a call-release template. In our application, we identify the following coupling templates between submodels :

Source term* interaction with *Transport is an *interact* coupling, where O_i operator of the first submodel if connected to B of the second submodel. *Source term* exchange informations with *transport* at each iteration.

Plume* interaction with *Advection field is a *dispatch* coupling, where O_f operator of the first submodel if connected to F_{init} of the second submodel. *Plume* compute velocity profile and geometry of the plume, possibly at different points in time, and send the result to *advection field*.

Transport* interaction with *Advection field is a *call-release* coupling. *Transport* call *advection field* at each iteration, which computes the advection field.

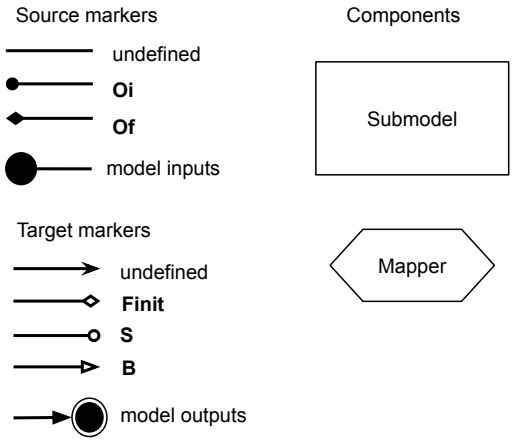


Figure 2.5: Components and markers considered in a gMML description. Mappers are component that allows for data transformation between submodels. There is no mapper used in the present application. See Chapter 5 for examples of the use of mappers. Circles representing model parameters and output accounts only for physical parameters, numerical parameters are not represented.

For the variant b of the application, we add the following coupling templates :

Plume* interaction with *Source term is a *dispatch* coupling. Plume computes the MER (Mass Eruption Rate) and send it to *source term*.

Atmosphere* interaction with *plume* and *advection field are *dispatch* coupling. *Atmosphere* retrieve atmospheric pressure, temperature and wind velocity from a reanalysis database and makes it available to *plume* and *advection field*.

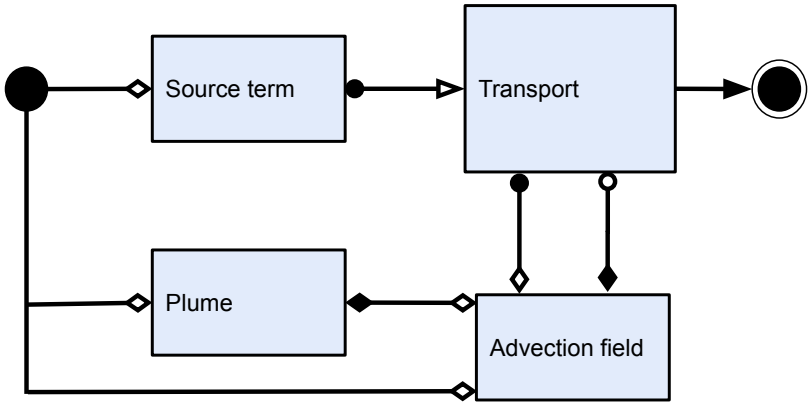
2.2.2 Details of the physical models

Now that we have defined the general structure of the application using MMSF formalisms, we give the detail of the physical models involved. We have made the choice to keep the original notation of each model to make it easier to refer to original publications.

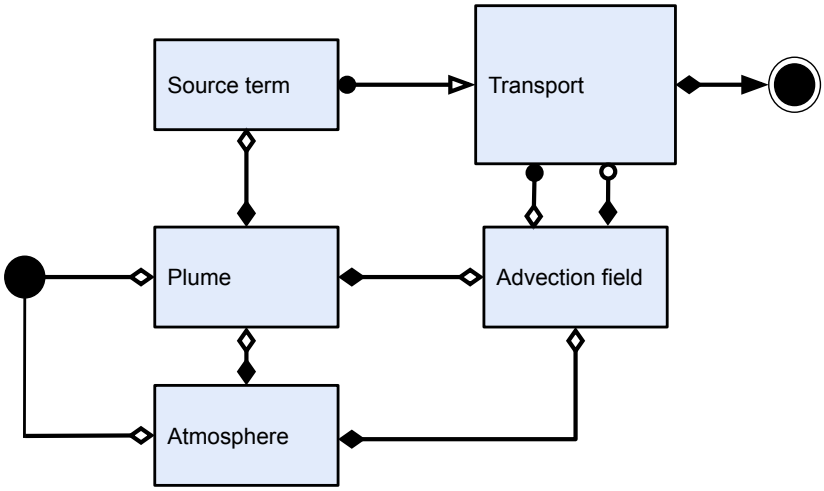
Plume velocity and geometry

Strong plume For the strong plume, we rely on the integral plume model of Woods [123], based on the buoyant plume theory. It is composed of a jet phase, where material is ejected from the vent at high speed. Then, speed decrease and material continue to rise due to buoyancy. When reaching the neutral buoyancy level (NBL), materials stop rising by buoyancy and starts spreading laterally as a gravity current.

The model describes the velocity profile and radius of the vertically rising volcanic plume as a



(a) gMML description of the tephra transport application which is used with an analytical description of the atmosphere and the Woods plume model. This corresponds to the design shown in Figure 2.4a.



(b) gMML description of the tephra transport application which is coupled with data from a reanalysis database and Degruyter plume model. This corresponds to the design shown in Figure 2.4b.

Figure 2.6: gMML description of the two variants of the tephra transport application. Once with atmosphere characteristics as a state equation integrated in *plume* and *advection field* (a) and with atmosphere characteristics as a separated submodel (b). In (a) *plume* is not connected to *source term*, because total erupted mass is given by model inputs, while in (b) it is connected because mass eruption rate of *plume* is considered. This also means that *source term* does not need to be connected to model parameters.

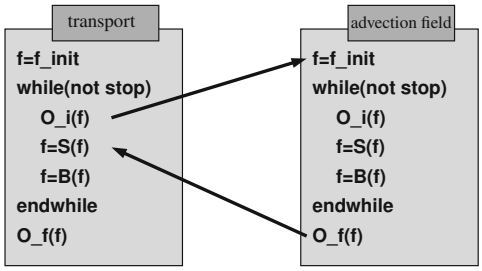


Figure 2.7: Call-release coupling between *transport* SEL and *advection field* SEL.

function of the height. It considers a top hat velocity profile (the vertical velocity is the same at a given height for any considered radius inside the plume).

The model is governed by the following set of differential equations :

$$\frac{d}{dz}\left(C_p\theta\right) = \frac{1}{\beta UL^2}\left[\left(C_aT + \frac{U^2}{2} - C_p\theta\right)\frac{d}{dz}\left(\beta UL^2\right) - \alpha UL^2 g\right] \tag{2.1}$$

$$\frac{1}{\beta} = (1-n)\frac{1}{\sigma} + \frac{nR_g\theta}{P} \tag{2.2}$$

$$n = 1 + (n_0 - 1)\frac{L_0^2 U_0 \beta_0}{L^2 U \beta} \tag{2.3}$$

$$R_g = R_a + (R_{g0} - R_a)\left(\frac{1-n}{n}\right)\left(\frac{n_0}{1-n_0}\right) \tag{2.4}$$

$$C_p = C_a + (C_{p0} - C_a)\left(\frac{1-n}{1-n_0}\right) \tag{2.5}$$

Jet phase model

$$\frac{dU}{dz} = -\frac{U}{8L}\sqrt{\frac{\alpha}{\beta}} + \frac{g(\alpha-\beta)}{\beta U} \tag{2.6}$$

$$\frac{d}{dz}(\beta UL^2) = \frac{g(\alpha-\beta)L^2}{U} - \beta L^2\frac{dU}{dz} \tag{2.7}$$

Convective phase model

$$\frac{dU}{dz} = \frac{1}{\beta UL^2}\left[g(\alpha-\beta)L^2 - U\frac{d}{dz}(\beta UL^2)\right] \tag{2.8}$$

$$\frac{d}{dz}(\beta U L^2) = 2kUL\alpha \tag{2.9}$$

Where jet phase model is applied if $\alpha(z) < \beta(z)$ (atmosphere density and plume density respectively, variables of the model are listed in Table 2.2) and convective phase model otherwise. This is a steady-state model. The set of differential equations is solved in the main C++ code (see Section 2.4) with a simple Euler scheme before the transport model is applied.

Symbol	Unit	Definition
U	$m \cdot s^{-1}$	plume vertical speed
L	m	plume radius
β	$kg \cdot m^{-3}$	plume density
C_p	$J \cdot kg^{-1} \cdot K^{-1}$	specific heat of the column material
θ	K	plume temperature
R_g	$J \cdot kg^{-1} \cdot K^{-1}$	specific constant of gas of the volcanic column
n		gas mass fraction
z	m	height
g	$m \cdot s^{-2}$	acceleration of gravity
σ	$kg \cdot m^{-3}$	pyroclasts density
α	$kg \cdot m^{-3}$	atmospheric density
C_a	$J \cdot kg^{-1} \cdot K^{-1}$	specific heat of air
P	Pa	atmospheric pressure
R_a	$J \cdot kg^{-1} \cdot K^{-1}$	specific gas constant of air
k		entrainment constant
T	K	atmosphere temperature

Table 2.1: Symbols of Woods plume model.

The main parameters of the model are U_0 the plume velocity at vent, L_0 the plume radius at the vent, n_0 the gas mass fraction of exolved volatiles at vent and T_0 the initial plume temperature. A specific set of those parameters will produce a plume with a given height. While it is often easier to have an estimation of the plume height rather than an estimation of those parameters, a simple Monte Carlo inversion strategy (described in Chapter 4) is implemented allowing retrieving a set of U_0, L_0, n_0, T_0 parameters compatible with a given value of plume height.

Weak plume When an eruption occurs in a windy atmospheric condition, the dynamic of volcanic plume can be modified by higher entrainment of ambient air, potentially developing a bent-over plume, or weak plume (Figure 2.1b). In that case, we rely on the model by Degruyter and Bonadonna, described in [49] and [48].

Again, this model is a steady state integral plume model, describing a profile of radius and velocity. This time, it does so along s, which is the curvilinear coordinate along the plume

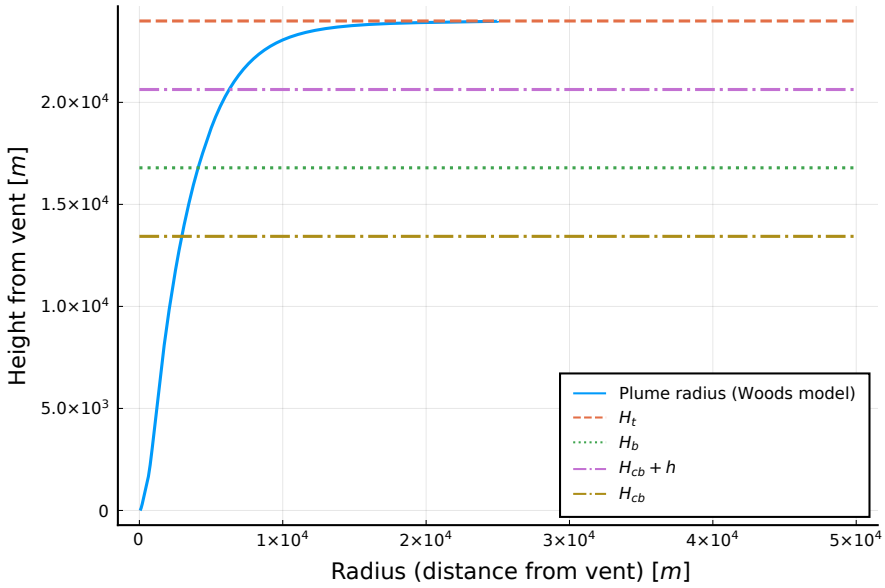


Figure 2.8: Geometry of a plume calculated with Woods model for $U_0 = 300m/s, L_0 = 50m, n_0 = 0.01, T_0 = 1200K$. Solid line is the height of the plume as a function of the radius. Horizontal lines represent the top of the plume H_t, the NBL H_b and the base of the umbrella cloud H_{cb}. The figure represents the radius of the plume as a function of the height, with the height on the y-axis.

trajectory. The governing equations of this model are

$$\frac{d}{ds}(\rho_d u r^2 \phi_d) = 2u_e r \rho_a \tag{2.10}$$

$$\frac{d}{ds}(\rho_v u r^2 \phi_v) = 0 \tag{2.11}$$

$$\frac{d}{ds}(\rho_s u r^2 \phi_s) = 0 \tag{2.12}$$

$$\frac{d}{ds}(\rho u^2 r^2) = g(\rho_a - \rho) r^2 \sin\varphi + v \cos\varphi \frac{d}{ds}(\rho u r^2) \tag{2.13}$$

$$\rho u^2 r^2 \frac{d\varphi}{ds} = g(\rho_a - \rho) r^2 \cos\varphi - v \sin\varphi \frac{d}{ds}(\rho u r^2) \tag{2.14}$$

$$\frac{d}{ds}\left(g \frac{\rho u r^2 (c\theta - c_a \theta_a)}{\rho_{a0} c_a \theta_{a0}}\right) = -\frac{\rho}{\rho_{a0}} u r^2 N^2 \sin\varphi \tag{2.15}$$

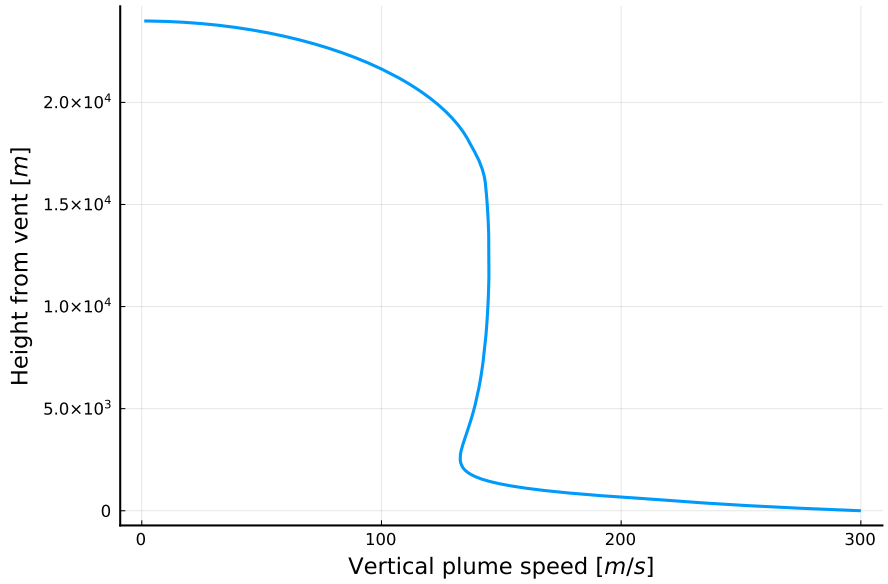


Figure 2.9: Plume vertical velocity as a function of the height calculated with Woods model for $U_0 = 300 m/s, L_0 = 50 m, n_0 = 0.01, T_0 = 1200 K$. H_t is defined by the height at which the velocity drops to zero. The figure represents averaged velocity of the plume material as a function of the height, with the height on the y-axis.

where N is the atmospheric buoyancy frequency defined as

$$N^2 = \frac{g^2}{c_a \theta_{a0}} \left(1 + \frac{c_a}{g} \frac{d\theta_a}{dz}\right) \tag{2.16}$$

and

$$\phi_d + \phi_v + \phi_s = 1 \tag{2.17}$$

$$(\rho_d \phi_d R_d + \rho_v \phi_v R_v)\theta = P \tag{2.18}$$

$$\rho = \rho_d \phi_d + \rho_v \phi_v + \rho_s \phi_s \tag{2.19}$$

$$c = \frac{c_d \rho_d \phi_d + c_v \rho_v \phi_v + c_s \rho_s \phi_s}{\rho} \tag{2.20}$$

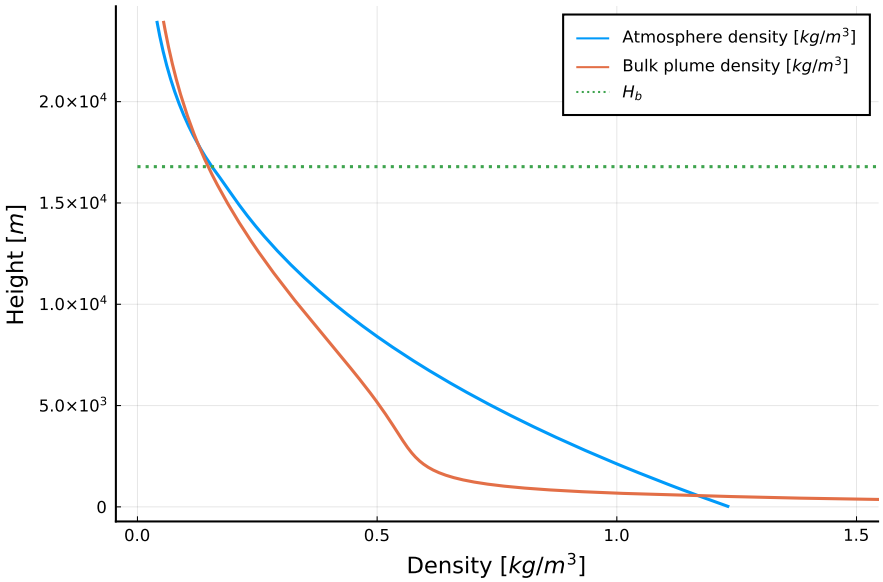


Figure 2.10: Atmosphere and plume bulk density calculated with Woods model for $U_0 = 300m/s, L_0 = 50m, n_0 = 0.01, T_0 = 1200K$. H_b is the NBL where plume bulk density becomes equal to the atmosphere density at high altitude and is here estimated as $H_b \approx 0.7H_t$ [25]. For the air density computation, values at sea level are $T_0 = 288K, P_0 = 101325Pa$. The figure represents bulk density of the plume and air density as a function of the height, with the height on the y-axis.

The entrainment velocity u_e is defined as

$$u_e = \alpha|u - v\cos\varphi| + \beta|v\sin\varphi| \tag{2.21}$$

and the pressure along the plume

$$\frac{dP}{ds} = -\rho_a g\sin\varphi \tag{2.22}$$

and the position of the centre of the plume

$$\frac{dx}{ds} = \cos\varphi \tag{2.23}$$

$$\frac{dz}{ds} = \sin\varphi \tag{2.24}$$

Again, this is a steady-state model that has to be solved before the transport model. It is solved by a separate Matlab package provided by the authors of the model, and its output read by the

main C++ code (see Section 2.4).

Symbol	Unit	Definition
s	m	plume centreline coordinate
x	m	horizontal coordinate
z	m	height
φ	rad	angle between x and plume centreline
ρ_d	$kg \cdot m^{-3}$	density of dry air in plume
ρ_v	$kg \cdot m^{-3}$	density of water vapour in plume
ρ_s	$kg \cdot m^{-3}$	density of solids
ρ	$kg \cdot m^{-3}$	bulk plume density
ρ_a	$kg \cdot m^{-3}$	atmosphere density
ϕ_d		volume fraction of dry air in plume
ϕ_v		volume fraction of water vapour in plume
ϕ_s		volume fraction of solids in plume
u	$m \cdot s^{-1}$	plume centreline velocity
v	$m \cdot s^{-1}$	horizontal wind velocity (considered constant)
r	m	plume radius
g	$m \cdot s^{-2}$	acceleration of gravity
c_d	$J \cdot kg^{-1} \cdot K^{-1}$	specific heat capacity at constant pressure of dry air
c_v	$J \cdot kg^{-1} \cdot K^{-1}$	specific heat capacity at constant pressure of water vapour
c_s	$J \cdot kg^{-1} \cdot K^{-1}$	specific heat capacity at constant pressure of solids
c	$J \cdot kg^{-1} \cdot K^{-1}$	specific heat capacity at constant pressure of the plume
c_a	$J \cdot kg^{-1} \cdot K^{-1}$	specific heat capacity at constant pressure of the atmosphere
R_d	$J \cdot kg^{-1} \cdot K^{-1}$	specific gas constant of dry air
R_v	$J \cdot kg^{-1} \cdot K^{-1}$	specific gas constant of water vapour
θ	K	plume temperature
θ_a	K	atmosphere temperature
α		radial entrainment coefficient
β		wind entrainment coefficient

Table 2.2: Symbols of Degruyter and Bonadonna plume model.

Gravity current

When plume material reaches the NBL, it continues to rise up to H_t due to momentum, and then starts spreading laterally as a gravity current. For the description of the velocity of this radial current, we rely on the umbrella cloud model of Rossi from [100], which is

$$U_{gc} = \left(\frac{\lambda N V_{wind}}{2r} \cos(\theta) R^2 + \frac{U_g \lambda N R^2}{4} \frac{1 - e^{-\frac{r^2}{b^2}}}{r} \right)^{\frac{1}{2}} \tag{2.25}$$

where $\lambda \approx 1$ is a correction factor, N the atmospheric buoyancy frequency, V_{wind} wind speed

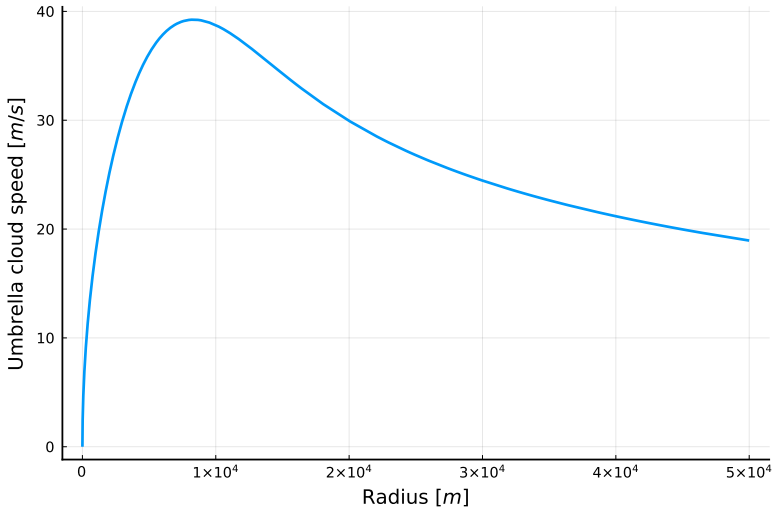


Figure 2.11: Example of umbrella cloud velocity as a function of the radius with plume characteristics computed from Woods model with parameters $U_0 = 300m/s, L_0 = 50m, n_0 = 0.01, T_0 = 1200K$.

at umbrella cloud height, θ the angle of the plume centreline with horizontal at the NBL, r the distance from the centre of the umbrella cloud, R the radius of the plume at NBL, U_g the speed of the plume at NBL and b the characteristic width of the Gaussian profile of a plume, defined as [45]

$$\frac{R^2}{b^2} = 2 \tag{2.26}$$

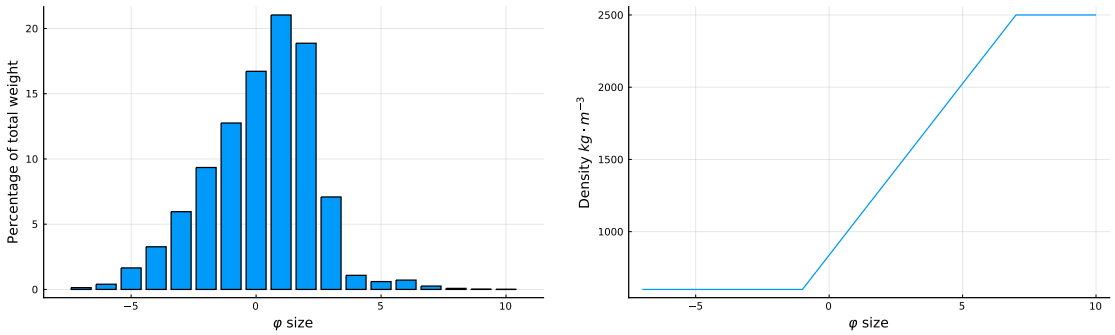
For a strong plume, H_t is the top of the plume, $H_b \approx 0.7H_t$ is the NBL, $H_{cb} \approx 0.8H_b$ is the height at which particles starts to sediment from umbrella cloud base [25]. It has been observed that $h \approx 0.3H_t$, where h is the thickness of the umbrella cloud [31]. Umbrella cloud is considered ranging from H_{cb} to $H_{cb} + 0.3H_t$.

For the weak plume, we consider the same proportions, and that H_b is the height at which the angle θ is 0. This model is solved at each iteration of the transport model.

Particles

Modelled particles are tephra particles, which designates solid material of all sizes ejected during a volcanic eruption. Tephra is compounded of clasts (fragments of rocks), whose sizes are noted using the Krumbein ϕ scale [76]

$$\phi = -\log_2 \frac{D}{D_0} \tag{2.27}$$



(a) (b)

Figure 2.12: TGSD estimated with Voronoi tesselation for Pululagua 2450 BP eruption [117], with one class per ϕ size (a) and density (b) of volcanic particles.

where ϕ is the grain size in phi units, D is the diameter of the rock in mm and D_0 is a reference diameter of $1mm$. Particles are classified in groups of the same density and size called classes or families of particles.

Total grain size distribution and density The total grain size distribution (TGSD) denotes the distribution in ϕ size of particles and their corresponding density. Usually, TGSD is expressed in percent of the total mass, and can be estimated from field measurement using various techniques, including Voronoi tesselation [19, 111].

Sedimentation velocity When a particle is released in the atmosphere, it accelerates due to gravity, and reaches a constant speed due to air resistance. This velocity is called the terminal velocity of the particle. While the distance to reach this terminal velocity is relatively small for considered particles ($130m$ for a $1cm$ spherical particle of density $2700\frac{kg}{m^3}$ falling at sea level [4]) compared to the height of volcanic plumes (several kilometres), we consider that particles reach instantaneously their terminal velocity.

Sedimentation speed of particles can be approximated from fluid theory as

$$v_t = \sqrt{\frac{4gd_{eq}(\rho_p - \rho_f)}{3\rho_f C_D}} \tag{2.28}$$

where g is the acceleration of gravity, d_{eq} is the diameter of the sphere surrounding the particle (the particle diameter for a sphere), ρ_p the density of the particle, ρ_f the density of the surrounding fluid and C_D the drag coefficient. C_D is the most difficult parameter to determine,

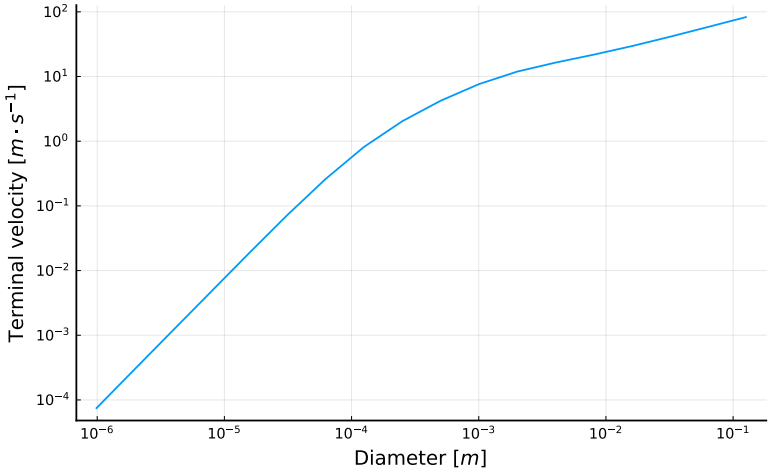


Figure 2.13: Terminal velocity of particles ranging from a size of -7φ to 10φ in an atmosphere at sea level ($\rho_a = 1.225 \frac{kg}{m^3}$, $\mu = 1.789 \times 10^{-5} \frac{Ns}{m^2}$) and particles of density $2500 \frac{kg}{m^3}$.

as it depends on the particle Reynolds number, which is defined as

$$R_e = \frac{\rho_f |v_r| d_{eq}}{\mu} \tag{2.29}$$

where v_r is the relative velocity of the particle to the surrounding fluid.

In this work, we rely on the formulation of a drag coefficient of Bagheri [3], which gives an expression valid for any regime $R_e < 3 \times 10^5$ and account for non-spherical particles. Particles are considered spherical in the present work. The drag coefficient is expressed as

$$C_D = \frac{24 k_S}{R_e} \left(1 + 0.125 (R_e kN/kS)^{2/3}\right) + \frac{0.46 k_N}{1 + \frac{5330}{R_e k_N / k_S}} \quad R_e < 3 \times 10^5 \tag{2.30}$$

where

$$k_S = \frac{1}{2} \left(F_S^{1/3} + F_S^{-1/3}\right) \tag{2.31}$$

$$k_N = 10^{0.45(-\log F_N)^{0.99}} \tag{2.32}$$

$$F_S = f e^{1.3} \left(\frac{d_{eq}^3}{LIS}\right) \tag{2.33}$$

$$F_N = f^2 e\left(\frac{d_{eq}^3}{LIS}\right) \tag{2.34}$$

L, I and S are the dimensions of the particle and are respectively L : longest, I : intermediate and S : shortest dimension of the particle. f and e are form factors, f is flatness and e is elongation, defined as

$$f = \frac{S}{I} \tag{2.35}$$

$$e = \frac{I}{L} \tag{2.36}$$

For spherical particles,

$$F_S = F_N = 1 \tag{2.37}$$

As

$$R_e = f(v_t) \quad \text{and} \quad v_t = f(R_e) \tag{2.38}$$

an iterative scheme is used to compute v_t, where v_t is first computed with $R_e = 1$, and v_t computed again from this new value, and the process is repeated until R_e remains almost the same between two iterations. The convergence of this procedure is usually fast (fewer than four iterations).

Atmospheric conditions

Previously described models need atmospheric conditions, such as atmospheric temperature and pressure. For this, it is either possible to rely on analytical approximation or on reanalysis database. It would as well be possible to use data from weather forecasting models, but this possibility has not been investigated in this work.

Weather reanalysis databases Weather reanalysis is the task of interpolating atmospheric condition data from sensors to give values of atmospheric conditions all over the globe at any time. For example, the ECMWF-ERA-Interim [47] provides data for any point on the globe from 1979 to 2019 with a time resolution of 12 hours and a spatial resolution of approximately $80km$.

It's replacement the ECMWF-ERA5 [15] database provides data from 1979 to present with a $30km$ grid and vertical resolution of the atmosphere over 137 levels from the ground to $80km$ on an hourly basis [69]. Data from 1950 to 1978 are available in preliminary version (as of May 2021). Another widely used database is the NOAA NCEP/NCAR reanalysis database which provides data every six hours on a 2.5° grid from 1948 to present [75, 88].

Analytical approximations Weather sensing only started last century, and for eruptions that occurred in the distant past, it is necessary to have models in order to approximate atmosphere characteristics. Regarding wind speed, we rely on the approximation proposed by [72], with a linear increase of wind velocity from sea level to the tropopause where the wind is maximum, and then a linear decrease of the speed of the wind up to the stratosphere where the wind is set to 10% of the maximum wind speed :

$$u_w(z) = \begin{cases} \dfrac{z}{H_{tp}} \cdot u_{wmax} & \text{for } z \leq H_{tp} \\ \left[0.9\left(1 - \left(\dfrac{z - H_{tp}}{H_{ss} - H_{tp}}\right)\right) + 0.1\right] \cdot u_{wmax} & \text{for } H_{tp} < z \leq H_{ss} \\ 0.1 \cdot u_{wmax} & \text{for } H_{tp} > H_{ss} \end{cases} \tag{2.39}$$

where H_{tp} is the height of the tropopause, H_{ss} is the height of the stratosphere and u_{wmax} is the maximum wind speed (at the tropopause). Figure 2.14 shows an example of the wind speed profile with this model. For atmospheric pressure and temperature, we use the same approximation as [123]. With the temperature

$$T(z) = \begin{cases} T_0 - \omega_t z & \text{for } z \leq H_{tp} \\ T_0 - \omega_t H_{tp} & \text{for } H_{tp} < z \leq H_{ss} \\ T_0 - \omega_t H_{tp} + \omega_s (z - H_{ss}) & \text{for } z > H_{ss} \end{cases} \tag{2.40}$$

and the pressure

$$P(z) = \begin{cases} P_0 \frac{T(z)}{T_0}^{\frac{g}{R_a \omega_t}} & \text{for } z \leq H_{tp} \\ P_0 \frac{T(z)}{T_0}^{\frac{g}{R_a \omega_t}} \exp\left(\frac{-g(z - H_{tp})}{R_a (T_0 - \omega_t H_{tp})}\right) & \text{for } H_{tp} < z \leq H_{ss} \\ P_0 \frac{T_0 - \omega_t H_{tp}}{T_0}^{\frac{g}{R_a \omega_t}} \exp\left(\frac{-g(H_{ss} - H_{tp})}{R_a (T_0 - \omega_t H_{tp})}\right) \frac{T(z)}{T_0 - \omega_t H_{tp}}^{\frac{-g}{R_a \omega_s}} & \text{for } z > H_{ss} \end{cases} \tag{2.41}$$

with $\omega_t = 6.5e^{-3} K/m$ the temperature gradient in the troposphere and $\omega_s = 2e^{-3} K/m$ the temperature gradient in the stratosphere. Finally, air density is computed with

$$\rho_a(z) = \frac{P(z)}{R_a T(z)} \tag{2.42}$$

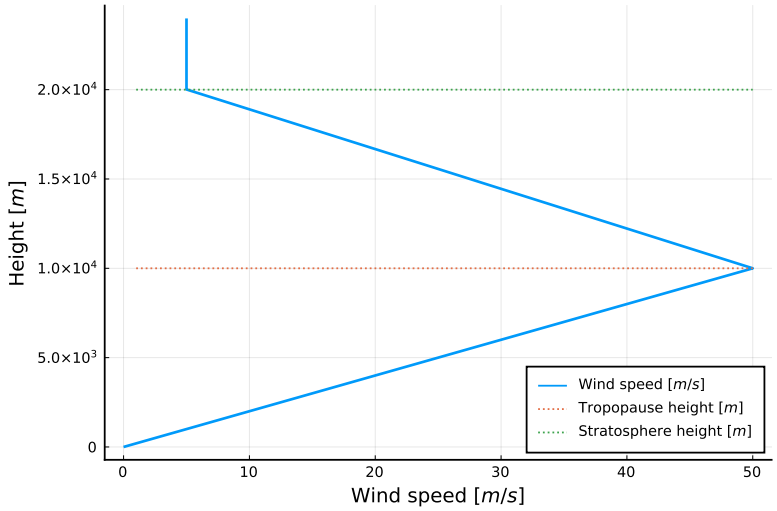


Figure 2.14: Wind velocity approximated with a maximum wind speed of $50m/s, H_{tp} = 10000m, H_{ss} = 20000m$. H_{tp} is the height of the tropopause, H_{ss} is the height of the stratosphere.

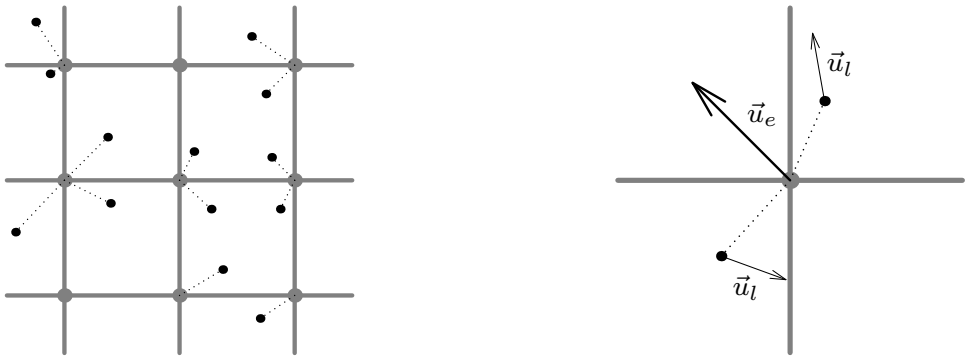
Figure 2.10 shows an example of air density with those models.

2.3 The Lagrangian on grid transport model

The particle transport submodel (named *transport* in the SSM and gMML descriptions, Figures 2.4 and 2.6) is the most numerically intensive submodel of the application. Several techniques like lattice Boltzmann, probabilistic multi-particle cellular automata or Lagrangian methods, can be used to model point particle displacements in an advection field [104, 36, 84, 113]. Cellular automata are discrete Eulerian models where the simulated space is subdivided along a Cartesian grid. Each grid site stores the number of particles residing on this site for every particle class, together with the local advection field. A probabilistic scheme is then applied to transport particles from one site to another at each iteration. This kind of model has the advantage of facilitating parallelization. However, the probabilistic nature of particle motion creates an anisotropic numerical diffusion ([112]).

The lattice Boltzmann method is another technique for modelling an advection-diffusion process. However, its numerical stability and accuracy depend on simulation parameters ([108]). Cellular automata and lattice Boltzmann methods are compared in [112].

From the pure Lagrangian perspective, the characteristics of each particle are considered individually. This kind of model doesn't generate any numerical diffusion while particles are not constrained to stay at grid positions on the domain sites (Figure 2.15). This allows a precise treatment of diffusion, regardless of the spatial integration step. However, the drawback is that parallelization is more tedious when particles can interact.



(a) Representation of the Lagrangian on grid model without the velocities.

(b) Differentiation between Eulerian and Lagrangian velocities.

Figure 2.15: Representation of the Lagrangian on grid model. Gray lines represent the grid, gray points are the centre of the sites, black points are the particles and dotted lines represent association between particles and their respective sites.

For these reasons, we chose a hybrid representation defined as a Lagrangian on grid model. In this case, particles are moved according to a Lagrangian model, but remain attached to sites in space (this representation is comparable to the one used in the cell linked-list algorithm [93]). This method combines the simplicity of pure Lagrangian and cellular automata models. As for a pure Lagrangian method, the Lagrangian on grid method doesn't generate any numerical diffusion. Moreover, this method remains unconditionally stable like cellular automata and Lagrangian methods but unlike lattice Boltzmann methods.

The Lagrangian on grid model represents space as a Cartesian grid. The grid nodes are the domain sites, each site storing the associated particles and applying the advection field to them. But in contrast to a cellular automata model, each particle stores a velocity as well as a location information (the displacement with respect to the site it is linked to).

Thus, the position of a particle at time t can be written $\vec{r}(t) = \vec{r}_s(t) + \vec{r}_p(t)$ with $\vec{r}_s$ the site position to which the particle is linked and $\vec{r}_p$ the particle displacement relative to this site. Similarly, the velocity applied to a particle can be written $\vec{u}(t) = \vec{u}_e(t) + \vec{u}_l(t)$ with $\vec{u}_e$ the Eulerian velocity (imposed by the site) and $\vec{u}_l$ the Lagrangian velocity (specific to the particle).

Then, when a particle is moved, we compute its new position after a time step Δt via $\vec{r}(t+\Delta t) = \vec{r}(t) + \Delta t\, \vec{u}(t)$. If the particle displacement is such that $\|\vec{r}(t+\Delta t) - \vec{r}_s(t)\| \leq \frac{\Delta x}{2}$ with Δx the distance between two sites, then $\vec{r}_s(t+\Delta t) = \vec{r}_s(t)$ (the particle remains linked to the same site). Otherwise the particle is linked to the closest site (Figure 2.15).

In the Lagrangian on grid model, no numerical diffusion is generated. In order to add some diffusion, we apply random velocities $\vec{u}_r$ to each particle ($\vec{u}_{ra}$ and $\vec{u}_{rp}$) in order to generate a given diffusion coefficient (see Appendix A for details).

2.4 Tephra transport simulator

The implementation of the model decribed in this chapter is concretely separated in three main parts :

- A Python script that gathers data from ECMWF-ERA-INTERIM database. It embeds the *amtosphere* submodel. Data gathering from NOAA NCEP/NCAR database has not been automated.
- A Matlab application that solves the Degruyter plume model, provided by the authors of the model. It embeds the *plume* submodel when working with Degruyter plume model.
- A C++ application named Tetras that solves the other described models. If necessary, it reads outputs from the aforementioned softwares. It embeds the *plume* submodel when working with Woods plume model, as well as the *source term*, *transport* and *advection field* submodels.

Tetras is the software on which the main development effort has been put among those components. It is structured as a generic C++ library to simulate particles in an advection field by applying the Lagrangian on grid method parallelized on a distributed memory architecture (PIAF, Particles In Advection Field). Tetras was then built on top of this library.

PIAF and Tetras have a modular structure that allows reusing software components and data structures for custom particle advection applications. For example, velocities are defined as function objects, which are data structures that can be called as functions. We have been able to directly reuse the definition of the velocities to implement volcano-lagrangian-seer, which is a pure Lagrangian TTDM described in Chapter 3. Implementation details of PIAF and Tetras are given in Chapter 5, Section 5.2.

2.4.1 Adaptive time step

In our application, we can decrease the number of iterations (and thus the computation time) by increasing Δt. Nevertheless, a movement for a particle is valid if the particle stays linked to the same site or moves to a direct neighbour. Consequently, the choice of Δt is limited by the choice of the grid spacing Δx and by the velocities $\vec{u}$.

We wish to choose the largest possible Δt for a given Δx. Since this value may vary over time, we propose to dynamically adapt Δt using the following simple heuristic :

- Every τ iterations, Δt is increased by a small quantity δt.
- If a particle makes an invalid movement, Δt is decreased by δt and the iteration cancelled and performed again.

In order to cancel an iteration, we keep the state of the system unmodified using temporary

buffers until the iteration is complete. The behaviour of the simulator with adaptive Δt is summarized in algorithms 2.2 and 2.3.

Algorithm 2.2: Main algorithm of the simulation, we suppose that data are shared between the main algorithm and the function `tryMove()`

```
1 while particles in domain do
2 |   if iteration mod τ == 0 then
3 |   |   Δt += δt;
4 |   end
5 |   while not tryMove() do
6 |   |   Δt -= δt;
7 |   end
8 |   deposit particles of terrain buffer on the terrain; put all particles of repository buffer
  |    into repository; swap domain and domain buffer;
9 end
```

2.4.2 Parallelization

For the implementation to scale well with the amount of data, our library has been parallelized using the MPI (Message Passing Interface) library. In our case, we chose to split the domain into blocks of sites, which are distributed across multiple cores[1]. We implemented two solutions : a 1D domain splitting and a 3D block cyclic domain splitting (Figure 2.16). The first consists in slicing the domain along a given axis, with each core taking care of one domain slice. The latter distribution allows to deal better with an inhomogeneous distribution of particles in the domain which would otherwise lead to load imbalance between the cores and thus performance loss. Indeed, since the domain is split across multiple cores and there is a unique injection point (in the case of the volcano model), at the beginning of a simulation, particles tend to remain linked to sites managed by a single core.

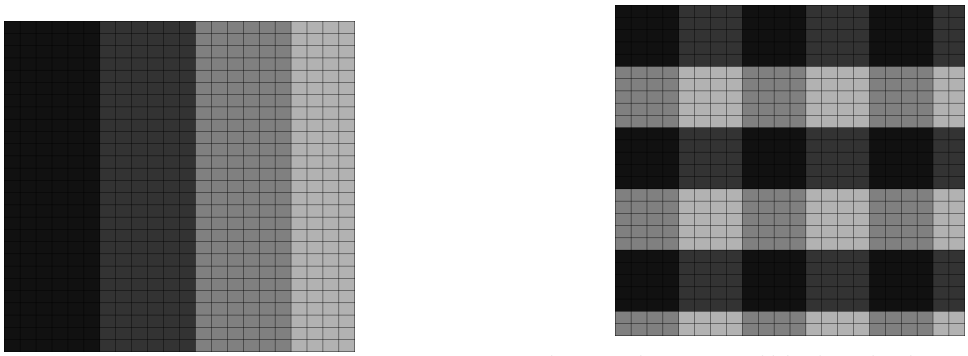
We want to assign each core a certain number of small domain blocks, distributed over the entire domain. We thus hope to obtain a better load balance as each core will have to manage blocks located at different places in the domain. The technique described here is comparable to the 2D block cyclic matrix distribution described in [92].

We first arrange the cores along a virtual 3D grid and thus assign each core an index (p_x, p_y, p_z) on the grid. Given the number of cores np, we factor np writing $np_x \cdot np_y \cdot np_z = np$ (it may occur that only a 1D or 2D structure is constructed). To a core of rank p ($0 \leq p \leq np-1$), we assign a grid index (p_x, p_y, p_z) with $p_x = p \mod np_x$, $p_y = \frac{p \mod (np_x \cdot np_y)}{np_x}$ and $p_z = \frac{p}{np_x \cdot np_y}$.

[1] We recall that we use the term "core" to denote a sequential computing unit.

Algorithm 2.3: Algorithm of the function `tryMove()`

```
1  forall site in domain do
2      compute velocity at the site; if site is in contact with ground then
3          forall particle in site do
4              put particle in terrain buffer;
5          end
6      else
7          forall particle in site do
8              compute velocity of the particle; compute new position of the particle; if new
                position is valid then
9                  if particle is outside the domain then
10                     put the particle in repository buffer;
11                 else
12                     put the particle in domain buffer;
13                 end
14             else
15                 return false;
16             end
17         end
18     end
19 end
20 return true;
```



(a) Unidimensional domain splitting

(b) Two-dimensional block cyclic domain splitting

Figure 2.16: Illustration of (a) unidimensional domain splitting and (b) two-dimensional block cyclic domain splitting with four cores, each colour representing a different core. The implemented three-dimensional block cyclic domain splitting is an extension to three dimensions of the two-dimensional block cyclic distribution presented here. Figures are inspired from [92].

We now discuss the 3D domain block splitting and the assignment of blocks to the cores. Let s_x, s_y, s_z and h_x, h_y, h_z be the dimensions respectively of the domain and of a block. The number of blocks nb_x, nb_y, nb_z in each dimension is given by

$$(nb_x, nb_y, nb_z) = \left(\lceil \frac{s_x}{h_x} \rceil, \lceil \frac{s_y}{h_y} \rceil, \lceil \frac{s_z}{h_z} \rceil \right) \tag{2.43}$$

Note that blocks on the boundary of the domain may be smaller than $h_x \cdot h_y \cdot h_z$.

We next decide how to allocate the blocks to the cores. The domain blocks are assigned cyclically to the cores along each dimension. More precisely, a site with global index (r_x, r_y, r_z) will belong to a block with index $b = \left(\frac{r_x}{h_x}, \frac{r_y}{h_y}, \frac{r_z}{h_z} \right)$. It also has a local index inside the block as in the case of the 1D splitting. This block will be managed by the core with index

$$(p_x, p_y, p_z) = (b_x \mod np_x, b_y \mod np_y, b_z \mod np_z) \tag{2.44}$$

A core with index (p_x, p_y, p_z) takes care along the x-axis of $\lfloor nb_x / np_x \rfloor$ blocks if $p_x < nb_x \mod np_x$, and $\lceil nb_x / np_x \rceil$ blocks otherwise. Similar expressions hold along the y- and z-axes.

2.5 Validation of the physical model with field observations

We ran our simulations on three systems : the *Baobab* and *Yggdrasil* clusters of the Geneva's higher education institutions and the *Piz Daint* supercomputer of the Swiss National Supercomputing Center.

We compared the model output with field data for two eruptions characterized by very different plume dynamics and interaction with the surrounding wind field. The first is the 2450 BP eruption of Pululagua volcano (Ecuador)[117], which produced an approximately $25km$ high strong plume above a vent located approximately at and altitude of $2500m$ that rose in the absence of wind (plume heights are here all indicated as above the vent). The second is the 17 June 1996 eruption of Ruapehu volcano (New Zealand)[20], which produced an approximately $7km$ high weak plume above a vent located at approximately $3000m$ in a wind field with an average speed of $24m \cdot s^{-}1$ at the tropopause. The associated field observations used for comparing computed and observed mass per unit area (MPA) include:

- the geographical position (UTM WGS84 for Pululagua or NZGD1949 for Ruapehu);
- the total deposited mass per unit area MPA $kg \cdot m^{-2}$;
- the proportion of MPA for each particle family.

Some crucial input parameters are reported in the literature or can be derived from external models (e.g., TGSD, erupted mass, wind profile and atmospheric conditions), while others (mostly plume and diffusion parameters) are more difficult to define. In particular, the TGSD is reported by [19] for Ruapehu (i.e., Md_ϕ and sorting of -0.8 ϕ and 2.4, respectively) and by [117] for Pululagua (i.e., Md_ϕ and sorting of 0.2 ϕ and 1.9, respectively), while [117] suggests a total erupted mass of $4.5 \times 10^{11} kg$ for Pululagua. Given that the Ruapehu eruption was observed and consisted of two sustained events of 4h30 and 2h respectively, we derived the total mass (i.e., $10^{10} kg$) directly from the mass eruption rate (given by the plume model) combined with the total eruption time. This result is in agreement with the total erupted mass of $5 \times 10^{9} kg$ derived by [20] based on field data. We used the NOAA NCEP/NCAR reanalysis database for the wind and atmospheric data, which provides four daily profiles averaged on a $2.5° \times 2.5°$ grid [75, 88]. For this particular test case, we used wind data at the vent location, and applied it to the whole domain. It is worth noting that, as shown by [20], the spreading of the Ruapehu cloud was strongly controlled by wind advection and, therefore, gravitational spreading was neglected. In contrast, the spreading of the Pululagua cloud was controlled by buoyancy given that the eruption occurred in absence or weak atmospheric wind.

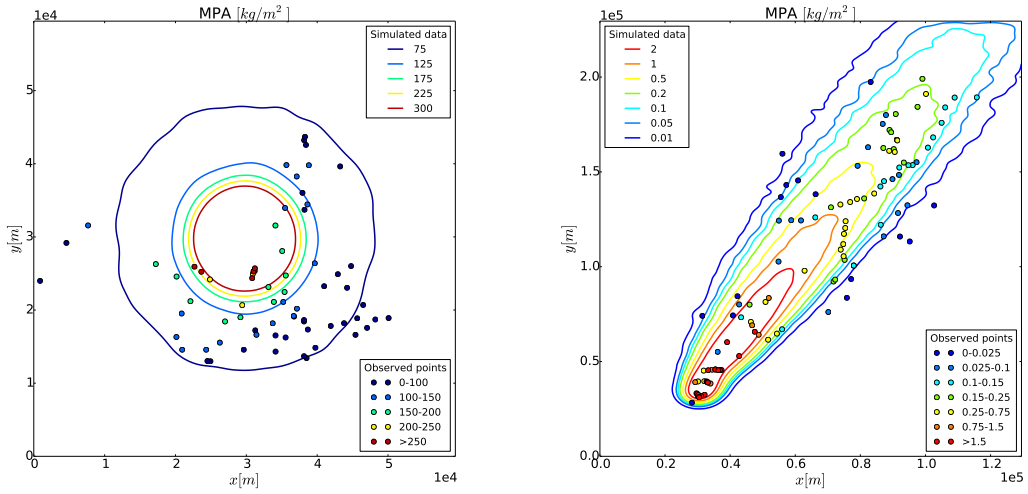
Simulations have been performed based on Woods plume model (for Pululagua) and Degruyter plume model (for Pululagua and Ruapehu). For simulations based on Degruyter plume model, the strategy adopted was to derive the plume and diffusion parameters by stochastically sampling appropriate variation ranges and identify the values associated with the best fit of observed MPA. Parameter exploration for each eruption was made with : $U_0 \in [20,300] m \cdot s^{-1}$, $r_0 \in [10,100] m$, $D_a \in [100,10000] m^2 \cdot s^{-1}$, $D_p \in [1000,30000] m^2 \cdot s^{-1}$. For Ruapehu, $T_0 = 1273K$ comes from [48] and n_0 has been set to 0.03. For Pululagua, T_0 was sampled in $[1000,1200]K$ and n_0 set to 0.01. Table 2.3 shows the best-fit parameters for the simulations. Plume heights above vent based on these best-fit parameters are $25km$ and $7km$ for Pululagua and Ruapehu, respectively, which are in good agreement with observations [117, 19]. Cloud dispersal was initialized considering plume height above vent (i.e., $\sim 3km$ for both Pululagua and Ruapehu

Symbol	Unit	2450 BP Pululagua	1996 Ruapehu
U_0	$m \cdot s^{-1}$	300	40
r_0	m	$50m$	50
T_0	K	1200	1273
D_a	$m^2 \cdot s^{-1}$	300	1000
D_p	$m^2 \cdot s^{-1}$	3000	4500

Table 2.3: Results of parameter exploration. These values are used for the presented simulation outputs.

volcanoes).

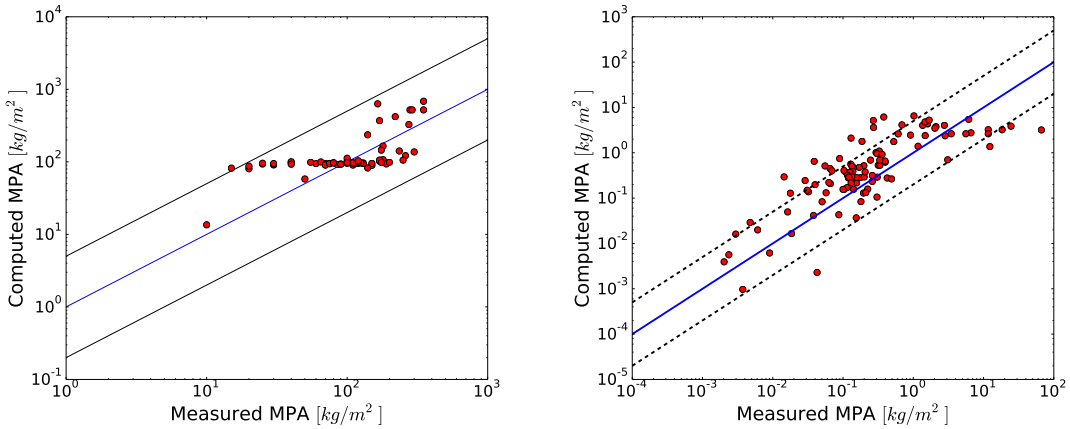
A quantitative comparison between results of simulations based on Degruyter plume model with observed MPA for both eruptions is shown in Figure 2.17 and Figure 2.18. In particular, the computed isomass map for the 1996 Ruapehu eruption well describes the strong wind advection (Figure 2.17). The associated plume being strongly bent-over by the wind, most lapilli and coarse ash particles fell in proximal area ([20]). Very proximal sedimentation couldn't be reproduced because of the lack of topography in our model. Topography would indeed strongly affect particle deposition up to $20km$ from vent.



(a) (b)

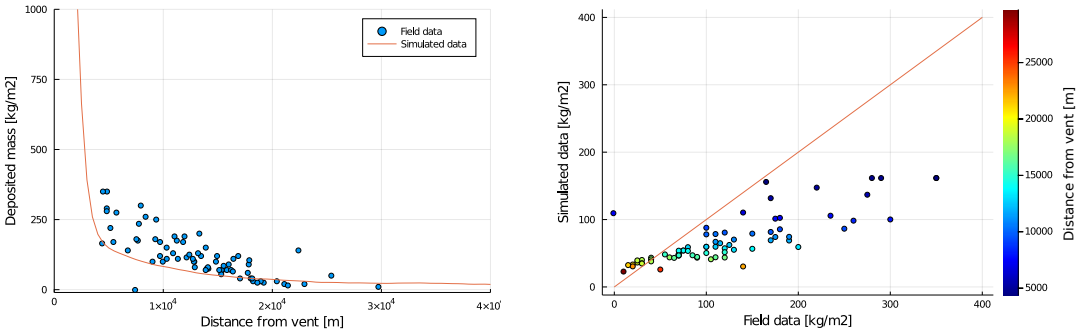
Figure 2.17: Simulated contour map of MPA (coloured lines) and observed MPA (coloured circles) for the 2450 BP Pululagua deposit (a) and the 1996 Ruapehu deposit (b).

For simulations based on Woods plume model, we can also determine plume parameters through a Monte Carlo inversion strategy applied to the plume model which allows obtaining eruption parameters based on a desired plume height (here $\approx 25km$). This procedure is described in Chapter 4, Section 4.2.



(a) (b)

Figure 2.18: Comparison of computed and observed MPA values for the Pululagua 2450 BP (a) and 1996 Ruapehu (b) eruptions. In the case of a perfect match between computed and observed values, dots would be located on the blue line. Dashed lines represent over or under estimations of 5 and 1/5 times the observed values.



(a) (b)

Figure 2.19: (a) Deposited mass plotted as a function of distance from vent for Pululagua eruption simulation with Woods plume model. (b) Comparison of measured MPA with computed MPA for this same eruption simulation. Colour of dots is a function of distance from vent (in m). Red dots are distal points and blue dot proximal points.

Figure 2.19 shows results of a simulation based on the Woods plume model with $U_0 = 318m \cdot s^{-1}, r_0 = 52m, n_0 = 0.01, T_0 = 1256K$. $D_p = 1000m^2 \cdot s^{-1}$ and $D_a = 300m^2 \cdot s^{-1}$ have been set empirically. As the deposit is considered axisymmetric, we plot the simulated deposit and measured deposit as a function of distance from vent. We see that a drawback of the present model is its inability to reproduce proximal deposits, where simulated deposited mass next to the vent is very high, and then becomes underestimated several kilometres away from the vent.

A dedicated sensitivity analysis, following [103], was performed to optimize the number of particles used in the simulations. In fact, the total number of particles corresponding to the erupted mass and the particle characteristics given in this section would be of the order of magnitude of 10^{22} to 10^{23} particles for Ruapehu and Pululagua simulations. Current computers cannot deal with such a number of particles injected in the simulator.

We therefore resorted to a statistical sampling approach. We injected a much smaller number of particles, and then scaled the output to get the correct number. Running our simulations with more than 10^6 particles proved to be statistically significant. We used a total of $18 \cdot 10^6$ particles for 2450 BP Pululagua simulations and $23 \cdot 10^6$ particles in 1996 Ruapehu simulations. For a complete description of the procedure, see Appendix A.

The input TGSD of Pululagua and Ruapehu consider respectively 18 and 23 classes of particles. In some cases, the number of classes of the TGSD is too low and particles of the same class tends to deposit in groups on the ground. For such cases, we rely on an interpolation strategy that allows us to increase the number of classes while keeping the same TGSD profile. For example, this allows using 180 classes of particles instead of 18 for Pululagua.

Note that prior to running our volcano simulations, we validated the advection-diffusion process (also in Appendix A).

2.6 Performance models

The main goal of a performance model is to characterize the execution time of a simulation in terms of its parameters. This execution time will, of course, depend on the physical model implemented. A parallel performance model will provide insight into the efficiency of the parallel implementation with respect to the computational resources involved. We have carried out a performance study with both simulation cases described before. Performance measurements for Pululagua were made with square domains with the vent located in the middle, and for Ruapehu with an elongated domain in the north direction with the vent located near the south-west corner of the domain, while the wind was blowing towards north-east. The performance model focuses on the submodels *transport* and *advection field*, which are implemented as a monolithic code within Tetras and are the most computationally intensive submodels of the overall application.

2.6.1 Choice of the adaptive time step

It is important to carefully choose δt to get an optimal performance improvement. We observe the evolution of execution time as a function of $\delta t s$. Knowing that the initial value of $\Delta t \approx 0.5 s$ at the beginning of the simulation, we empirically choose δt values $\in [0.001 : 3.0] s$ (Figure 2.20).

We see that execution time falls steeply for small values of δt, then becomes stable and finally begins to grow again slowly as δt increases in $[0.5 : 3.0] s$. The latter phenomenon occurs when

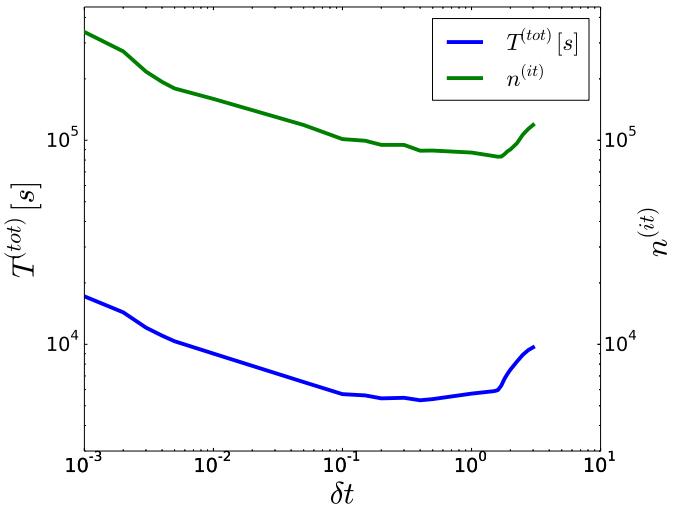


Figure 2.20: Execution time and number of iterations for a sequential simulation of the 2450 BP Pululagua eruption. In this simulation, 18×10^6 particles are injected into a domain of size $60km \times 60km \times 30km$. Parameter values: $\Delta t = 0.5s$ initially, $\Delta x = 500m$, $\delta t \in [0.001 : 3.0]\,s$.

more and more adjustments are rejected.

Now a crucial question is to determine if the dynamic adjustment of Δt involves a significant overhead. Figure 2.20 shows the evolution of the number of iterations until the end of the simulation as a function of δt as well as the corresponding simulation time. The curves are very similar.

This supports the claim that the use of an adaptive Δt scheme decreases the number of iterations without adding a significant amount of work when the value of δt is properly chosen.

2.6.2 Computational work

The total work performed in the simulation is the sum of the work W_i done at each iteration i, plus the initial and final work ($W^{(init)}$ and $W^{(final)}$). With $n^{(it)}$ the total number of iterations, we thus have

$$W^{(tot)} = W^{(init)} + W^{(final)} + \sum_{i=1}^{n^{(it)}} W_i \tag{2.45}$$

The work W_i consists of several parts which are :

- the work required for the computation of Eulerian velocities $n^{(s)} \cdot W^{(ed)}$ with $W^{(ed)}$ the work to compute one site and ns the number of sites in the domain;
- the work to compute the Lagrangian velocities and to move the particles $n_i^{(p)} \cdot W^{(lm)}$ with $W^{(lm)}$ the work to compute and move one particle, and $n_i^{(p)}$ the number of particles in motion at iteration i;
- the work involved for the particles that leave the domain, which can be neglected as this work occurs only once for each particle.

Thus, we can write $W_i = n^{(s)} W^{(ed)} + n_i^{(p)} W^{(lm)}$

Equation (2.45) can be rewritten (by taking into account that $W^{(init)} + W^{(final)} \ll \sum_{i=1}^{n^{(it)}} W_i$)

$$\begin{aligned} W^{(tot)} &= \sum_{i=1}^{n^{(it)}} (n^{(s)} W^{(ed)} + n_i^{(p)} W^{(lm)}) \\ &= n^{(it)} n^{(s)} W^{(ed)} + (\sum_{i=1}^{n^{(it)}} n_i^{(p)}) W^{(lm)} \\ &= n^{(it)} (n^{(s)} W^{(ed)} + n^{(p)} W^{(lm)}) \end{aligned} \tag{2.46}$$

where $n^{(p)} = \frac{1}{n^{(it)}} \sum_{i=1}^{n^{(it)}} n_i^{(p)}$ is the average number of particles in motion.

2.6.3 Sequential performances

We can directly deduce the sequential complexity from Equation (2.46).

Given the total work $W^{(tot)}$ (in clock cycles) performed in a simulation and $Freq[Hz]$ the frequency of the core, the total computation time for a sequential execution is

$$T^{(tot)} = \frac{W^{(tot)}}{Freq} \tag{2.47}$$

$$T^{(tot)} = n^{(it)} (n^{(s)} T^{(ed)} + n^{(p)} T^{(lm)}) \tag{2.48}$$

with $n^{(s)}$ the number of domain sites, $n^{(p)}$ the average number of particles in motion, $n^{(it)}$ the number of iterations, and the times $T^{(ed)} = W^{(ed)}/Freq$ and $T^{(lm)} = W^{(lm)}/Freq$.

We verified on some simulations that the execution time $T^{(tot)}$ indeed depends linearly on $n^{(p)}$ and $n^{(s)}$. Note that the average number of particles in motion in a simulation is itself proportional to the total number of particles injected in the domain.

We first observed the effect of the number of particles on the execution time. We fixed Δx, Δt and $n^{(it)}$ as well as the domain size (we didn't use the adaptive time step in this case) and varied the number of particles injected in the domain in order to confirm the linear dependence of the execution time on the number of particles (data not shown).

Next, we examined the effect of the number of domain sites on the execution time. Here, the number of injected particles, the domain size and the number of iterations were kept constant, while the number of sites was varied. Here again, we observed a linear dependence between the number of sites and execution time (data not shown).

Finally, we express the execution time $T^{(tot)}$ in terms of the time step Δt and the lattice spacing Δx. If we assume the physical time and Δt are constant in the simulation, then the number of iterations is directly proportional to $\frac{1}{\Delta t}$. Moreover, the number of sites $n^{(s)}$ can be expressed in terms of volume V of the domain and Δx as $n^{(s)} = V/(\Delta x)^3$.

By taking this into account and rewriting Equation (2.48), we get

$$T^{(tot)} = C\frac{1}{\Delta t}\left(\left(\frac{1}{\Delta x}\right)^3 \cdot V T^{(ed)} + n^{(p)} T^{(lm)}\right) \tag{2.49}$$

for some constant C.

With the adaptive Δt scheme, the value of Δt is no longer a constant. Recall that the goal of this optimization is to allow the fastest particles in the domain to perform the largest possible movement. Their motion is, however, limited by Δx. This hints towards the number of iterations $n^{(it)}$ not being anymore proportional to $\frac{1}{\Delta t}$, but to $\frac{1}{\Delta x}$. Therefore, we will assume that the number of iterations $n^{(it)}$ in Equation (2.48) is no longer proportional to $\frac{1}{\Delta t}$, but to $\frac{1}{\Delta x}$. By taking that into account, we get the following expression

$$T^{(tot)} = \hat{C}\frac{1}{\Delta x}\left(\left(\frac{1}{\Delta x}\right)^3 \cdot V T^{(ed)} + n^{(p)} T^{(lm)}\right) \tag{2.50}$$

for some constant $\hat{C}$.

To verify the model, we performed several simulations to record the execution time as a function of Δx with fixed physical simulation time and fixed number of particles injected into the domain. The adaptive Δt scheme is used. The results are shown in Figure 2.21. By adjusting the constants using the `curve_fit` function of `scipy`[2], we can make the theoretical curve fit the simulation data well. It seems therefore that the proposed performance model is satisfactory.

[2] Non-linear least squares method optimized with the Levenberg-Marquardt algorithm.

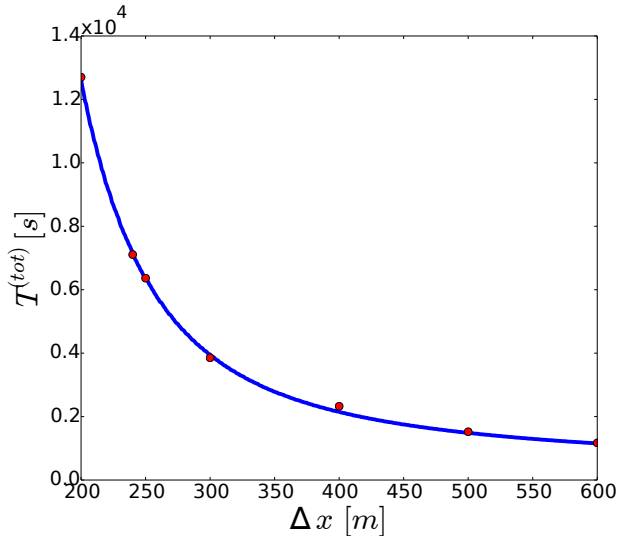


Figure 2.21: Sequential execution time of the simulation of the 2450 BP Pululagua eruption as a function of Δx with a domain of size $60km \times 60km \times 30km$ and $1 \cdot 10^5$ particles, using the adaptive Δt scheme. Theoretical curve with $\hat{C} \cdot T^{(ed)} \approx 1.411 \cdot 10^{-1}$ and $\hat{C} \cdot T^{(lm)} \approx 6.237$.

2.6.4 Parallel performances

In an ideal context, the execution time of a parallel implementation should equal the sequential time divided by the number of cores used. Unfortunately, the performances and scalability of a parallel application are limited mainly by three factors: network communications, work overhead and load imbalance. Their impact must be taken into account in a parallel performance model.

Communication time

For simplicity, we consider that all cores can communicate at the same time. In the block cyclic splitting of the domain, each domain block has 26 neighbours, implying that each core has virtually 26 neighbours. At each iteration, each core sends and receives from its neighbours a particle packet. Obviously, we don't know how many particles will travel from a domain block to another, but we can assume that for a given physical model, the average number of particles $n^{(p)}$ travelling between domain blocks at each iteration, is proportional to the number of particles injected into the domain.

So each core must send data to its 26 neighbours, first sending the data size, before transmitting the data itself. This implies that each communication pays 52 times the latency. The rest of the communication time is on average directly proportional to the number of particles in the domain. However, the latency contribution can be neglected if $n^{(p)}$ is large enough, which is

the case in our simulations. We will thus approximate the communication time by

$$T^{(com)} = C \cdot n^{(p)} \tag{2.51}$$

for some constant C.

Work overhead

Each particle that travels from a domain block to another is placed into a particle packet before being transmitted. Upon reception, the particles received must be inserted at the correct location in the recipient's local domain. We thus must add a term for this work, here again proportional to $n^{(p)}$

$$T^{(over)} = C \cdot n^{(p)} \tag{2.52}$$

for some constant C.

Load imbalance

Load imbalance is actually the factor that impacts the most performances in this application. In order to characterize load imbalance, let's define two notions: real work and apparent work.

Real work is simply the sum of the work achieved by each core during the execution of the algorithm, in other words, the sum of all clock cycles consumed by the CPUs during the computation. This can be written

$$W^{(r)} = \sum_{i=1}^{n^{(it)}} \sum_{j=1}^{n^{(proc)}} W_{ij} \tag{2.53}$$

where W_{ij} is the work performed by core $j \in \{1, \ldots, n^{(proc)}\}$ at iteration i. Note that the total work W_i at iteration i is given by $W_i = \sum_{j=1}^{n^{(proc)}} W_{ij}$.

cores must synchronize with each other at each iteration. Therefore, at each iteration, all cores must wait for the core with the heaviest load. It thus appears as if all cores seems had the same load as the core with the heaviest load at each iteration. We now define the apparent work as

$$W^{(a)} = n^{(proc)} \cdot \sum_{i=1}^{n^{(it)}} \max_{j=1,\ldots,n^{(proc)}} W_{ij} \tag{2.54}$$

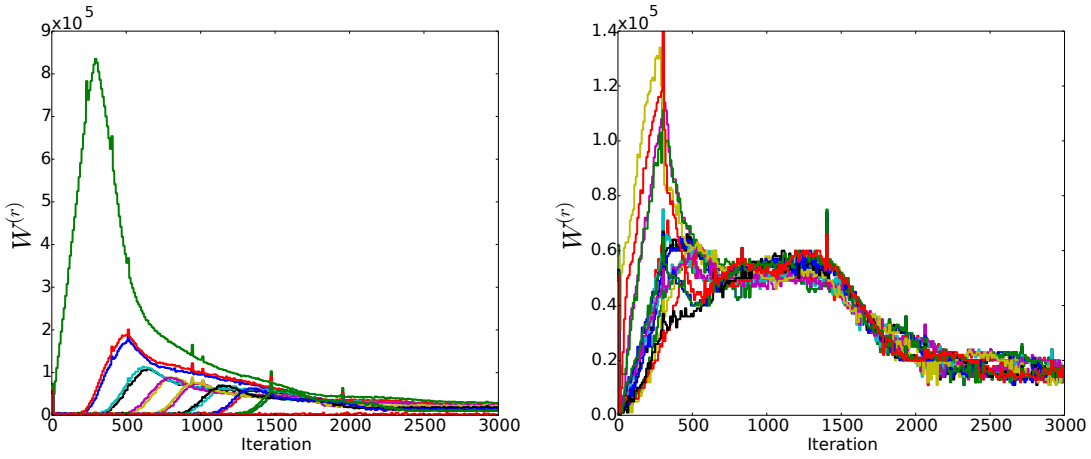
Then a natural indicator of load balance is the ratio

$$Eq = \frac{W^{(r)}}{W^{(a)}} \tag{2.55}$$

Load imbalance is then simply defined as the inverse of load balance.

In order to illustrate these definitions, let's consider a simulation with 1D domain splitting over 16 cores. The graph of real work $W^{(r)}$ is shown in Figure 2.22a. We observe a very high work load at the beginning of the execution. The load imbalance appears clearly during the first 10000 iterations. This execution shows a global load balance of $Eq \approx 0.45$, implying a load imbalance of $1/Eq \approx 2.2$.

Figure 2.22b shows the same execution but with the block cyclic domain splitting. This time, the load is better balanced between cores, and the value of the balance indicator is $Eq \approx 0.72$, representing an imbalance of $1/Eq \approx 1.4$. This is a big improvement over to the previous simulation and shows that the 3D block cyclic domain distribution helps to reduce the load imbalance.



(a) Unidimensional domain splitting.

(b) Block cyclic domain splitting.

Figure 2.22: Real work achieved by 16 CPUs in the first 3000 iterations of the 2450 BP Pululagua simulation, each colour representing the work of one core.

Execution time and speedup

We saw that the block cyclic domain splitting allows achieving better load balance. We thus examine the performance and speedup obtained with this implementation. Based on the

discussions of the previous section, we express the parallel execution time as

$$T^{(par)} = \frac{1}{Eq}\left(\frac{T^{(seq)}}{n^{(proc)}} + T^{(over)}\right) + T^{(com)} \tag{2.56}$$

with $Eq \in]0,1]$ the load balance, $T^{(seq)}$ the sequential execution time, $n^{(proc)}$ the number of cores used, $T^{(over)}$ the work overload due to parallelization and $T^{(com)}$ the communication time. We know that $T^{(over)}$ and $T^{(com)}$ are proportional to $n^{(p)}$. Unfortunately, we don't know the evolution of load balance as a function of $n^{(proc)}$. We know that load balance has a maximum value of 1 and it is likely that it decreases as $n^{(proc)}$ increases. Empirically, we state for testing that

$$Eq = (n^{(proc)})^{\frac{-1}{C}} \tag{2.57}$$

for some constant C could be an acceptable approximation.

Finally, the parallel execution time takes the following form

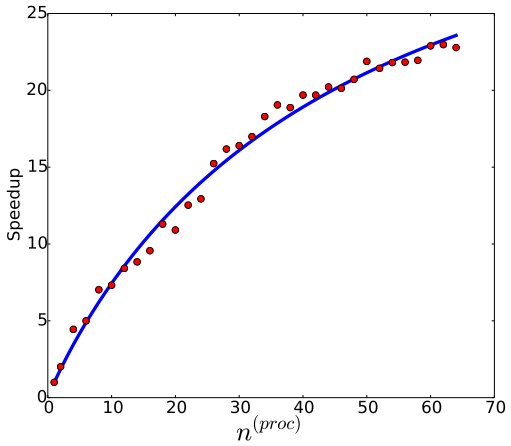
$$T^{(par)} = (n^{(proc)})^{\frac{1}{C_1}}\left(\frac{T^{(seq)}}{n^{(proc)}} + C_2 \cdot n^{(p)}\right) + C_3 \cdot n^{(p)} \tag{2.58}$$

with parameters C_1, C_2 and C_3 which allow tuning the load balance, the work overhead and the communication time.

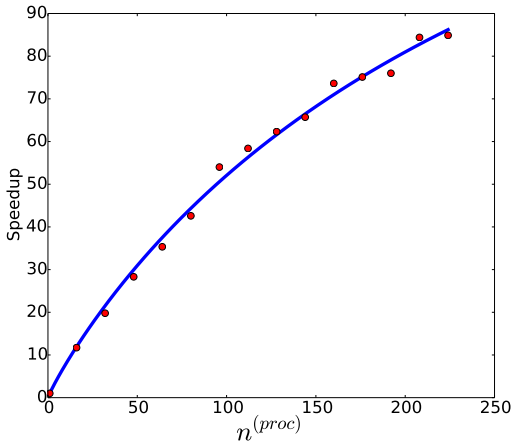
Three simulation cases were tested in order to observe the behaviour of the parallelized implementation and the accurateness of the performance model. The first two are simulations of the 2450 BP Pululagua eruption, with a small domain of $60km \times 60km \times 30km$ then with a larger domain of $200km \times 200km \times 30km$. The last is a simulation of the 1996 eruption of Ruapehu with a domain of size $650km \times 1050km \times 30km$. Each simulation case was run with $\Delta x = 500m$ and 10^6 particles per family (18 families for Pululagua and 23 for Ruapehu).

Figure 2.23 shows the speedups obtained. We see that the expression in Equation (2.58) is a good approximation for the parallel performances. Note that the average number of particles in motion depends on the domain size. This is due to the fact that in a smaller domain more particles will exit the domain during a given physical simulation.

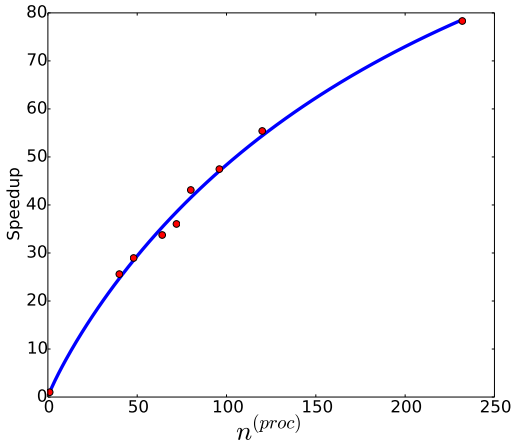
Figure 2.24 shows a comparison between theoretical speedup for the different simulation cases. We see that the larger the domain, the better the performances with Pululagua simulations. Alternately, we see that Ruapehu parallel simulations are less efficient. Indeed, while particles spread in every direction in the case of Pululagua, they mainly move along the same direction in the case of Ruapehu, which makes it a more difficult case to parallelize, particles not being well spread across the domain. Nonetheless, performance remains good thanks to the 3D domain block cyclic distribution.



(a) Domain size $60km \times 60km \times 30km$. Theoretical curve with $C_1 \approx 1.745 \cdot 10^1$, $C_2 \approx 6.463 \cdot 10^{-4}$ and $C_3 \approx 1.259 \cdot 10^{-10}$. $R^2 = 0.99$. Computation time with one core $\approx$ 10 hours.



(b) Domain size $200km \times 200km \times 30km$. Theoretical curve with $C_1 \approx 1.095 \cdot 10^1$, $C_2 \approx 4.546 \cdot 10^{-4}$ and $C_3 \approx 3.570{-}11$. $R^2 = 0.995$. Computation time with one core $\approx$ 48 hours.



(c) Domain size $650km \times 1050km \times 30km$. Theoretical curve with $C_1 \approx 1.013 \cdot 10^1$, $C_2 \approx 3.484 \cdot 10^{-4}$ and $C_3 \approx 1.284 \cdot 10^{-11}$. $R^2 = 0.996$. Computation time with one core $\approx$ 31 hours.

Figure 2.23: The speedup for two simulation cases of 2450 BP Pululagua eruption with 18×10^6 particles (a,b) and one simulation case of 1996 Ruapehu eruption with 23×10^6 particles (c), $\Delta x = 500m$.

2.7 Conclusion

In this chapter, we introduced the SSM and gMML formalisms of MMSF. We used them to describe the TTDM Tetras. We gave the details of the considered submodels and the underlying physical models. The overall model is compounded of the submodels *plume*, *source term*, *transport*, *advection field* and *atmosphere*.

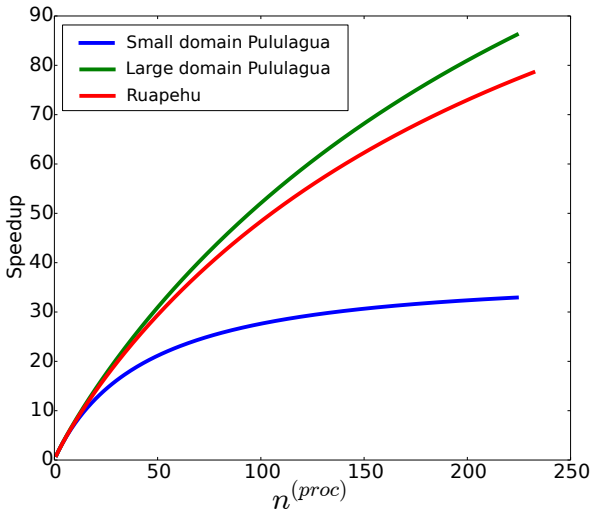


Figure 2.24: Theoretical speedup comparison between a small and a large domain for Pululagua eruption (respectively $60km \times 60km \times 30km$ and $200km \times 200km \times 30km$) and a domain of size $650km \times 1050km \times 30km$ for Ruapehu.

The central component of this design is the advection-diffusion C++ code, which embeds the submodels *source term*, *transport* and *advection field* and *plume* when working with Woods plume model. This software is based on a hybrid Eulerian-Lagrangian numerical scheme (Lagrangian on grid model). The numerical model offers multiple advantages. First, simulations remain unconditionally stable; second, it does not generate numerical diffusion; third, space which is represented as a Cartesian grid, can be easily subdivided, thus facilitating a parallel implementation. Finally, the modular structure of the numerical model makes possible the addition of more processes (e.g., particle aggregation) and allows for both the monitoring of atmospheric ash concentration and ground mass load at any given time during a simulation.

The implementation Tetras consists of multiple parts: a library dedicated to the implementation of the numerical model and its parallelization (PIAF) , and the software Tetras itself, which contains part of the physical model. It also integrates a particular optimization feature with the dynamic adaptation of the time step, which was shown to increase the efficiency of the simulator. Note that both the model and the simulation tool are flexible in the sense that the design is made to include new components for additional physical submodels.

The correctness of the advection and diffusion processes described in the model was validated (see Appendix A.2, we recall that turbulence was modelled using a diffusion process). We also determined the adequate number of particles to use in the simulations as a function of correlation thresholds. For our case studies, we obtained a mean correlation of 0.99 with a little more than 10^6 particles.

A performance model for the sequential implementation of Tetras is established, which takes into account the adaptive time step scheme. A parallel performance model which also encompasses load imbalance is then derived. These models were validated by recording execution times over many simulations. The computational efficiency of the parallel Lagrangian on grid model makes it suitable in the context of reproduction of past eruptions, for real time and ensemble forecasting as well as inversion of eruptions source parameters when a parallel optimization strategy is used.

Agreement between simulated and field data is good, but we also illustrate the limitations of the proposed particles insertion and transport scheme. The source term is considered to be a point, and particles are transported according to plume, umbrella cloud and wind velocity fields, together with diffusion generated by random velocities. This way of coupling together physical models makes it difficult to reproduce proximal deposits, even in no wind condition as shown by simulations of the Pululagua eruption. We propose in the next chapter a new formulation of the source term that aims to cope with this issue.

3 The spatially extended exit rate model family, a new definition of source term

The content presented in this chapter is original unpublished work.

3.1 Introduction

To model tephra transport and deposition, TTDMs apply a transport process governed by wind, diffusion and sedimentation to particles released from a source term in a simulation domain. The transport is then applied through an Eulerian finite difference scheme [61, 102], particle-puff [7, 74] or Lagrangian particles framework [104, 97].

A crucial question is the characterization of the source term : where, and in what quantity particles are released in the atmosphere. The source term can be implemented in the transport code with various levels of complexity, such as a point [61, 7], a set of point [61], a line or a distribution of density above the vent [38, 102]. More recent works have used a disk source term to capture the geometry of the umbrella cloud [40]. Then quantity of particles injected in the atmosphere from a source term can be characterized by a distribution of particles [109, 38] or a buoyant plume theory (BPT) model [123, 49, 48, 60, 86]. Fully resolved 3D plume model exists as well [34], but they are too computationally intensive to be used in real time forecasting or inversion context.

For example, Ash3d [102] and Tephra2[38] consider a vertical distribution of particles as the source term. Fall3D-8 [61] can account for a source term made of a set of points that emit a flow of particles governed by a BPT model. In vol-calpuff [7], there is as well a BPT model taking into account particles fall out and re-entrainment, but particles are only injected at the top of the plume in the transport model, in the form of particle puff. NAME [74, 12, 11] can as well account for a vertical distribution of particles, or insertion at the top of a buoyant plume model.

While those ways of characterizing source terms allow for a good reproduction of long-range transport of volcanic ash and observed deposits, it does not originate from first principles (in

the case of a source term characterized by a vertical distribution of mass) and lack the ability to capture the spatial geometry of the volcanic plumes. This is usually alleviated by a diffusion coefficient applied to the particles, which can be unrealistically high.

In the previous chapter, we proposed a solution where the source term is a point located at the vent. A BPT model is used to approximate velocity of particles in the transport code and diffusion is applied to account for turbulent processes in the plume and atmosphere. However, the reproduction of the proximal deposition has shown to be difficult with this technique.

In this chapter, we propose a model that describes the movement of particles in the plume and umbrella cloud from the moment they exit the vent. They are subject to a process of transport, governed by the velocity of the surrounding material, and a process of exit, governed by gravity and re-entrainment. While this type of model describes a source term that respects the geometry of a plume and umbrella cloud, we call it the Spatially Extended Exit Rate (SEER) model family.

We developed four SEER based models :

Simplified-analytical-seer which is an analytical solution for a simplified plume and umbrella cloud geometry.

Volcano-semianalytical-seer which is the coupling of simplified-analytical-seer with a BPT model to simulate tephra deposition of strong plumes eruptions in no wind condition with a fast-solving time.

Volcano-lagrangian-seer which is the coupling of a numerical Lagrangian implementation of SEER source term with BPT plume models and a Lagrangian transport model to simulate tephra transport in any condition (with time and space varying wind field for example).

Simplified-lagrangian-seer which is a numerical Lagrangian implementation that mimics simplified-analytical-seer and is used to validate the numerical scheme of volcano-lagrangian-seer.

In volcano-lagrangian-seer, we simulate individual particles through a pure Lagrangian scheme. This numerical scheme is easy to parallelize as long as particle interaction or atmospheric ash concentration is not desired. If particle interaction is wanted, a Lagrangian on grid numerical model would be preferable (see Chapter 2). The flexible design of Tetras (also Chapter 2) allows us to use the same definition of the physical models, and usage of the Lagrangian on grid numerical scheme would be easily doable.

Volcano-lagrangian-seer and simplified-lagrangian-seer are parallelized in a distributed memory architecture using MPI. We observe the strong scaling behaviour of the implementation. The good speedup allows solving the model in a matter of seconds when enough computing cores are available. This makes possible the integration of volcano-lagrangian-seer in

an optimization or inference framework, and thus makes eruption source parameters (ESP) inversion possible either with volcano-semianalytical-seer on a standard computer or with volcano-lagrangian-seer in a High Performance Computing (HPC) infrastructure.

By solving the model analytically in simplified-analytical-seer, we show that specific physical conditions have to be met in order to obtain a deposit of particles on the ground which is continuous between deposit from the plume and from the umbrella cloud under the plume corner area. The question remains open to know if this condition is actually met in reality, or if this discontinuity is not observed on the field due to atmospheric turbulent diffusion.

We propose a simple mechanistic model with three resolutions schemes (analytical, semi-analytical and numerical) and four implementations (simplified-analytical-seer, volcano-semianalytical-seer, volcano-lagrangian-seer, simplified-lagrangian-seer), suitable for reproduction of the deposit of past eruptions or ESP inversion (including plume initial speed, radius and temperature) without relying on excessive diffusion to capture processes such as umbrella cloud spreading. The correspondence of the analytical formulation with the numerical one is validated. We show comparison between volcano-lagrangian-seer simulations and field data for three past eruptions ; Pululagua 2450BP (strong plume without wind), Cotopaxi layer 3 (strong plume with wind) and Ruapehu 1996 (weak plume). We compare as well those data with another model routinely used for inversion (Tephra2). The numerical model is parallelized and a performance analysis is carried out.

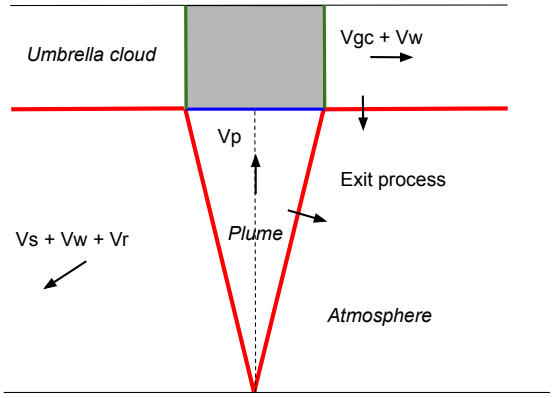
3.2 Model description

We consider the two eruptions scenarios seen in Chapter 2, namely the strong plume and the weak plume. In the case of a strong plume, the plume material rises up to the neutral buoyancy level (NBL) and starts to spread horizontally as a gravity current (GC) and forms an umbrella cloud (UC).

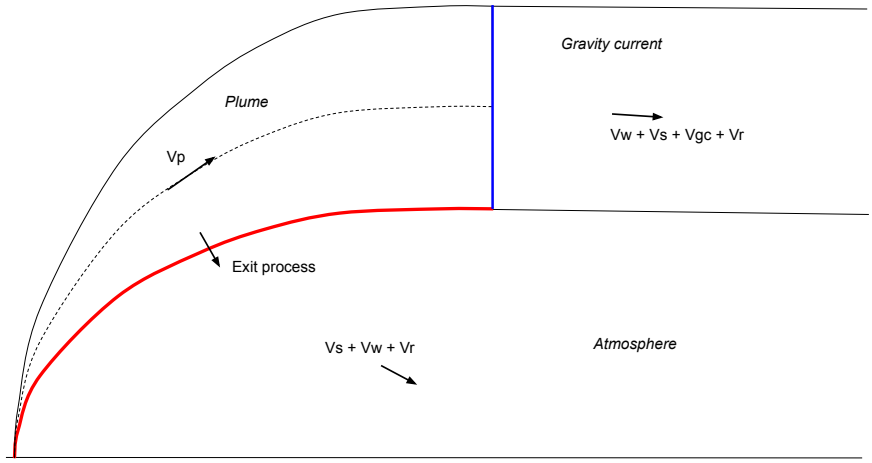
In the usual case of a strong plume eruption occurring in a windy atmosphere, the shape of the UC results from an interplay between wind and GC velocities and is approximately the shape of an ellipse. In more unusual situation of a strong plume eruption occurring in a steady atmosphere, the spreading velocity of the UC is the same in every direction, and the resulting shape is closer to a circle. If the wind is strong enough compared to plume speed, the plume becomes bent over and the GC spread mainly in the wind direction.

3.2.1 Overview of the model

For a strong plume, space is divided into three regions : the plume, the umbrella cloud and the atmosphere. SEER model is applied in the plume and umbrella cloud, while an advection-diffusion model is applied in the atmosphere. Particles are first considered to be injected at the plume base and rise at the velocity of the plume fluid V_p. They can exit the plume through



(a)



(b)

Figure 3.1: (a) Regions considered for a strong plume. Particles in the plume are subject to a rising vertical velocity and can exit the plume by the border (red) due to an exit process governed by exit rates. Once they reach the top of the plume (blue), they are transported to the entry of the umbrella cloud (green) and are then subject to a horizontal velocity which is the sum of the gravity current and wind speed. Particles can exit the umbrella cloud region by an exit process governed by exit rates. (b) Regions considered for a weak plume. Particles in the plume are subject to the plume velocity that is collinear to the plume centreline. They can exit the plume by the plume border (red) due to an exit process governed by exit rates. Once they reach the top of the plume (blue) they are scattered uniformly on the plume top. They are then transported by an advection-diffusion process. Ground deposit appears to be too concentrated along a line when SEER model is applied inside GC in the case of a weak plume, advection-diffusion is then applied after particles exit the plume.

the plume border (red in Figure 3.1a) by an exit process. Particles reaching the top of the plume (blue in Figure 3.1a) are considered to be injected at the entrance of the umbrella cloud

with a random direction (green in Figure 3.1a). Particles entering the umbrella cloud are then transported by a horizontal velocity that is the sum of the gravity current velocity V_{gc} and the wind velocity V_w. They can exit the umbrella cloud by the base of this one through an exit process. Particles exiting the plume or umbrella cloud are then transported in the atmosphere by an advection-diffusion process until they reach the ground. In the atmosphere, they are subject to sedimentation velocity V_s, wind velocity and random velocities V_r that generate a diffusion coefficient (see Chapter 2 and Appendix A).

For a weak plume, space is also divided in three regions : the plume, the gravity current and the atmosphere. However, on the contrary of the strong plume, SEER model is applied only in the plume, and advection-diffusion is applied in the gravity current and in the atmosphere. Indeed, ground deposits of particles were found to be too narrow (not scattered enough around wind transport direction axis) if a SEER model without diffusion is applied in the gravity current region of a bent-over plume. This hints toward the interpretation that turbulent diffusion plays an important role in the dynamic of the GC. Particles are injected at the plume base and transported by plume velocity. They can exit the plume by a plume border (red in Figure 3.1b). Particles reaching the end of the plume (blue in Figure 3.1b) are scattered uniformly on the plume top. They are then subject to wind, sedimentation, random and gravity current velocities. When particles enter the atmosphere region, they are no more subject to gravity current velocity and are transported until they reach the ground.

It is important to note that in the current formulation, particles leaving the SEER model cannot re-enter it, and are subject to advection-diffusion only until they reach the ground. The SEER model is then a source term of the advection-diffusion model.

3.2.2 Vertical plume

Let us consider a volcanic plume rising vertically. Particles distributed into classes of size ϕ are injected at the vent and are rising at a given plume velocity $u(z)$ with z the height. The plume border is defined by the plume radius $r(z)$.

By adopting a Lagrangian point of view following particles individually, each particle has a probability to exit the plume at each second by plume margins $p_\phi(z)$ at height z. We call this probability an exit rate.

By adopting a Lagrangian point of view following a growing control volume of the shape of a cylinder of thickness dz and radius $r(z)$ moving at speed $u(z)$, the control volume loose particles from its margins as

$$\frac{\partial N_\phi(t)}{\partial t} = -p_\phi(z) N_\phi(t) \tag{3.1}$$

where $N_\phi(t)$ is the number of particles of size ϕ in the control volume.

From an Eulerian point of view, one can divide the plume in fixed control volumes (slices) of radius $r(z)$ and height dz. $Q_{z\phi}(z,t)$ is the flow of ϕ-sized particles crossing a horizontal plane at height z and time t and $Q_{r\phi}(z,t)$ is the flow of ϕ-sized particles exiting a slice by the margins at height z and time t.

Then, the variation in time of the number of particles of size ϕ in a slice at height z can be written

$$\frac{\partial N_\phi(z)}{\partial t} = Q_{z\phi}(z,t) - Q_{z\phi}(z+dz,t) - Q_{r\phi}(z,t) \tag{3.2}$$

in a steady state where the number of particles in a slice remains constant over time $\partial_t N_\phi(z) = 0$. Thus the flow of exiting particles from a slice can be written

$$Q_{r\phi}(z) = Q_{z\phi}(z) - Q_{z\phi}(z+dz) = p_\phi(z) N_\phi(z) \tag{3.3}$$

Let us now consider that particles are evenly distributed in each slice. If we assume that the flux of particles exiting a slice at height z by its side $J_{r\phi}(z)$ is proportional to the density of particles in the slice $\rho_\phi(z)$ in particles per unit volume, we can write

$$J_{r\phi}(z) = v_{e\phi}(z)\rho_\phi(z) \tag{3.4}$$

where $v_{e\phi}(z)$ is an exit velocity specific to each particle class.

From equations 3.3 and 3.4, we can write

$$\begin{aligned} Q_{r\phi}(z) &= J_{r\phi}(z) 2\pi r(z) dz \\ &= v_{e\phi}(z) \frac{N_\phi(z)}{\pi r^2 dz} 2\pi r(z) dz \\ &= \frac{2 v_{e\phi}(z) N_\phi(z)}{r(z)} \\ &= p_\phi(z) N_\phi(z) \end{aligned} \tag{3.5}$$

Thus, the exit rate in that case is

$$p_\phi(z) = \frac{2 v_{e\phi}(z)}{r(z)} \tag{3.6}$$

3.2.3 Circular umbrella cloud

Let us now consider the model of transport inside a radially spreading circular umbrella cloud on top of the previously considered plume. Bottom of the umbrella cloud is at height z_h and

the top is at height $z_h + h$, h being the thickness of the umbrella cloud. Particles reaching the height z_h are injected into the umbrella cloud in a random position on a ring centred on the plume of radius $R_h = r(z_h)$. Particles inside the umbrella cloud are transported horizontally and radially to the plume centre at a given velocity $U(R)$.

Again, if we adopt a Lagrangian point of view following individual particles, each particle has a probability to exit the umbrella cloud by the umbrella cloud base at each second P_ϕ.

Then, by following a control volume of the shape of an empty cylinder (or shell) of interior radius R and exterior radius $R + dR$ expanding at velocity $U(R)$, the number of particles lost by the bottom of the volume is

$$\frac{\partial N_\phi(t)}{\partial t} = -P_\phi N_\phi(t) \tag{3.7}$$

We can divide the umbrella cloud into shells of thickness dR at radius R. $Q_{R\phi}(R)$ is the flow of particles crossing a vertical plane at radius R, and $Q_{z\phi}(R)$ is the flow of particles crossing a horizontal plane leaving a shell.

The variation in time of the number of particles in a shell at radius R can be written

$$\frac{\partial N_\phi(R)}{\partial t} = Q_{R\phi}(R) - Q_{z\phi}(R) - Q_{R\phi}(R + dR) \tag{3.8}$$

in a steady state where the number of particles in a shell remains constant over time $\partial_t N_\phi(R) = 0$. Thus the flow of exiting particles from a shell can be written

$$Q_{z\phi}(R) = Q_{R\phi}(R) - Q_{R\phi}(R + dR) = P_\phi(z) M_\phi(R) \tag{3.9}$$

We consider particles to be evenly distributed in each shell. We assume that the flux $J_{z\phi}(R)$ of ϕ-sized particles exiting a shell at radius R by the bottom is proportional to the density of particles in the shell $\rho_\phi(R)$ in number of particles per unit volume. We can then write

$$J_{z\phi}(R) = V_{e\phi} \rho_\phi(R) \tag{3.10}$$

where $V_{e\phi}$ is an exit velocity specific to each particle class.

From combining equations 3.9 and 3.10, we can write

$$\begin{aligned} Q_{z\phi}(R) &= J_{z\phi}(R)2\pi R dR \\ &= V_{e\phi}\frac{N_\phi(R)}{2\pi R dR h}2\pi R dR \\ &= V_{e\phi}\frac{N_\phi(R)}{h} \\ &= P_\phi N_\phi(R) \end{aligned} \tag{3.11}$$

Thus, in that case, the exit rate is

$$P_\phi = \frac{V_{e\phi}}{h} \tag{3.12}$$

3.2.4 Simplified-analytical-seer, an analytical solution for a simplified setup

We consider now a simplified setup where the vertical velocity and exit velocity in the plume are constant and where radius varies linearly with height. Horizontal velocity and exit velocity in the umbrella cloud are considered constant as well and the thickness of the umbrella cloud is constant too.

Even if those are large simplifications compared to the actual physics of a volcanic plume, this model allows us to highlight the fact that sedimentation from the plume and from umbrella cloud is not necessarily continuous, to produce a fast approximation of deposit and to validate the Lagrangian numerical model later on. We call this model simplified-analytical-seer.

The rest of the demonstration applies to any class of particle, the ϕ subscript is omitted for clarity.

Plume

Let us consider a slice of thickness dz located at height z. The number of particles in this slice is

$$dN(z) = \rho(z)\pi r^2(z)dz \tag{3.13}$$

Particles enter and leave the slice from its lower and upper sides. Let us denote $J_z(z)$ the flux of particles crossing a horizontal plane at height z. The balance of entering and exiting particles within the slice reads

$$\pi r^2(z)dz\frac{\partial\rho(z)}{\partial t} = J_z(z)\pi r^2(z) - J_z(z+dz)\pi r^2(z+dz) - 2\pi r(z)J_r(z)dz \tag{3.14}$$

which simplifies as

$$dz\frac{\partial\rho(z)}{\partial t} = J_z(z) - J_z(z+dz)\frac{r^2(z+dz)}{r^2(z)} - 2\frac{J_r(z)}{r(z)}dz \tag{3.15}$$

one has

$$J_z(z+dz) = J_z(z) + dz\frac{\partial J_z}{\partial z} \quad \text{and} \quad r(z+dz) = r(z) + dz\frac{\partial r}{\partial z}$$

thus

$$\frac{r(z+dz)}{r(z)} = 1 + \frac{dz}{r(z)}\frac{\partial r}{\partial z} \quad \text{and} \quad \left(\frac{r(z+dz)}{r(z)}\right)^2 = 1 + 2\frac{dz}{r(z)}\frac{\partial r}{\partial z}$$

equation 3.15 then becomes

$$\frac{\partial\rho(z)}{\partial t} = -\frac{\partial J_z}{\partial z} - \frac{2J_z(z)}{r(z)}\frac{\partial r}{\partial z} - 2\frac{J_r(z)}{r(z)} \tag{3.16}$$

We need now to specify how the fluxes depend on ρ. For an exit velocity that does not change with height $v_e(z) = v_e$ we have

$$J_z(z) = \rho(z)u(z) \quad \text{and} \quad J_r(z) = v_e\rho(z) \tag{3.17}$$

Again, if we consider a steady state for which the number of particles in the slice is constant over time, we have

$$\frac{\partial\rho}{\partial t} = 0$$

and equation 3.16 becomes

$$\frac{\partial u(z)\rho(z)}{\partial z} + \frac{2\rho(z)u(z)}{r(z)}\frac{\partial r}{\partial z} + 2v_e\frac{\rho(z)}{r(z)} = 0 \tag{3.18}$$

Let us now make another simplification and assume that the plume velocity $u(z)$ does not depend on z and that $r(z)$ grows linearly with z

$$u(z) = u \quad r(z) = az$$

With these simplifications, equation 3.18 reads

$$u\frac{\partial\rho}{\partial z} + \frac{2au}{az}\rho + \frac{2v_e}{az}\rho = 0 \tag{3.19}$$

which can be written as

$$\frac{\partial\rho}{\partial z} = -\frac{2}{z}\left(1 + \frac{v_e}{au}\right)\rho = -b\frac{\rho}{z} \tag{3.20}$$

where

$$b = 2\left(1 + \frac{v_e}{au}\right) \tag{3.21}$$

Equation 3.20 is rewritten as

$$\frac{d\rho}{\rho} = -b\frac{dz}{z}$$

and can be integrated as

$$\int_{\rho(z_0)}^{\rho(z)} \frac{d\rho}{\rho} = -b\int_{z_0}^{z} \frac{dz}{z}$$

which gives the solution

$$\rho(z) = \rho(z_0)\left(\frac{z}{z_0}\right)^{-b} \tag{3.22}$$

Thus, the number of particles that leave the plume laterally from a slice of thickness dz is

$$\begin{aligned} 2\pi r(z) J_r(z) dz &= 2\pi r(z) v_e \rho(z) dz \\ &= 2\pi r(z) v_e \rho(z_0)\left(\frac{z}{z_0}\right)^{-b} dz \\ &= 2\pi z v_e \rho(z_0)\left(\frac{z}{z_0}\right)^{-b} dr \end{aligned} \tag{3.23}$$

where we used that $r(z) = az$ and $adz = dr$.

The plume velocity is u and the exit velocity is v_e. Thus, there is a power law decay of the number of particles falling off the plume as the height increases.

Assuming that the particles fall vertically from where they leave the plume (no wind and no diffusion in the atmosphere), we have that the rate of the number of ϕ-sized particles $dN_\phi(r)$ that deposit on the ground per unit time in the ring of radius r and width dr is

$$dN_\phi(r) = 2\pi v_{e\phi} \rho_\phi(z_0) z\left(\frac{z}{z_0}\right)^{-b_\phi} dr \tag{3.24}$$

with

$$b_\phi = 2\left(1 + \frac{v_{e\phi}}{au}\right) \tag{3.25}$$

where we reintroduced the ϕ particle class notation.

Umbrella cloud

The conservation equation reads (again for any class of particles)

$$2\pi R h dR \frac{\partial \rho}{\partial t} = 2\pi R h J_R(R) - 2\pi(R + dR) h J_R(R + dR) - 2\pi R dR J_z(R) \tag{3.26}$$

As before, we assume that the radial current is

$$J_R(R) = U(R)\rho(R) \tag{3.27}$$

Equation 3.26 becomes

$$dR\frac{\partial \rho}{\partial t} = J_R(R) - \left(1 + \frac{dR}{R}\right) J_R(R + dR) - \frac{dR}{h} J_z(R) \tag{3.28}$$

With

$$J_R(R + dR) = J_R(R) + dR\frac{\partial J_R}{\partial R} \tag{3.29}$$

we obtain

$$dR\frac{\partial \rho}{\partial t} = J_R(R) - \left(1 + \frac{dR}{R}\right)\left(J_R(R) + dR\frac{\partial J_R}{\partial R}\right) - \frac{dR}{h} J_z(R) \tag{3.30}$$

which reduces to

$$dR\frac{\partial \rho}{\partial t} = -\frac{J_R}{R} dR - dR\frac{\partial J_R}{\partial R} - \frac{dR}{h} J_z(R) \tag{3.31}$$

and finally to

$$\partial_t \rho = -\frac{J_R}{R} - \frac{\partial J_R}{\partial R} - \frac{J_z}{h} \tag{3.32}$$

With equations 3.10 and 3.27, we get in a steady state

$$0 = -\frac{\rho U}{R} - \frac{\partial \rho U}{\partial R} - \frac{V_e \rho}{h} \tag{3.33}$$

For U, h and V_e independent of R, we obtain

$$\frac{d\rho}{dR} = -\left(\frac{1}{R} + \frac{V_e}{Uh}\right)\rho \tag{3.34}$$

which can then be cast as

$$\frac{d\rho}{\rho} = -\frac{dR}{R} - \frac{V_e}{Uh} dR \tag{3.35}$$

which can be integrated as

$$\int_{\rho(R_0)}^{\rho(R)} \frac{d\rho}{\rho} = -\int_{R_0}^{R} \frac{dR}{R} - \frac{V_e}{Uh}\int_{R_0}^{R} dR \tag{3.36}$$

which gives

$$\ln \rho(R) - \ln \rho(R_0) = -(\ln R - \ln R_0) - \frac{V_e}{Uh}[R - R_0] \tag{3.37}$$

Then the solution is

$$\rho_{uc}(R) = \rho_{uc}(R_0)\frac{R_0}{R}\exp\left(-\frac{V_e}{U}\frac{R-R_0}{h}\right) \tag{3.38}$$

where we added the subscript uc to make it clear that this density refers to the umbrella cloud density.

The condition to determine $\rho_{uc}(R_0)$ is that at some radius

$$R_0 = R_h = r(z_h) \tag{3.39}$$

where the two models are coupled, the flow of particles leaving the plume is the same as the flow of particles entering the umbrella cloud. So the flux times the surface should match.

$$2\pi R_h h J_R(r(z_h)) = \pi R_h^2 J_z(z_h) \quad \Rightarrow \quad 2Uh\rho_{uc}(R_h) = uR_h\rho_p(z_h) \tag{3.40}$$

where we call ρ_p the density in the plume, as defined in Equation 3.22.

Equation 3.40 then reads

$$\rho_{uc}(R_h) = \frac{uRh}{2Uh}\rho_p(z_h) \tag{3.41}$$

or in more details

$$\rho_{uc}(R_h) = \frac{uRh}{2Uh}\rho_p(z_0)\left(\frac{z_h}{z_0}\right)^{-b} \tag{3.42}$$

And finally, the density of particles in the umbrella cloud as a function of its distance to the vent is

$$\rho_{uc}(R) = \frac{u}{U}\frac{R_h^2}{2Rh}\rho_p(z_h)\exp\left(-\frac{V_e}{U}\frac{R-R_h}{h}\right) \tag{3.43}$$

At location R the deposition rate dN_ϕ of a given class of particle due to particles falling vertically from the umbrella cloud in a ring of radius R and width dR is

$$dN_\phi = 2\pi R V_{e\phi}\rho_{uc\phi}(R)dR \tag{3.44}$$

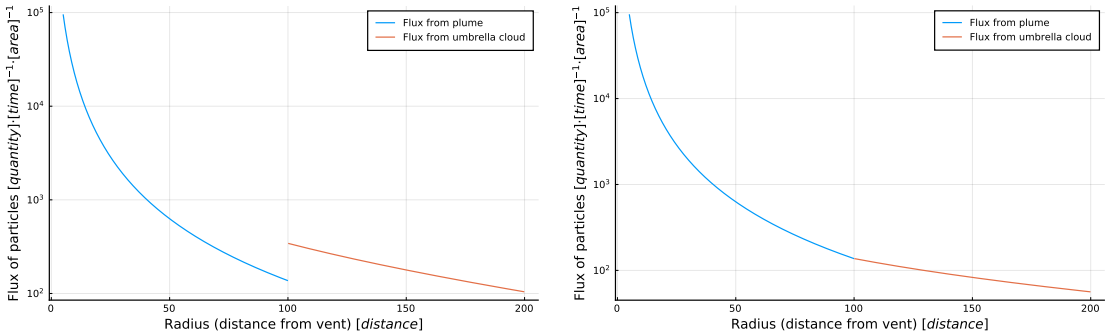
Table 3.1 shows the list of parameters of simplified-analytical-seer model.

3.2.5 Continuity condition

Figure 3.2 shows the flux of particles depositing on the ground from the plume and from the umbrella cloud. Figure 3.2a shows a result for arbitrary values of velocity, exit velocities and arbitrary geometry. We see that the deposition rates from the plume and umbrella cloud are not guaranteed to be continuous.

Symbol	Definition
z_0	height of particles insertion
ρ_{z_0}	particle density at z_0
z_h	height of the umbrella cloud base
h	thickness of the umbrella cloud
a	slope of the plume border
u	vertical speed
U	horizontal speed
v_e	exit velocity in the plume
V_e	exit velocity in the umbrella cloud

Table 3.1: Volcano-semianalytical-seer parameters.



(a) $v_{e\phi} = 0.05, V_{e\phi} = 0.05, R_h = 100, h = 10, a = 0.5, u = 1, U = 1$.

(b) $v_{e\phi} = 0.05, V_{e\phi} = 0.02, R_h = 100, h = 10, a = 0.5, u = 1, U = 1$.

Figure 3.2: Flux of particles reaching the ground according to equations 3.24 and 3.44 for two sets of parameters. Units are arbitrary. In (a), parameters does not respect Equation 3.47 while in (b) they do.

Then, for $R = R_h$, we want to establish the condition to have the same deposition rate from the plume and from the umbrella cloud for a given class of particle in a ring of radius R_h and width dR. By comparing equation 3.24 with equation 3.44, we get

$$2\pi v_{e\phi} z_h \rho_{p\phi}(z_h) dR = 2\pi R_h V_{e\phi} \rho_{uc\phi}(R_h) dR \tag{3.45}$$

or equivalently

$$v_{e\phi} z_h = R_h V_{e\phi} \frac{u R_h}{2Uh} \tag{3.46}$$

which, with $z_h = R_h / a$, reduces to

$$v_{e\phi} = a V_{e\phi} \frac{u}{U} \frac{R_h}{2h} \tag{3.47}$$

Figure 3.2b shows a deposition pattern for a specific case where this continuity condition is met.

3.2.6 Quantification and generalization of the exit velocities

Previous sections show the solution of the deposition of particles for a simplified geometry with arbitrary constant exit velocities. Bursik et al. [30] propose that the sedimentation rate from the base of the umbrella cloud should be equal to the sedimentation velocity of particles and that sedimentation from plume borders should be proportional to sedimentation velocity and dependent on the plume geometry, where the more vertical the plume border is, the higher is the probability of a particle exiting the plume to be re-entrained.

We try to capture the same process of exit and re-entrainment with the following simple assumption

$$v_{e\phi}(z,\theta) = -\sin(\theta)\, v_{s\phi}(z) \tag{3.48}$$

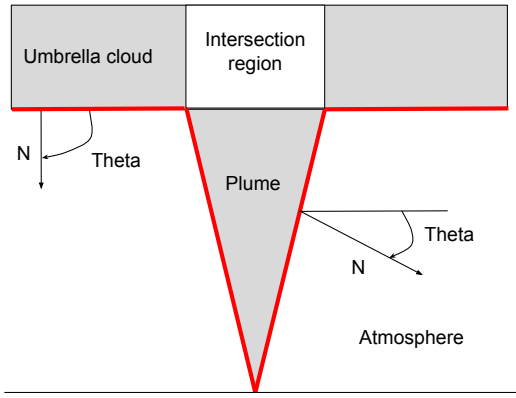
where $v_{s\phi}$ is the sedimentation velocity of particles and θ is the angle with the horizontal of the vector normal to the plume or umbrella cloud border, pointing to the exterior of the plume or umbrella cloud (Figure 3.3). We thus consider that the orientation of the surface through which the particle is exiting is the determining factor of its exit velocity. The more this surface is horizontal, the higher the exit velocity is, with the maximum being the sedimentation velocity of the particle (case that is reached in the umbrella cloud).

Informally, that means that a particle exiting through a sub-vertical plume border will surely be re-entrained, while a particle exiting through a horizontal border (from the umbrella cloud) cannot be re-entrained. By definition, this angle is $-\frac{\pi}{2}$ in the umbrella cloud, then

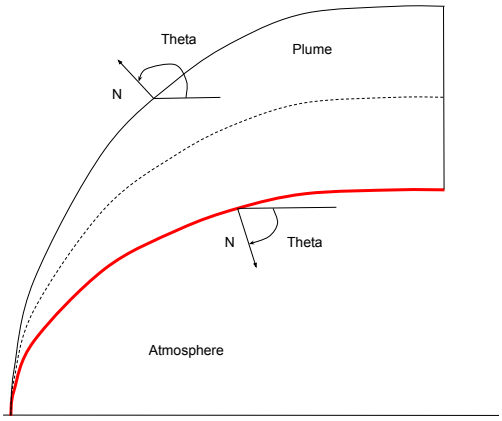
$$V_{e\phi}(z) = v_{s\phi}(z) \tag{3.49}$$

This formulation can be applied to any plume or umbrella cloud geometry. Particles exit radially to the plume centreline. Then, for a strong plume, the exit velocity depends only on the height, while θ is the same for any direction at a given height. For a weak plume, however, the exit velocity depends on the coordinate along the plume centreline and on the exit direction γ. This assumption allows defining an exit velocity for any geometry.

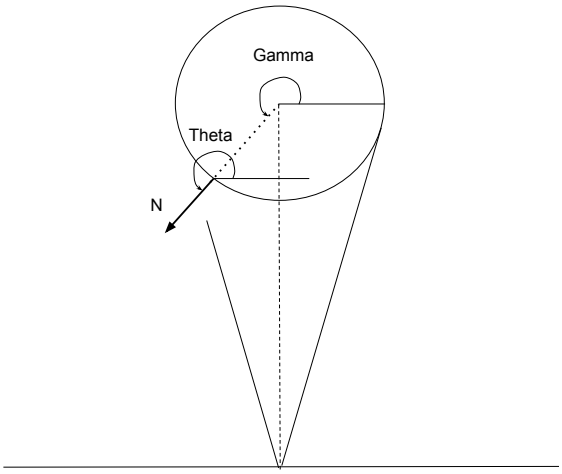
That means that when the exit surface points toward the ground, the exit velocity is positive. Otherwise, the exit velocity is ≤ 0, which means that particles cannot exit the plume or umbrella cloud through such a surface.



(a)



(b)



(c)

Figure 3.3: Illustration of the exit angle θ and exit direction γ for a strong plume and a weak plume. $\gamma = \theta$ in the front view of the weak plume, this is a specific case. Vectors N are normal to the plume border or base of the umbrella cloud.

3.2.7 Volcano-semianalytical-seer, solving for a volcanic eruption

To reproduce the deposition of a volcanic eruption, it would be possible to solve the simplified-analytical-seer model by picking appropriate parameters shown in Table 3.1. However, in that case, parameters would not be linked to any physical consideration.

We then propose a solution where parameters of simplified-analytical-seer are approximated based on physical models. We thus make a coupling between plume, umbrella cloud, terminal velocity and simplified-analytical-seer models and name this implementation volcano-semianalytical-seer.

For this, we compute simplified-analytical-seer model parameters as follows :

$\boldsymbol{z_0}$ height of particles insertion above vent. It is a parameter of simplified-analytical-seer that has to be set because density of particles in the plume cannot be computed at vent where radius is considered to be 0. It is not deduced from another model.

$\boldsymbol{\rho_{z_0}}$ density of particles in number of particles per unit volume at z_0. This is directly deducted from the total injected mass M, TGSD and characteristics of particles. It is considered that all the mass is injected during a time $t = 1s$ in a volume of $v = \pi r(z_0)^2 ut$.

$\boldsymbol{z_h}$ height at which particles are released from the base of the umbrella cloud. This is determined by the plume model (Chapter 2, Section 2.2.2).

$\boldsymbol{h}$ is the thickness of the umbrella cloud (Chapter 2, Section 2.2.2).

$\boldsymbol{a}$ is the slope of the plume border with respect to the vertical. This is considered to be the slope of the line originating at vent to the intersection of the plume model border with z_h (Figure 3.4a).

$\boldsymbol{u}$ is the vertical plume speed, set as the mean of plume velocity profile.

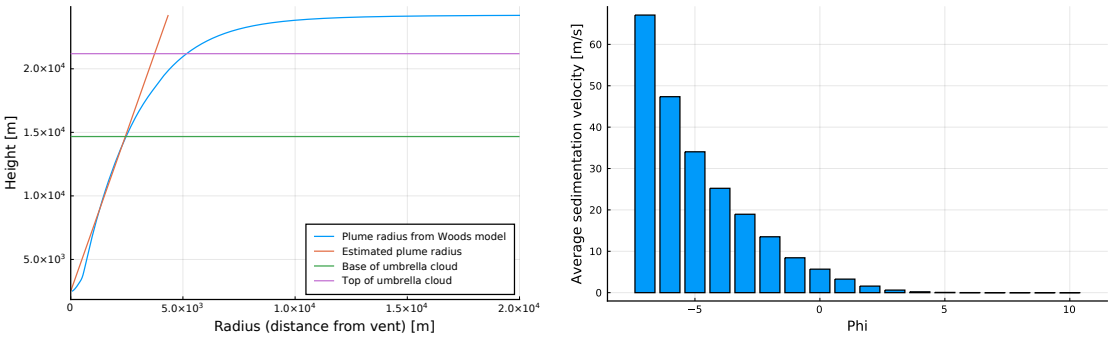
$\boldsymbol{U}$ is the horizontal umbrella cloud speed, set as the mean of the umbrella cloud speed profile.

$\boldsymbol{v_e}$ is the exit velocity in the plume, set as indicated in equation 3.48 with sedimentation velocity considered to be the terminal velocity of particles (Chapter 2, Section 2.2.2). As terminal velocity depends on altitude, we take the average terminal velocity along the plume profile (Figure 3.4).

$\boldsymbol{V_e}$ is the exit velocity in the umbrella cloud, which is considered to be equivalent to the terminal velocity of particles (Chapter 2, Section 2.2.2).

Figure 3.4 shows an example of approximated geometry and sedimentation velocity of particles.

Parameters of the resulting model are summarized in Table 3.2. We make here a distinction between parameters that are defined by underlying models and parameters introduced by the



(a) (b)

Figure 3.4: Approximation of plume geometry with the plume slope with vertical $a = 0.2$ and average sedimentation velocities of particles used to solve the analytical SEER model. Sedimentation velocity is computed for each class from the vent height to the base of the umbrella cloud and then averaged.

Symbol	Unit	Definition
Parameter specific to the model		
z_0	m	Height of particles insertion above vent
Particles		
TGSD	%	Distribution and sizes of particles
ρ_ϕ	$kg \cdot m^{-3}$	Particles density
M	kg	Total erupted mass
Plume model parameters		
r_0	m	Radius at the vent
U_0	$m \cdot s^{-1}$	Velocity at vent
T_0	K	Temperature at vent
n_0		Initial gas mass fraction
z	m	Vent height

Table 3.2: Volcano-semianalytical-seer parameters. z_0 is considered to be specific to the model because it is not a parameter of another underlying model.

present model that we call specific to the model. For example, here z_0 is a parameter specific to the model, it defines at which height particles are considered to be inserted in the plume (or that particles cannot exit the plume below this height) and is not inherent to any underlying model.

3.2.8 Volcano-lagrangian-seer, particle-based computational formulation for a volcanic eruption

The analytical resolution of the SEER model gives a solution only for a strong plume geometry and constant velocities in the plume and umbrella cloud, constant exit velocities and a radius

that is a linear function of the height. We provide a numerical implementation that is able to work with arbitrary definition of those variables and couple it with a transport model in the atmosphere. The numerical SEER model is then considered as a source term of the atmospheric transport model which can account for diffusion, wind and sedimentation.

Multiscale formulation

The numerical transport model for the strong plume geometry is separated in three distinct transport processes : transport in the plume, transport in the gravity current and transport in the atmosphere. Transport in the plume and umbrella cloud for strong plume geometry is done according to SEER model, and transport in the atmosphere by advection-diffusion model. For the weak plume geometry, transport in the plume is done according to SEER model, while transport in the gravity current and atmosphere is done by advection-diffusion. Using SEER model for transport in the gravity current for a weak plume produces too narrow ground deposits (concentrated along the wind direction axis). It is thus necessary to rely on diffusion in that case to obtain a satisfactory correspondence of simulation results with field data. This hints that atmospheric turbulence plays an important role in the transport of particles inside the gravity current of a weak plume.

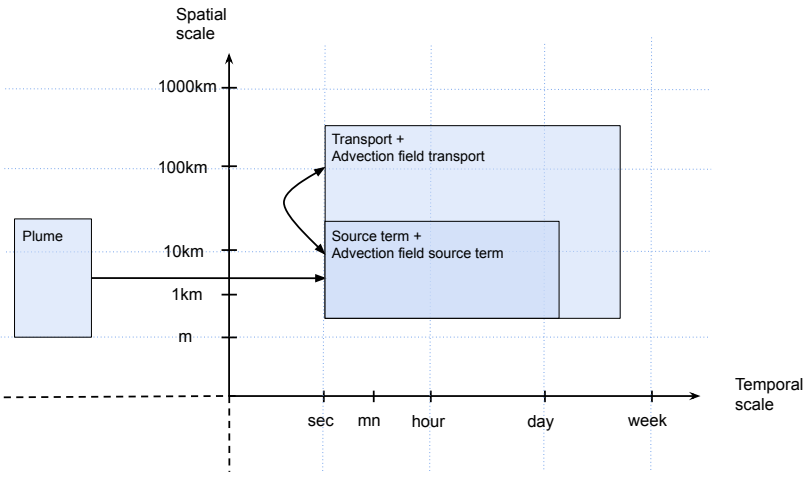
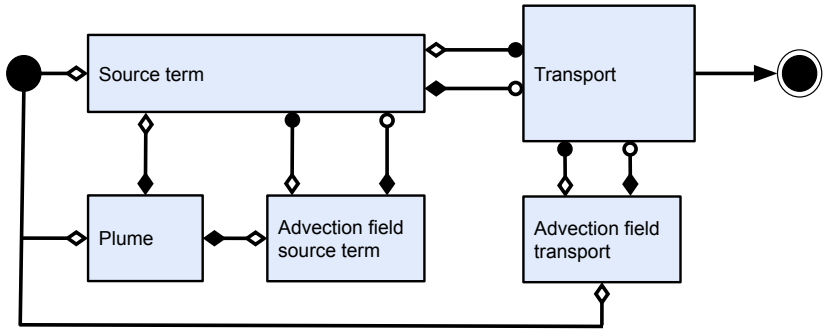
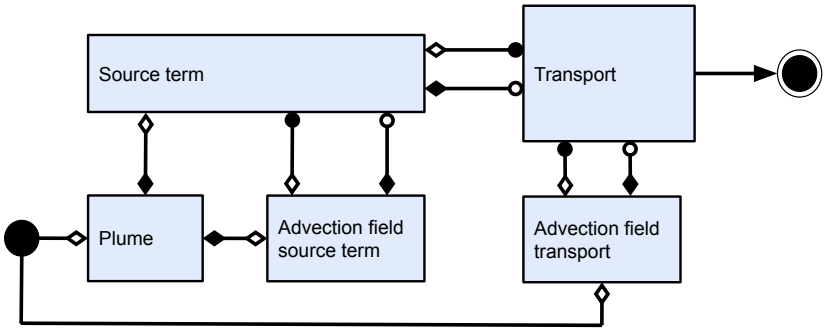


Figure 3.5: SSM of the TTDM Seer-lagrangian-tetras. *Plume* solves a steady state plume model, *source term* applies the SEER numerical model based on plume geometry and velocity field using *advection field source term*, *transport* apply Lagrangian transport in the atmosphere using *advection field transport*. The scale of *transport* overlaps with the scale of *advection field transport* and the scale of *source term* overlaps with the scale of *advection field source term*.

Figure 3.5 shows the SSM description of a TTDM based on the numerical SEER source term coupled with a plume model, a transport model and advection field models. We name this implementation volcano-lagrangian-seer. In this application, we identify the following sub-models :



(a)



(b)

Figure 3.6: gMML description of volcano-lagrangian-seer. (a) describes the case where total erupted mass is given by an independent parameter (used with Woods plume model for strong plumes). (b) describes the case where total erupted mass is derived from the plume model (used with Degruyter plume model for weak plumes).

Plume computes the steady state plume profile of Woods [123] or Degruyter and Bonadonna [49, 48].

Source term transport particles inside the plume and umbrella cloud and computes where and when particles are injected into the atmospheric transport domain.

Transport transport particles in the atmosphere according to an advection field.

Advection field source term computes the advection field of the source term based on plume velocity profile. Also include GC [100] velocity in the case of a strong plume.

Advection field transport computes the advection field for atmospheric transport based on the diffusion coefficient, terminal [3] and wind velocity [72]. Also include GC velocity in the case of a weak plume.

Figure 3.6 shows two variations of the gMML formulation of volcano-lagrangian-seer. As in

Chapter 2, there are two versions of the coupling, one where injected mass is an independent parameter (Figure 3.6a, used with Woods plume model for strong plumes) and the other where the injected mass is derived from the plume model (Figure 3.6b, used with Degruyter plume model for weak plumes). We identify the following coupling templates :

Plume* interaction with *source term is a dispatch template. *Plume* computes velocity and radius profile and send them to *source term* together with the MER if needed.

Advection field source term* interaction with *source term is a call-release template. *Source term* calls *advection field source term* at each iteration.

Plume* interaction with *advection field source term is a dispatch template where plume characteristics are computed and sent once to *advection field source term*.

Source term* interaction with *transport is a call-release template. *transport* manage particles, and use *Source term* at each iteration to move them while they are inside the plume or umbrella cloud.

Advection field transport* interaction with *transport is a call-release template. *Transport* calls *advection field transport* at each iteration to compute the advection field.

Computational formulation

Multiscale formulation gives the description of the model in terms of components called submodels. Computational formulation of SEER model for strong plumes and weak plumes are given in Algorithms 3.1 and 3.2 respectively. They differ from each other by the treatment they apply to particles reaching the top of the plume. Algorithms 3.3 and 3.4 shows advection-diffusion numerical models for strong plumes and weak plumes. They differ slightly from each other by the velocity they apply (there is no gravity current region in the strong plume advection-diffusion algorithm).

Then, two distinct algorithms are given, each making use of the previously described transport models. Algorithm 3.5 shows the procedure applied to simulate a volcanic eruption and correspond to the multiscale description displayed on Figures 3.5 and 3.6. Particles are injected at a given flow rate during a certain time and are simulated inside a domain of size $x \cdot y \cdot z$. They can deposit on a flat ground of size $x \cdot y$ (topography is not taken into account) divided in squares of size Δx^2. The terrain retains the deposited mass in $kg \cdot m^{-2}$ of each class of particles in each square of size Δx^2. The procedure is applied until no particles (or a small number of particles that has to be decided, for example less than 1% of the total number of injected particles) are left in the domain. This corresponds to the volcano-lagrangian-seer implementation used to simulate volcanic eruptions. Table 3.3 shows the list of parameters of this model.

Algorithm 3.1: Procedure *move particle seer strong plume* . This procedure takes a particle as input and dt is the time step.

```
if particle is inside plume then
    pick a random direction γ ∈ [0, 2π];
    pick a random number x ∈ [0, 1];
    if x ≤ p_φ(z) · dt then
        put the particle on the plume border in direction γ;
    else
        move the particle according to v_v(z);
    end
    if particle height > Z_h then
        pick a random direction γ ∈ [0, 2π];
        put the particle on the plume border in that direction;
    end
else if particle is inside umbrella cloud then
    pick a random number x ∈ [0, 1];
    if x <= P_φ(z) · dt then
        put the particle on the umbrella cloud base;
    else
        move the particle according to v_h(R);
    end
end
```

Algorithm 3.2: Procedure *move particle seer weak plume*. This procedure takes a particle as input. s is the position on the curvilinear coordinate generated by the plume centerline trajectory, dt is the time step. γ is chosen in $[0,2\pi]$ because it's the orientation of the vector normal to the plume border at the potential exit position of the particle that determines the exit probability (Figure 3.3).

```
pick a random direction γ ∈ [0, 2π];
pick a random number x ∈ [0, 1];
if x ≤ p_φ(s, γ) · dt then
    put the particle on the plume border in direction γ;
else
    move the particle according to v_v(s);
    if particle is outside plume then
        put the particle at a random uniform position on plume top;
    end
end
```

3.2.9 Simplified-lagrangian-seer, validation of the Lagrangian formulation

Analytical and numerical models are formulated in such a way that they should be in almost exact correspondence when the later one is considered in a steady state, with enough particles, v_e, V_e, u and U constant, $r(z) = az$, a unique particle class and no wind or diffusion in the

Algorithm 3.3: Procedure *move particles* of transport model for strong plume geometry.

```
for each particle do
    if particle is inside plume or umbrella cloud then
        call move particle seer strong plume;
    else
        v_r = random velocity;
        v_s(p) = sedimentation velocity;
        v_w(x) = wind velocity;
        move particle accoring to v_r + v_s + v_w;
        if particle reached the ground then
            deposit particle on the ground;
        end
    end
end
```

Algorithm 3.4: Procedure *move particles* of transport model for weak plume geometry.

```
for each particle do
    if particle is inside plume then
        call move particle seer weak plume;
    else
        v_r = random velocity;
        v_s(p) = sedimentation velocity;
        v_w(x) = wind velocity;
        if particle is in gravity current area then
            v_gc = gravity current velocity;
            move particle accoring to v_r + v_s + v_w + v_gc;
        else
            move particle accoring to v_r + v_s + v_w;
        end
        if particle reached the ground then
            deposit particle on the ground;
        end
    end
end
```

atmosphere.

Algorithm 3.6 is used for comparison with the analytical solution, only with the strong plume geometry model. This allows for a cross validation between the analytical resolution and the numerical implementation.

The analytical formulation gives a result in terms of rate of deposition of particles on the ground in a ring of width dR at a distance R of the vent. Thus, in order to validate the corre-

Algorithm 3.5: General volcano-lagrangian-seer algorithm. This correspond to the implementation used to simulate volcanic eruptions. n is the total number of particles per class to inject, d is the total duration of the eruption and dt is the time step.

```
1  n_i = n · 1/d/dt;
2  t = 0;
3  while t < d do
4  |   insert n_i particles of each class at vent position;
5  |   move particles;
6  |   t = t + dt;
7  end
8  while there are particles in the domain do
9  |   move particles;
10 |   t = t + dt;
11 end
```

Symbol	Unit	Definition
Parameter specific to the model		
D	$m^2 \cdot s^{-1}$	Diffusion coefficient in the atmosphere
Particles		
TGSD	%	Distribution and sizes of particles
ρ_ϕ	$kg \cdot m^{-3}$	Particles density
M	kg	Total erupted mass
Eruption and wind		
t	s	Eruption duration
W_{max}	$m \cdot s^{-1}$	Maximum wind speed
γ_w	rad	Wind orientation
Plume model		
r_0	m	Radius at the vent
U_0	$m \cdot s^{-1}$	Velocity at vent
T_0	K	Temperature at vent
n_0		Initial gas mass fraction
H_t	m	Plume height (optional)

Table 3.3: Volcano-lagrangian-seer parameters summary. D if considered specific to the model because it is not a parameter of an underlying model. Parameters not listed are considered constants in this work. H_t has to be set if any of r_0, U_0, n_0 or T_0 is not defined. In that case, a Monte Carlo inversion scheme is applied to find a compatible value for r_0, U_0, n_0 and T_0 (see Chapter 4).

spondence between the numerical and analytical formulations, one has to use a numerical model that works in a steady state and counts particles falling in rings on the ground instead of squares.This algorithm is applied without wind. First, it waits for the system to reach steady state (when particles reach the border of the domain), then run the simulation for a given amount of time in order to observe the flux of particles reaching the ground. We state here

Algorithm 3.6: Steady state model numerical model. n_s is the number of particles to inject per time unit, dt is the time step, t is the total measurment duration.

$n_i = n_s \cdot dt$
while *steady state is not reached* **do**
    inject n_i particles randomly distributed in a plume slice of thickness $v_v(z_0) \cdot dt$ centered on z_0;
    move particles;
end
clear $grounddeposit$
repeat t/dt **times**
    inject n_i particles randomly distributed in a plume slice of thickness $v_v(z_0) \cdot dt$ centered on z_0;
    move particles;
end

the changes made to Algorithm 3.5 to make it consistent with the analytical model and obtain Algorithm 3.6 :

- The ground is constructed as concentric rings of thickness ΔR centred on vent location.
- Particles are injected in a plume slice of thickness $u(z_0)\Delta t$ centred on z_0.
- Particles are considered to reach the ground by falling vertically and instantaneously after they exit the plume or umbrella cloud.
- The condition for reaching the steady state is when a particle reaches the border of the domain.
- Units are arbitrary.

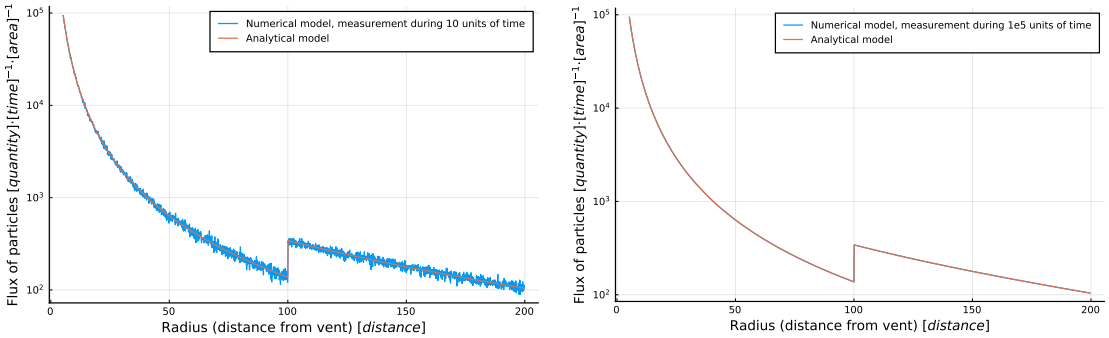
While the numerical model is stochastic by nature, its outcome cannot be exactly equal to the analytical one. Even with an arbitrarily fine space and time discretization and arbitrarily high number of simulated particles. The flux of particles reaching the ground as a function of the distance to the vent is a random variable, and the similarity of the outcome with the analytical expression should increase as we consider more samples (in that case, more particles).

Thus, we've designed a numerical experiment to ensure that simplified-lagrangian-seer outcome converges to the value of simplified-analytical-seer as we increase the number of simulated particles. For this, we run a simulation using the Algorithm 3.6 and the parameters summarized in Table 3.4. This procedure allows validating the submodel *source term* (Figures 3.5 and 3.6).

To monitor the evolution of the impact of the increase of the number of particles to the correspondence between numerical and analytical formulations, one needs a measure to quantify similarity or dissimilarity between the outputs.

Symbol	Definition	Value
Physical parameters		
z_0	height of particles insertion	10
ρ_{z_0}	particle density at z_0	1e6
z_h	height of the umbrella cloud base	200
h	thickness of the umbrella cloud	10
a	slope of the plume border	0.5
u	vertical speed	1
U	horizontal speed	1
v_e	exit velocity in the plume	0.05
V_e	exit velocity in the umbrella cloud	0.05
Numerical parameters		
ΔR	thickness of terrain rings	0.1
Δt	time step	0.1
n_s	numerical particles per unit of time	10^5
t	measurement time	$[10, 10^5]$

Table 3.4: Parameters used to test convergence of the numerical model. Units are arbitrary.



(a) (b)

Figure 3.7: Comparison between the outcome of simplified-lagrangian-seer and simplified-analytical-seer. In each case, 10^5 numerical particles are injected per unit of time. In 3.7a, we can see statistical fluctuations due to the randomness of the process. In 3.7b, those fluctuations cannot be seen anymore because the duration of the measure (thus the number of measured particles) is much higher.

To do so, we rely on the mean absolute percentage error (MAPE) which can be defined as follows

$$\text{MAPE} = \frac{100}{n} \sum_{i=1}^{n} \left| \frac{A_i - N_i}{A_i} \right| \tag{3.50}$$

where A_i is the result of the analytical expression and N_i is the outcome of the numerical model. The error is here expressed as a percentage of the reference value.

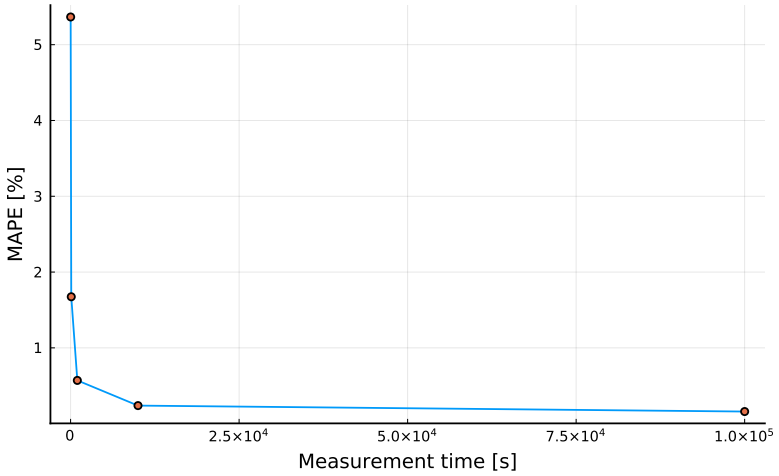


Figure 3.8: Convergence of the numerical model by increasing measurement time.

Results of those simulations are shown in Figure 3.7 and 3.8, which shows that the numerical model converges to a value that is close to the expected one (the smallest mean error is less than 0.2%). Which allows us to conclude that the analytical and numerical stochastic formulations are in agreement.

3.3 Simulation of past volcanic eruptions

In this section, we show simulation results of three past eruptions (Pululagua 2450BP, Ruapehu 1996 and Cotopaxi Layer 3). We show results from models volcano-lagrangian-seer, volcano-semianalytical-seer and Tephra2.

3.3.1 Tephra2

Tephra2 [39, 22] is a semi-analytical tephra transport model used both for the compilation of hazard maps and ESP inversion. Particles are considered to be released from a source term that is a line above the vent, with a given distribution of mass (here specified by a beta distribution). TGSD is approximated by a Gaussian distribution. Deposition of particles at given points on the ground is computed by solving the advection-diffusion equation. The atmosphere is layered, with each layer having a specific wind velocity.

Tephra2 offers an inversion procedure based on the downhill simplex optimization algorithm [89] (the algorithm is described in Chapter 4, Section 4.4.1). The inversion version of Tephra2 is parallelized using MPI. The parallelization is done on ground sites. While result of a ground site does not depend on the solution of any other location, the parallelization does not need communication between sites and the speedup is quasi linear with a small number of processes [38].

Symbol	Unit		Definition
Parameters specific to the model			
α		Inverted	α parameter of mass distribution (beta distribution)
β		Inverted	β parameter of mass distribution (beta distribution)
D	$m^2 \cdot s^{-1}$	Inverted	Diffusion coefficient (large particles)
FTT	$m \cdot s^{-1}$	Inverted	Fall time threshold
Particles			
M	kg	Inverted	Total erupted mass
max ϕ	ϕ		TGSD Maximum
min ϕ	ϕ		TGSD Minimum
μ_ϕ	ϕ	Inverted	TGSD Median
σ_ϕ	ϕ	Inverted	TGSD standard deviation
max ρ_ϕ	$kg \cdot m^{-3}$		Maximum density of particles
min ρ_ϕ	$kg \cdot m^{-3}$		Minimum density of particles
Plume and atmosphere			
H_t	m	Inverted	Plume height
W_{max}	$m \cdot s^{-1}$		Wind speed
γ_w	rad		Wind direction

Table 3.5: Tephra2 parameters summary. Parameters not listed are considered constants in this work. Parameters that are inverted in this work are marked as such. α, β, D and FTT are considered specific to the model because they cannot be approximated by another model. Tephra2 considers plume height above sea level. In this text, all considered plume heights are above vent.

In its inversion version, Tephra2 requires ranges of parameters to explore and output the best fit achieved with a set of field data given as input. It takes a seed value for the random number generator as well. Thus, each seed produce a specific process of optimization. The table 3.5 summarizes the parameters of Tephra2 and specifies the parameters being inverted.

Here, we consider that α, β, D and FTT are parameters introduced by Tephra2 because we don't have models to compute them.

3.3.2 Considered models and procedure

We compare simulation results for three different models : volcano-lagrangian-seer, volcano-semianalytical-seer and Tephra2. Tephra2 and SEER based models are very different in their underlying principles, making the comparison of those models difficult. It has then been chosen not to compare the models with the same set of parameters, as some of them does not have the same meaning in Tephra2 and in SEER based models (diffusion for example) or are found in a model but not in the other (like FTT in Tephra2 or U_0 in SEER based models).

Parameters of Tephra2 come from inversion with a range of values proposed in the literature. Parameters for volcano-lagrangian-seer and volcano-semianalytical-seer does not come

from an inversion procedure, but are taken from the literature, and manually adjusted when necessary to achieve a good fit (see Chapter 4 for preliminary results on inversion using volcano-semianalytical-seer).

Simulations with volcano-lagrangian-seer and Tephra2 have been made for the three eruption cases. Simulations with volcano-semianalytical-seer, on the other hand, have been performed only for the Pululagua eruption while this model is only able to reproduce strong plume eruption in no wind condition.

Tephra2 considers plume heights above sea level, while SEER based models consider plume heights above vent. We use the latter for plume heights in this text.

Results of Tephra2 are not given to show best achievable fit, but to give an idea of the kind of results the model produces. The same apply for SEER based models, where further investigations on inversion could surely improve fit between simulation and observed deposits.

Volcano-lagrangian-seer simulate the transport of individual particles and output the deposition in a matrix that represent the ground. Tephra2 produces deposition quantity for specified points on the ground. To produce outputs that can be visualized in the same way for both models, we pass to Tephra2 a list of points on the ground that are the same as the modelled terrain sites of volcano-lagrangian-seer. Volcano-semianalytical-seer output the deposition as a function of distance from vent.

As the deposit of Pululagua is considered almost axisymmetric, data are plotted on a side view as a function of distance from vent. Deposits of Cotopaxi and Ruapehu are shown on a top view as a heatmap.

3.3.3 Simulations results

$\Delta t = 1s$ for every volcano-lagrangian-seer simulation.

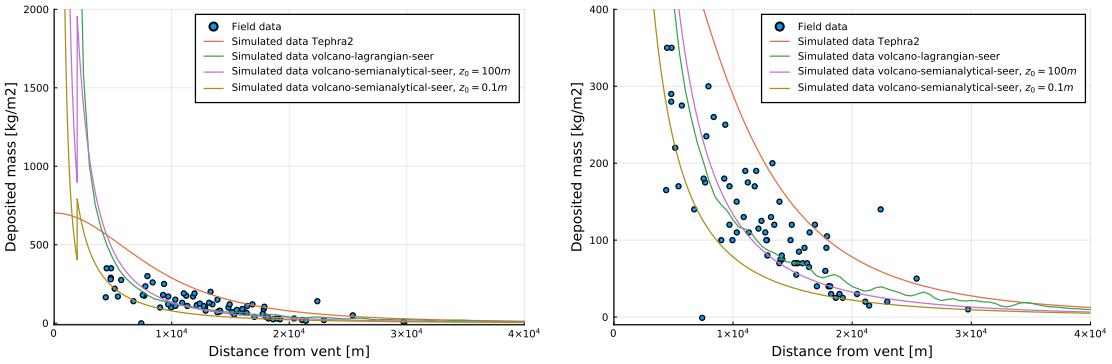
Pululagua 2450BP BF2

We refer the reader to Chapter 2 Section 2.5 for information on the Pululagua 2450BP eruption. We recall that this is a strong plume that occurred in almost no wind condition [117].

For volcano-lagrangian-seer, values of $U_0 = 124m \cdot s^{-1}$, $r_0 = 57m$ comes from work on inversion with statistical inference using Tetras (see Chapter 4). $n_0 = 0.01$ and $T_0 = 1256K$ comes as well from Chapter 4. It produces a plume height $H_t \approx 22km$. Diffusion in the atmosphere is the same as in Chapter 2 and TGSD is from [117], Figure 2.12 illustrates the TGSD used for the simulations of Pululagua. Density of particles is considered ranging from 600 to $2500kg \cdot m^{-3}$ [111]. Total erupted mass is set to $4.5 \times 10^{11} kg$. Vent is located at an altitude of approximately $2500m$.

Figure 3.4 shows the approximate geometry and sedimentation velocities considered for volcano-semianalytical-seer. z_0 has alternatively been set to $100m$ and to $0.1m$ to observe the effect on the deposition pattern.

For Tephra2 simulations, parameters are explored in the range $H_t \in [17.5, 33.5]km, D \in [10, 10000]m^2 \cdot s^{-1}, M \in [3,6] \times 10^{11}kg, \mu_\phi \in [-1,2], \sigma_\phi \in [1,3]$, FTT $\in [10, 7 \times 10^5]s, \alpha \in [0.1, 2.0], \beta \in [0.1, 2.0]$. The retained parameters after one inversion run are $H_t \approx 33.5km, D = 10000m^2 \cdot s^{-1}, M \approx 6 \times 10^{11}kg, \mu_\phi \approx 2, \sigma_\phi \approx 3$, FTT $= 457573s, \alpha \approx 2, \beta \approx 0.1$. It is important to note that those results should be investigated furthermore and compared with [117] because some values correspond to extremum values of the exploration range.



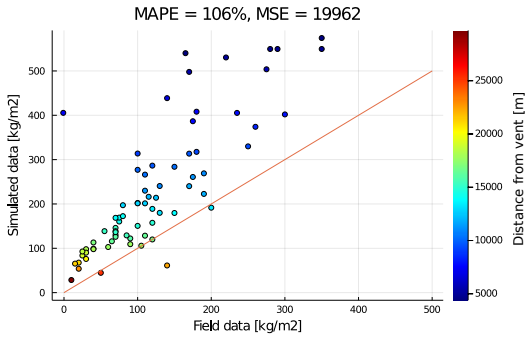
(a) (b)

Figure 3.9: Comparison between field data and simulations for Pululagua 2450BP eruption. (a) and (b) show the same data with different y-axis scales.

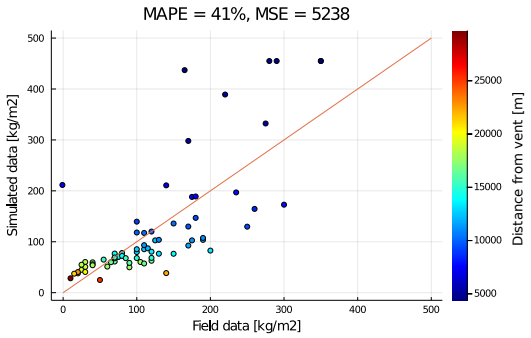
Results of simulations with the different models are shown in Figure 3.9. We can see that reproduction of proximal points is particularly difficult for a model like Tephra2 in such an eruption occurring in no wind condition. Further from the vent, all models seems to converge. The value of z_0 has an important impact on the results of volcano-semianalytical-seer. Informally, this parameter represents the minimum height of the exit of the particles. We see that when $z_0 = 100m$, outputs from volcano-semianalytical-seer and volcano-lagrangian-seer are much closer. This can be due to the choice of $\Delta t = 1s$ for volcano-lagrangian-seer simulations. While $U_0 \approx 100m \cdot s^{-1}$, there is a gap of approximately $100m$ where particles cannot exit with $\Delta t = 1s$. We could hypothesize that particles cannot exit the plume in the jet phase, but more investigation would be needed. Moreover, volcano-semianalytical-seer consider radius at the vent to be 0, while this is not the case in volcano-lagrangian-seer which considers $r_0 > 0$. This is another source of discrepancy.

To compare field measures with simulation results, we use the MAPE error as in section 3.2.9 and the mean squared error MSE. One outlier point that has a value of almost 0 has been removed for the computation of MAPE. The result is shown in Figure 3.10. We see that SEER based models outperforms Tephra2 in that case. We can see that choice of the error measure is of great importance, for example volcano-lagrangian-seer has a better MSE (Figure 3.10b) than volcano-semianalytical-seer with $z_0 = 100m$ (Figure 3.10d), while it's the other way around

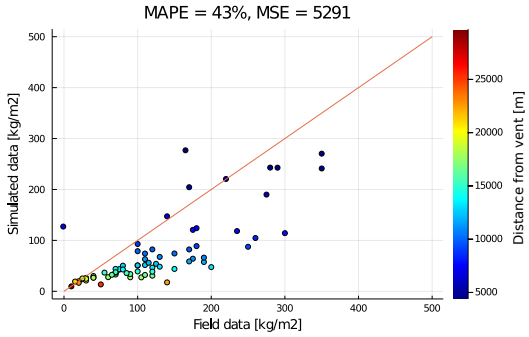
for MAPE.



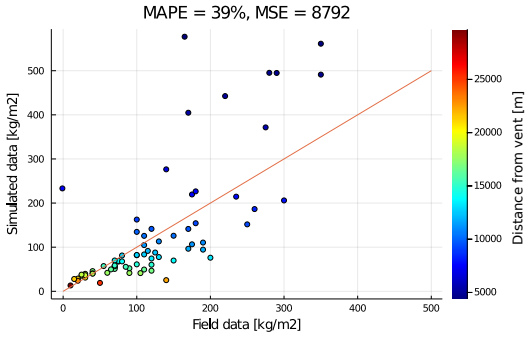
(a) Mass per area comparison for Tephra2.



(b) Mass per area comparison for volcano-lagrangian-seer.



(c) Mass per area comparison for volcano-semianalytical-seer when $z_0 = 0.1m$.



(d) Mass per area comparison for volcano-semianalytical-seer when $z_0 = 100m$.

Figure 3.10: Field data compared to simulated data for Pululagua eruption. In the case of a perfect match between simulation and field data, dots would be on the string line $y = x$. Dots are colorized according to the distance to the vent of the considered point. Blue dots corresponds to proximal points while red dots to distal points.

Cotopaxi layer 3

Cotopaxi is a volcano located in Ecuador, close to Quito like Pululagua, but south of the city. The considered deposition layer 3 is thought to result from an eruption that occurred approximately 900 years ago in a moderate wind condition, with the deposit elongated in the north-west direction [17, 5]. Vent is located at an altitude of approximately $5700m$.

For simplicity, wind is considered to be known with a maximum speed of $28m \cdot s^{-1}$, blowing in the direction 280.7° ([111], angle from the north with anti-trigonometric rotation). Density of particles ranges from $600kg \cdot m^{-3}$ to $2600kg \cdot m^{-3}$ and TGSD is obtained by Voronoi tessellation [111]. [17] investigates ESP and propose a total erupted mass of $1.7 \times 10^{12} kg$ and a plume height of $29km$.

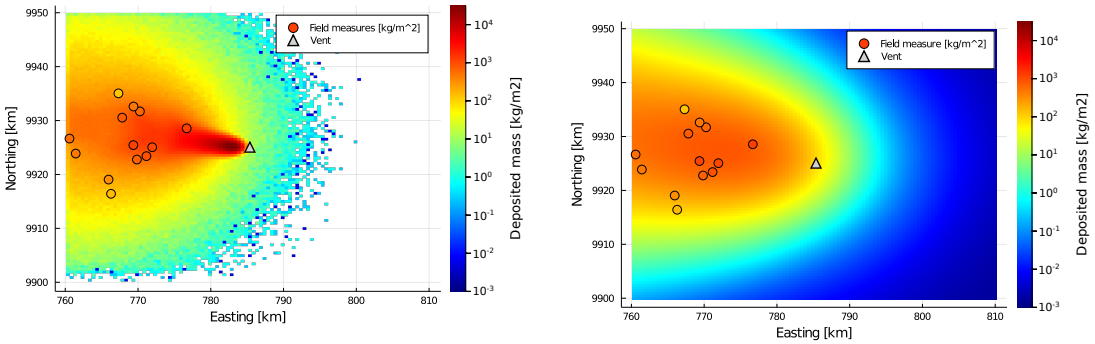
For Tephra2, total mass has been set to $1.7 \times 10^{12} kg$ according to [17] and plume height

empirically set to $23km$. $\alpha = 0.1, \beta = 2, \mu_\phi = 2, \sigma_\phi = 3$ have been defined by inversion, as well as FTT to $44897s$. D has been manually set to $10000m^2 \cdot s^{-1}$ while a high coefficient of diffusion seemed to improve results.

For volcano-lagrangian-seer, rather than setting values for U_0, r_0, T_0 and n_0 as for Pululagua, we set a value of H_t and the model finds a compatible set of plume parameters by random sampling. H_t has been set to $23km$ as for Tephra2. The corresponding parameters obtained are $U_0 = 230m \cdot s^{-1}, r_0 = 65m, n_0 \approx 0.016, T_0 = 1208$ for a plume height of $23062m$. Total injected mass has been empirically set to $8 \times 10^{11} kg$ to improve fitting (approximately twice less than suggested in [17]). Diffusion in the atmosphere is set to $300m^2 \cdot s^{-1}$, also empirically.

Results are shown in Figure 3.11 as a heatmap showing ground deposition compared to field data in log scale. In Figure 3.11a, empty areas correspond to position where no particle reached the ground. The deposition pattern again shows a very high deposit close to the vent for volcano-lagrangian-seer, while the deposit from Tephra2 is smoother.

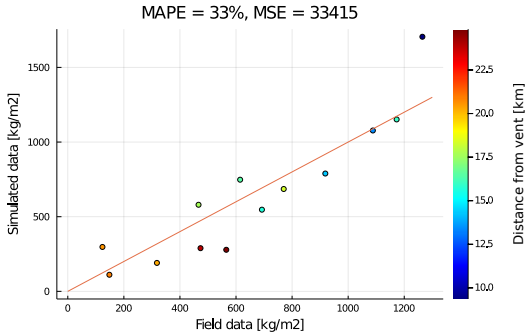
Again, comparison between mass per surface area from the field and from simulations is compared in Figure 3.12.



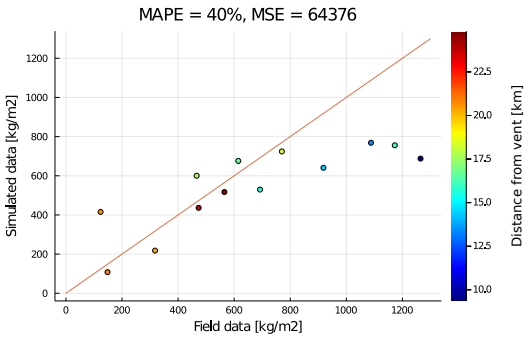
(a) Mass per area comparison for volcano-lagrangian-seer.

(b) Mass per area comparison for Tephra2.

Figure 3.11: Deposit on the ground computed with (a) volcano-lagrangian-seer and (b) Tephra2 compared to field data for Cotopaxi layer 3. Values of deposit are in $kg \cdot m^{-2}$.



(a) Mass per area comparison for volcano-lagrangian-seer.



(b) Mass per area comparison for Tephra2.

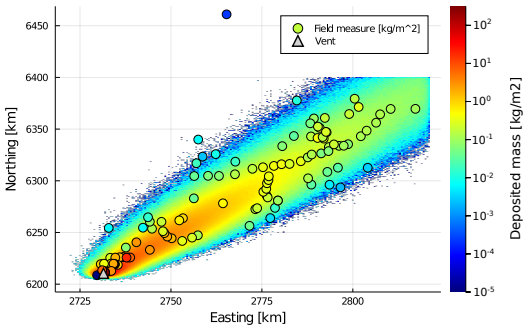
Figure 3.12: Comparison of mass per area between data from the field and from simulations for (a) volcano-lagrangian-seer and (b) Tephra2 for Cotopaxi layer 3. Dots are colorized according to the distance to the vent of the considered point. Blue dots corresponds to proximal points while red dots to distal points.

Ruapehu 1996

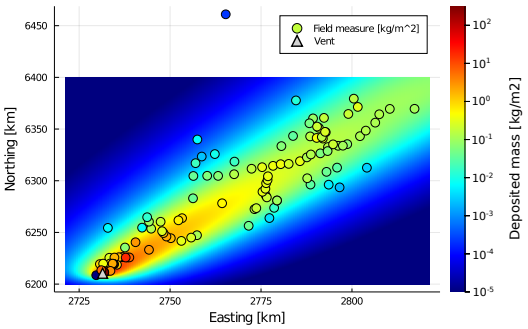
We refer the reader to Chapter 2, Section 2.5 for details on Ruapehu 1966 eruption. To model the eruption, we considered a wind field of $30m \cdot s^{-1}$ at the tropopause. For Tephra2, the following range of parameters has been considered for inversion : $H_t \in [3,13]km, M \in [3 \times 10^9, 2 \times 10^{10}]kg$, FTT $\in [10, 7 \times 10^5]s, \alpha \in [0.1,2], \beta \in [0.1,2], \mu_\phi \in [-1,2], \sigma_\phi \in [1,3]$. Parameters have been set to $H_t \approx 13km, \alpha \approx 2, \beta \approx 1.13, M \approx 4 \times 10^9 kg, \mu_\phi \approx 2, \sigma_\phi \approx 3, D = 1962m^2 \cdot s^{-1}$, FTT $\approx 292000s$. Here again, we note that results of some parameters correspond to extrema of the inversion range, which indicates that more investigations are needed.

For volcano-lagrangian-seer, Degruyter plume model parameters have been selected as follows : $\alpha = 0.1, \beta = 0.5, U_0 = 40m \cdot s^{-1}, T_0 = 1273K, n_0 = 0.03, r_0 = 40m$. Those parameters produce a plume of $H_t \approx 5km$ with a mass eruption rate of $\approx 3.56 \times 10^5 kg \cdot s^{-1}$. This produces a total injected mass $M \approx 8.3 \times 10^9 kg$.

Simulated points that have a difference larger than 10^5% with field data are considered outliers and are not considered for error computation. Figure 3.14 shows the comparison between deposition from volcano-lagrangian-seer and from Tephra2. Mass deposited from Ruapehu 1996 eruption are way smaller than depositions from Pululagua and Cotopaxi, this can disadvantage percentage error measures because a small absolute error can produce a high percentage error. Figures 3.14a and 3.14b show a comparison between measured and simulated deposition on the ground in log scale. In case of a perfect match between simulation and field data, points would be on a line. Dashed lines show the value of a perfect match times 5 and perfect match divided by 5. We can consider that simulated values lying in this range are of the correct order of magnitude. Here again it is interesting to note that both models are considered better depending on the error considered, which show the importance of the choice of the error

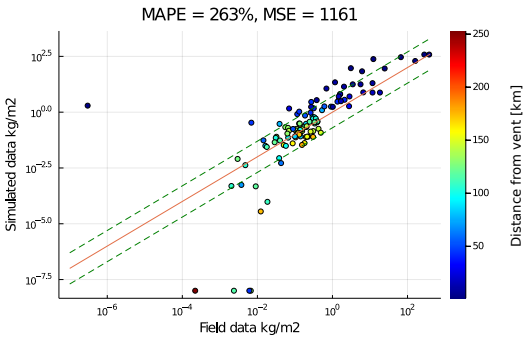


(a) Mass per area comparison for volcano-lagrangian-seer.

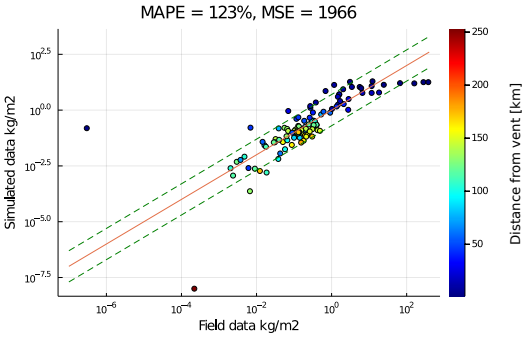


(b) Mass per area comparison for Tephra2.

Figure 3.13: Deposition on the ground computed with (a) volcano-lagrangian-seer and (b) Tephra2 compared to field data for Ruapehu 1996 eruption. Values of deposition are in $kg \cdot m^{-2}$.



(a) Mass per area comparison for volcano-lagrangian-seer.



(b) Mass per area comparison for Tephra2.

Figure 3.14: Comparison of mass per area between data from the field and from simulations for (a) volcano-lagrangian-seer and (b) Tephra2 for Ruapehu 1996 eruption. Dots are colorized according to the distance to the vent of the considered point. Blue dots corresponds to proximal points while red dots to distal points.

measure. Absolute error is smaller with the volcano-lagrangian-seer, while the percentage of error is in average lower with Tephra2.

3.4 Parallelization and performances

A use case of SEER based models is to couple them with optimization algorithms to do inversion. This kind of task requires to solve hundreds or thousands of instances of a model. This is easily doable for volcano-semianalytical-seer, but is a difficult problem for the volcano-lagrangian-seer numerical implementation due to its heavier computational load. We are

thus interested in the runtime of simulations. Some optimization algorithms allow running multiple instances of the model at the same time, such as statistical inference algorithms [55], while others require to run instances sequentially, such as the downhill simplex [89].

The property of a system to handle the growth of available resources is called scaling. In the context of a parallel programming, scaling denotes the capacity of a program to use an increasing number of cores. In the domain of HPC, two types of scaling are mainly considered : strong scaling and weak scaling. Strong scaling denotes the capacity of a program to use more cores while the total work remains constant (i.e., make the program run faster for a given problem size), while weak scaling denotes the capacity of a program to use more cores while the total work becomes larger (i.e., threat larger problems with a constant time, or simply being able to treat larger problems due to memory occupancy).

In this work, we are particularly interested in the strong scaling of the program, while the goal is to solve the model as fast as possible to integrate it in an optimization algorithm.

Volcano-lagrangian-seer is implemented in C++ and parallelized using MPI. While there is no interaction between particles, the parallelization is done on particles. At each iteration, n_i particles of each class are injected while the duration of the eruption is not reached as indicated in Algorithm 3.5. When running on p processor cores, each core will inject n_i/p particles of each class at each iteration. If this number is not an integer, the remainder $n_i \pmod p$ of particles will be inserted uniformly on $n_i \pmod p$ cores.

The solving of Woods model is embedded in the main C++ code and is integrated with a simple inversion strategy by random sampling which allows to find a set of compatible parameters U_0, r_0, n_0, T_0 with a given plume height H_t (see Chapter 4, section 4.2). It is also possible to not run this inversion by specifying U_0, r_0, n_0, T_0 parameters.

Degruyter plume model is solved by a separated Matlab code and its output written to a file which is then read by the C++ code. This plume model is not integrated with an inversion strategy at the moment.

3.4.1 Performance analysis

Computationally, a simulation involves three main steps :

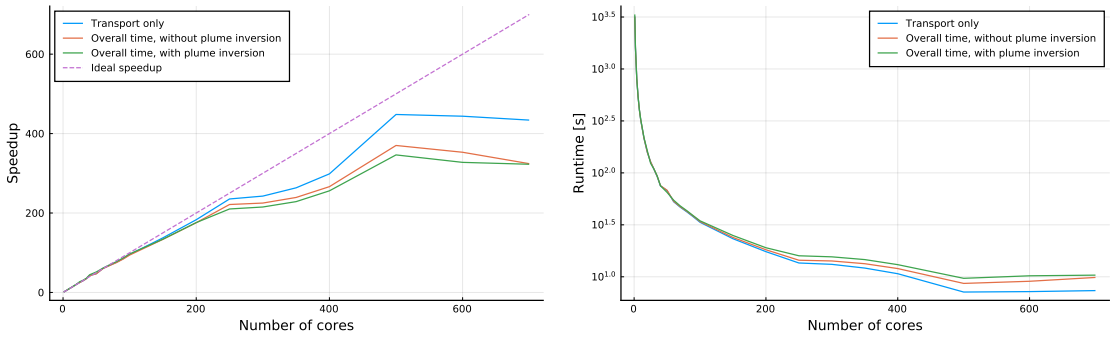
- Inversion or solving of the steady state plume model;
- Transport and deposition of particles;
- Gathering and writing of data.

Performance of the model is strongly dependent on numerical parameters (number of particles, the size of the domain), but also on physical parameters (wind speed, diffusion coefficient). We give performance analysis for the three studied eruption cases.

Model is evaluated in condition suited for inversion. A critical parameter to achieve reliable results is the number of particles used for the simulation. We choose to run all simulations with about 2×10^6 particles, which is enough to produce stable simulations (see Appendix A). Each simulation configuration is run 5 times and we take the mean value of those 5 runs as runtime. Simulations are run on the *Yggdrasil* cluster of the Geneva's higher education institutions on nodes equipped with Intel Xeon Gold 6240 CPU @ 2.60GHz processors.

Pululagua

A typical sequential run for Pululagua takes about $3400s$ to terminate, which is approximately $57min$. Solving of Woods plume model is fast as it usually takes between 5 and $8ms$. When inversion on plume model is used, it can take up to p to 300 random samples ($\approx 2s$) to find a correct set of parameters. This is negligible when running on one core, but while this part is not parallelized, it can have an impact on speedup if the model is run on a large number of cores as suggested by Amdahl's law [2].



(a) Speedup obtained for Pululagua simulation. (b) Execution time for Pululagua simulation.

Figure 3.15: Speedup (a) and execution time (b) for Pululagua Simulation. Measurements are done with 1.8×10^6 particles injected over $200s$ in a domain of size $50km \times 50km$.

The final result produced by the model is a matrix of sites on the ground, where each element of the matrix stores the number of particles for each class deposited on the ground at the corresponding site. Thus, at the end of the computation, each process stores a partial result that has to be gathered on the main process. This step is done by a reduction operation. However, the internal data structure used (array of structures instead of structure of arrays) is not well suited to perform this operation, while one reduction per matrix element and per particle class is necessary. The impact is low when using a small number of cores, but becomes problematic when this number is increased. This problem could be alleviated by changing the data structure used for reduction and doing the reduction only on corresponding field points which are necessary for inversion.

Figure 3.15 shows the runtime and speedup obtained for Pululagua simulation. We consider first the runtime of transport only. The speedup is almost linear up to 250 cores. The only

overhead coming from synchronization and communication between cores to test the termination of the simulation. Beyond 250 cores, we observe an inflexion in the speedup, that could be due to a slight load balancing problem, but performances remain good. Then, from 500 cores, we observe a stagnation of the speedup. This is due to the way particles are distributed on computing cores in that case. 1.8×10^6 particles are injected in total over $200s$ with $dt = 1s$. This means that $1.8 \times 10^6/200 = 9000$ particles per iteration are injected. When using 500 cores, which means that each core inject $9000/500 = 18$ particles per iteration. In the current implementation, each core inject particles of every class. While in this simulation we use 18 particle classes, that means that we cannot divide any more the number of particles injected per iteration. This particle injection strategy could be modified to improve further the strong scaling of the application.

Then, if we look at the speedup obtained with the overall time (including reduction) but without plume parameters inversion, we see that the overhead induced by reduction becomes significant with more than 250 cores, but the speedup remains reasonable up to 500 cores. If we look at the plume parameters inversion impact on speedup, we see that it's not significant up to 500 cores.

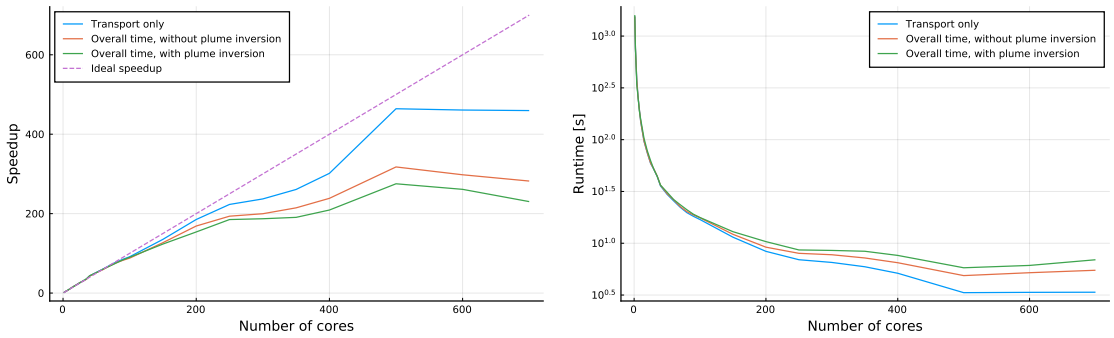
Finally, by using 500 cores for this simulation, which is the maximum useful number of cores for this implementation and this simulation, each simulation takes $\approx 8.7s$ without plume parameters inversion, which would allow running more than 400 instances per hour. Each simulation takes $\approx 9.7s$ with plume parameters inversion, which would allow running more than 370 instances per hour.

Cotopaxi

Figure 3.16 shows the runtime of Cotopaxi simulations. The runtime on one core of a Cotopaxi simulation is $\approx 1600s$. This is less than a Pululagua simulation, partly because there are slightly less particles, but the main reason is the presence of wind, that carry particles (1.6×10^6 instead of 1.8×10^6) out of the domain faster.

The same conclusions can be drawn for Cotopaxi than for Pululagua, except that the final reduction has a greater impact on performances. Which makes sense, because the transport part of the computation is faster and the size of the data to reduce remains the same.

If we use 500 cores for this simulation, the runtime for one instance is $\approx 4.9s$ without using plume parameters inversion, which allows to run more than 700 instances per hour. With plume parameter injection, the runtime is $\approx 5.8s$, which allows to run more than 600 instances per hour.



(a) Speedup obtained for Cotopaxi simulation.

(b) Execution time for Cotopaxi simulation.

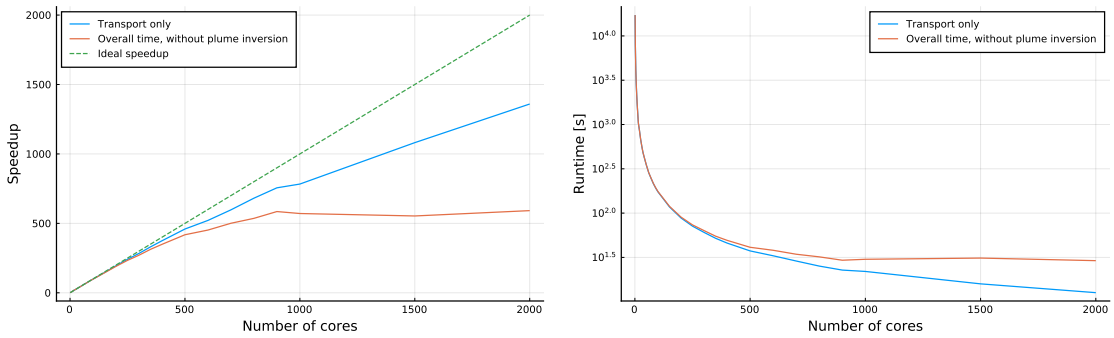
Figure 3.16: Speedup (a) and execution time (b) for Cotopaxi Simulation. Measurements are done with 1.6×10^6 particles injected over $200s$ in a domain of size $50km \times 50km$.

Ruapehu

Degruyter plume model used for weak plumes such as Ruapehu is implemented in Matlab. Runtime loading of Matlab takes about $7s$ on *Yggdrasil* and solving of plume model itself about $0.4s$, which is approximately 100 times more than Woods model implemented in C++. Inversion of plume parameters is not implemented for weak plumes as it would require a more optimized implementation. Thus, we consider here only the runtime of transport model and reduction.

Runtime for one sequential simulation of Ruapehu takes $\approx 17200s$, which is almost $5h$. Figure 3.17 shows the evolution of speedup of the model. We see that the transport part is not affected by a limit on the speedup, while it increases up to 2000 cores. However, the impact of reduction makes useless to use more than 900 cores with the current implementation.

With the current implementation, running the model on 500 cores takes $\approx 41s$, which would allow running almost 90 instances per hour. On 900 cores, it takes $\approx 29s$, for more than 120 instances per hour. With an optimization of the reduction step, this could drop under $15s$ per run on 2000 cores, which would mean 240 runs per hour.



(a) Speedup obtained for Ruapehu simulation.

(b) Execution time for Ruapehu simulation.

Figure 3.17: Speedup (a) and execution time (b) for Ruapehu Simulation. Measurements are done with 2.3×10^6 particles injected over $200s$ in a domain of size $100km \times 200km$.

3.5 Conclusion

We propose a family of models named SEER (Spatially Extended Exit Rate) that describes volcanic particles movement from the moment they leave the vent until they reach the ground by a set of mechanistic processes. We propose four SEER based models : simplified-analytical-seer, volcano-semianalytical-seer, simplified-lagrangian-seer and volcano-lagrangian-seer.

Simplified-analytical-seer is an analytical solution for a simplified plume and umbrella cloud geometry and constant velocities. It allows for a fast solving and a validation of the numerical SEER based implementations. Volcano-semianalytical-seer is a coupling of volcanological models with simplified-analytical-seer. It allows for a fast approximation of deposition for strong plume in steady atmosphere. Simplified-lagrangian-seer is a numerical particle-based model that mimics simplified-analytical-seer. It's numerical model is validated using simplified-analytical-seer and is used in volcano-lagrangian-seer. Volcano-lagrangian-seer is a numerical particle-based model coupled with volcanological models and a transport model in the atmosphere. It allows for simulations of eruptions of any type, in any atmospheric condition.

In SEER models, a process describes how particles leave the plume or umbrella cloud region. In volcano-semianalytical-seer and volcano-lagrangian-seer, this process is considered to be quantified by settling velocity of particles and geometry of the plume. This avoids introduction of new parameters. The only parameter introduced by volcano-lagrangian-seer is the diffusion coefficient in the atmosphere.

Simplified-analytical-seer allows us to highlight that it is not obvious that sedimentation regimes from plume and umbrella regions are continuous. The question remains open whether this is an artefact of the model or an actual observable phenomenon. It could be that turbulent diffusion in the atmosphere prevents from observing on the ground a discontinuity in

sedimentation rates that exists at the junction between plume and umbrella cloud.

Simulations results for three past eruptions of different types are shown. Pululagua 2450BP which is a strong plume without wind, Cotopaxi layer 3 which is a strong plume with wind and Ruapehu 1996 which is a bent-over plume.

Pululagua 2450BP allows comparing results from volcano-semianalytical-seer and volcano-lagrangian-seer. They cannot be in exact correspondence while volcano-semianalytical-seer makes simplifications that are not made in volcano-lagrangian-seer. We see that for specific value of z_0 parameters of volcano-semianalytical-seer, results of each model are close. z_0 parameter has an important impact on the behaviour of volcano-semianalytical-seer, which indicates that sedimentation close to the vent is of high importance with SEER based models and would deserve more investigations.

Volcano-lagrangian-seer is parallelized on a distributed memory architecture using MPI. The strong scaling behaviour allows solving the model in less than a minute if enough computing power is available. This allows using this model in an inversion context where the solving of hundreds of instances is necessary. It would also be possible to use volcano-semianalytical-seer for such a task, or as fast first exploration of the parameter space.

The main advantages of SEER based models are their modularity, their mechanistic nature and their accounting for arbitrary plume geometry. They also do not rely on unrealistically high diffusion coefficients. Other strategies could be used to solve SEER based models. Such as numerically computing flux of particles on the plume and umbrella cloud borders using a finite difference scheme.

4 Toward new strategies for inversion of eruption source parameters using HPC

The Approximate Bayesian Computation part of this chapter has been published in

L. Pacchiardi, P. Künzli, M. Schöngens, B. Chopard, and R. Dutta. "Distance-learning For Approximate Bayesian Computation To Model a Volcanic Eruption". In: *Sankhya B* (Jan. 2020). DOI: 10.1007/s13571-019-00208-8

and

R. Dutta, M. Schoengens, L. Pacchiardi, A. Ummadisingu, N. Widmer, P. Künzli, J.-P. Onnela, and A. Mira. "ABCpy: A High-Performance Computing Perspective to Approximate Bayesian Computation". In: *arXiv:1711.04694 [stat]* (Feb. 25, 2021).

The text is adapted to focus on the integration of expensive MPI models in an ABC inference framework. The rest is original unpublished work.

4.1 Introduction

Inversion is the task of optimizing input parameters of a model (called the forward model) to obtain best fit of the model output with field measurements. For a volcanic eruption, this allows estimating eruptions source parameters (ESP) such as plume height, initial plume speed, radius and temperature, total erupted mass, eruption duration, particle characteristics, or even atmospheric conditions when the eruption occurred before meteorological surveys existed.

Inversion is generally done through heuristics or statistical inference algorithms. Those techniques work by running a lot of instances of the forward model (usually thousands) and optimizing input parameters by exploration of the parameter space. For heavy numerical simulations, this imposes that the forward model is parallelized or that the optimization algorithm itself is parallelized by allowing running multiple instances of the forward model at

the same time.

This chapter is divided in three parts. In the first part, we describe a simple Monte Carlo inversion strategy that allows retrieving plume model parameters given a plume height. Note that this technique output only one set of compatible parameters, while multiple sets of parameters can exist that generates a given plume height.

In the second part, we describe the integration of Tetras (see Chapter 2), which is a mildly expensive C++ distributed memory parallel model, into the statistical inference Python framework ABCpy [54, 55]. ABCpy is an open source framework for Approximate Bayesian Computation (ABC) which implements algorithms that allows inferring parameters of a model given field observations. This framework is parallelized in a distributed memory architecture either using Spark or MPI as backend. Tetras being itself parallelized using MPI, it is necessary to develop a nested parallelized communicator structure to handle this model with ABCpy.

In the third part, we describe the integration of the SEER based model volcano-semianalytical-seer (see Chapter 3) to the Nelder-Mead optimization algorithm [89] through the Python package SciPy [116]. This allows us to show that the parameter space of the model is manageable by such an optimization algorithm. This opens the way for more investigations on ESP inversion with SEER based model, with statistical inference or optimization algorithms and with volcano-lagrangian-seer.

Various algorithmic approaches exist for inversion of eruption source parameters. Tephra2 software package includes an inversion procedure based on the Nelder-Mead algorithm [38]. In that case, Tephra2 forward model is parallelized, but the optimization procedure itself is sequential. [119] propose an integration of Tephra2 to the Levenburg-Marquardt algorithm and [124] with the Metropolis-Hastings algorithm.

The two later options have the advantage of allowing for uncertainty quantification. However, those works all rely on the Tephra2 dispersal model, which does not account for a first principles plume model, and is thus not able to reproduce in a satisfactory way proximal deposit, and does not account for position dependent wind field, which impacts greatly long-range deposits. Moreover, all the aforementioned works make use of a sequential optimization algorithm, which prevents benefiting from modern clusters and supercomputers, routinely available in universities and research centres.

In this chapter, we propose a perspective of using high-performance computing to perform inversion with parallel statistical inference algorithms and parallel MPI forward model. We use the ABCpy approximate Bayesian computation(ABC) framework written in Python and the Tetras model. ABCpy embed a great variety of ABC algorithms and is parallelized using an Apache Spark or a MPI backend. In order to integrate Tetras, which is a parallel distributed memory architecture MPI based parallel model, to this framework, we extend ABCpy with a nested parallelization architecture. Nested parallelization makes possible to differentiate the MPI communicator of the inference algorithm and the communicator of the forward model.

We investigate as well the possibility to perform inversion with volcano-lagrangian-seer and volcano-semianalytical-seer forward models. For this, we couple volcano-semianalytical-seer with an optimization function of SciPy and perform inversion with the Nelder-Mead downhill simplex algorithm. This allows confirming the feasibility of inversion using the parallel model volcano-lagrangian-seer if an HPC infrastructure is available, or a joint optimization procedure based on volcano-semianalytical-seer and volcano-lagrangian-seer.

4.2 Eruption source parameters

Eruptions source parameters (ESP) are the parameters describing the initial condition of a volcanic eruption model. They can differ between models, but they usually include :

- plume height,
- eruption duration,
- mass eruption rate (MER) or total erupted mass or volume,
- particle characteristics and distribution.

The ability to quantify those parameters is of great importance for reconstructing past eruption events or to study the impact of future eruptions. Often, the goal is to characterize those parameters from tephra deposited on the ground after the eruption, which is sort of a footprint of the eruption.

Some models, such as Tephra2 [38], take directly the plume height as a parameter of the model. Others, such as the considered models in this work Tetras (Chapter 2) and SEER based models (Chapter 3), rely on a plume model. In that case, the source parameters of the dispersion model includes the parameters of the plume model, including

- initial velocity of the plume,
- radius of the plume at the vent,
- temperature of the plume at the vent,
- gas mass fraction of exoslved volatiles at the vent.

Plume height and mass eruption rate can then be derived from the forward plume model. However in some situations, it is more convenient to consider the plume height as a parameter, rather than an output of the model. For example, when trying to compare the model with plume heights given in the literature, or when observing an eruption in order to do real time forecasting. In the latter case, it is possible to estimates plume height from sensors such as radar and cameras [53], while directly observing in real time plume source parameters such as initial velocity and radius is much more challenging. A forward model like Tephra2 can then

be directly used to invert plume height, while a supplementary step is necessary if we consider Tetras or a SEER based model.

4.2.1 A simple Monte Carlo strategy to invert plume source parameters

We present here the simple strategy adopted to retrieve a set of compatible values for parameters of a plume model with a given plume height. This is the strategy used in Chapter 2 and 3 when using H_t (the plume height) as an input parameter of the model. This is simple and efficient and allows for a very accessible introduction to the principle of inversion.

We consider the plume model of Woods. We recall that this model describes the characteristics of a vertical volcanic plume [123]. Its main parameters are the initial velocity of the plume U_0, the radius of the plume at the vent r_0, the temperature of the plume at the vent T_0 and the gas fraction of exolved volatiles n_0. The model outputs a velocity and radius profile from the vent to the top of the plume. Plume height H_t is the height at which the vertical velocity becomes 0.

Algorithm 4.1: Woods plume model Monte Carlo inversion algorithm. f is the function that computes plume height from Woods plume model given a set of plume parameters.

Input: The desired plume height H_t, the height tolerance ϵ, the maximum number of iterations i_{max}, the research intervals of the plume parameters $U_0^{min}, U_0^{max}, r_0^{min}, r_0^{max}, T_0^{min}, T_0^{max}, n_0^{min}, n_0^{max}$

Output: Plume models parameters U_0, r_0, T_0, n_0

$U_0 \leftarrow U(U_0^{min}, U_0^{max})$
$r_0 \leftarrow U(r_0^{min}, r_0^{max})$
$T_0 \leftarrow U(T_0^{min}, T_0^{max})$
$n_0 \leftarrow U(n_0^{min}, n_0^{max})$
while $\left| f(U_0, r_0, T_0, n_0) - H_t \right| > \epsilon$ **do**
    $U_0 \leftarrow U(U_0^{min}, U_0^{max})$
    $r_0 \leftarrow U(r_0^{min}, r_0^{max})$
    $T_0 \leftarrow U(T_0^{min}, T_0^{max})$
    $n_0 \leftarrow U(n_0^{min}, n_0^{max})$
    if *Number of iterations* $> i_{max}$ **then**
        return *No valid configuration found*
    end
end
return U_0, r_0, T_0, n_0

To find a set of parameters compatible with a given plume height we apply a naive Monte Carlo sampling strategy shown on Algorithm 4.1. In our implementation, ϵ is set to $100m$, $i_{max} = 100000, U_0 \in [10,400]\, m \cdot s^{-1}, L_0 \in [20,200]\, m, n_0 \in [0.01, 0.05], T_0 \in [200,1300]\, K$. Woods plume model is implemented in C++ and is solved in $5-8ms$ on an Intel Xeon Gold 6240 CPU @ 2.60 GHz. The maximum observed number of samples when a valid configuration exists does not exceed 300, which produce an inversion procedure of $\approx 2s$ maximum.

4.3 High-performance computing inversion of eruption source parameters using approximate Bayesian computation

Statistical parameters inference of a model is the task of inferring parameters as a probability distribution, allowing quantifying the uncertainty of the determined parameters. The main bottleneck of the inference for mechanistic models is the intractability of the likelihood function of the data generating process. Widely applicable frequentist or Bayesian inferential techniques cannot be directly applied for these models due to the absence of the likelihood function. To deal with such a situation, a technique is to estimate the form of the likelihood function, such as in [124]. Other techniques have been developed which are collectively referred to as likelihood-free inference or *approximate Bayesian computation (ABC)* [82].

In a nutshell, ABC algorithms are able to sample from an approximate posterior distribution of the parameters by finding values which yield simulated 'fake' data resembling the observed data to a sufficient degree. To quantify the resemblance, we need to find a discrepancy measure between the simulated and observed dataset. ABCpy [55] is a modular scientific library implementing a range of ABC algorithms in Python. Algorithms are parallelized in a map reduce manner either using a Spark or a MPI backend, which makes them usable in a HPC environment.

The two main difficulties in application of ABC methods are the choice of the discrepancy measure and performing inference for computationally expensive models. In this chapter, we develop a nested parallelization scheme based on MPI for ABCpy to deal with expensive mechanistic models. A metric learning [107] and deep metric learning (see for instance [62]) is used to learn a discrepancy measure between datasets.

We use ABC in conjunction with nested parallelization and deep metric learning to calibrate Tetras, which is an expensive model. As a result of applying ABC to this context, not only we can automatically and efficiently estimate the model parameters, but we can also perform parameter uncertainty quantification in a rigorous manner. This also provides us with a way to validate the numerical model. We illustrate the performance of the developed inferential scheme for a simulated dataset and a real data collected from field observation of ground tephra depositions at 72 ground locations associated with the 2450 BP Pululagua (Ecuador) volcanic eruption [117].

4.3.1 Likelihood free inference

We can quantify the uncertainty of the parameter θ by its posterior distribution $p(\theta|x)$ given the observed dataset $x = x^0$. The posterior distribution is obtained by Bayes' theorem as

$$p(\theta|x^0) = \frac{\pi(\theta)\, p(x^0|\theta)}{m(x^0)}, \tag{4.1}$$

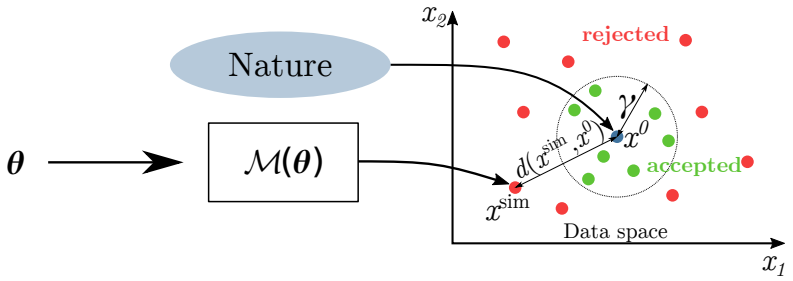


Figure 4.1: ABC rejection sampling: having observed data x^0 provided by nature (the blue dot), we sample parameter values and generate observations through the simulator, that are then accepted (green) or rejected (red) according to their distance from the observation. The visualization is in a 2-dimensional data space.

where $\pi(\theta)$, $p(x^0|\theta)$ and $m(x^0) = \int \pi(\theta)p(x^0|\theta)d\theta$ are, correspondingly, the prior distribution on the parameter θ, the likelihood function, and the marginal likelihood. If the likelihood function could be evaluated, at least up to a normalizing constant, then the posterior distribution could be approximated by drawing a representative sample of parameter values from it using (Markov chain) Monte Carlo sampling schemes [98]. Unfortunately, the likelihood function induced by the volcanic eruption model is analytically intractable. In this setting, approximate Bayesian computation (ABC) [82] offers a way to sample from an approximate posterior distribution and opens up the possibility of sound statistical inference on the parameter θ. Here we only focus on parameter estimation/calibration and uncertainty quantification but we stress that ABC easily allows us to perform parameter hypothesis testing and model selection as well.

Approximate Bayesian computation (ABC)

The fundamental ABC rejection sampling scheme iterates the following steps:

1. Draw θ from the prior $\pi(\theta)$.
2. Simulate a synthetic dataset x^{sim} from the simulator-based model $\mathcal{M}(\theta)$.
3. Accept the parameter value θ if $d(x^{\text{sim}}, x^0) < \gamma$. Otherwise, reject θ.

See Figure 4.1 for a visualization of the above algorithm.

Here, the metric on the data space $d(x^{\text{sim}}, x^0)$ measures the closeness between x^{sim} and x^0. The accepted (θ, x^{sim}) pairs are thus jointly sampled from a distribution proportional to $\pi(\theta)p_{d,\gamma}(x^0|\theta)$, where $p_{d,\gamma}(x^0|\theta)$ is an approximation to the likelihood function $p(x^0|\theta)$

$$p_{d,\gamma}(x^0|\theta) = \int p(x^{\text{sim}}|\theta)\mathbb{K}_\gamma(d(x^{\text{sim}}, x^0))dx^{\text{sim}} \tag{4.2}$$

where $\mathbb{K}_\gamma(d(x^{\text{sim}}, x^0))$ is in this case a probability density function proportional to

$\mathbb{1}(d(x^{\text{sim}}, x^0) < \gamma)$[1]. Besides this choice for $\mathbb{K}_\gamma(d(x^{\text{sim}}, x^0))$, which has been exploited in several ABC algorithms (for instance [8, 52, 50, 81]), ABC algorithms relying on different choices exist, for instance being proportional to $\exp(-d(x^{\text{sim}}, x^0)/\gamma)$ in simulated-annealing ABC (SABC) [1]. In general, $\mathbb{K}_\gamma(\cdot)$ needs to be a probability density function with a large concentration of mass near 0, in which the parameter γ denotes the amount of concentration (the smaller γ, the more concentrated the density is). This guarantees that, in principle, the above approximate likelihood converges to the true one when $\gamma \to 0$. Of course, decreasing the threshold increases the computational cost, as fewer simulations will be accepted.

More advanced algorithms than the simple rejection scheme detailed above are possible, for instance ones based on Sequential Monte Carlo [50, 81], in which various parameter-data pairs are considered at a time and are evolved over several generations, while γ is decreased towards 0 at each generation to improve the approximation of the likelihood function, so that you are able to approximately sample from the true posterior distribution.

4.3.2 ABCpy algorithms and parallelism

A range of algorithms are implemented within ABCpy besides the standard ABC rejection described above. Here, following [55], we present the explicit population Monte Carlo ABC PMCABC [8, 110] algorithm which is a classical ABC algorithm and the difference with probabilistic acceptance scheme like APMCABC [81] to illustrates ABCpy parallel design and the importance of algorithm selection regarding the scalability of the inference scheme. Note that the implementation of PMCABC and APMCABC algorithms and their performance evaluation have not been conducted within this work. We present those algorithms and their scalability because we used APMCABC for our inference on the volcano application and it allows illustrating the parallelism of ABCpy in which we integrated the nested parallelization.

Algorithm 4.2 illustrates the PMCABC algorithm. The principle is to draw points randomly (often called *particles*) in the parameter space according to the prior and forward simulate the model. This step is repeated until the distance between simulated and observed data is considered sufficiently small according to some level of tolerance ϵ_1.

Then, at each iteration t, particles are modified according to a perturbation kernel and the model forward simulates until the level of tolerance ϵ_t is reached. Those succeeding tolerances must be decreasing in order for the algorithm to converge. Most of the ABC algorithm relies on the same principle of drawing and perturbing particles in the parameter space.

Algorithms like PMCABC have an explicit acceptance step, where simulations of the forward model are run until an acceptance criterion is met. There exist other algorithms with a probabilistic acceptance step. In that case, the new particle is accepted with a probability that depends on the distance of the simulated data to the observed data, otherwise we keep the previous value. APMCABC belongs to the second category.

[1] $\mathbb{1}(\cdot)$ is used as an indicator function.

Algorithm 4.2: Population Monte Carlo ABC (PMCABC) algorithm for generating N samples from the approximate posterior distribution. Here $K_t(\cdot|\theta, \Sigma_{t-1})$ is the perturbation kernel, and `weigthed-Covariance` (not shown here) updates the covariance matrix of the perturbation kernel according to the drawn samples and weights. The algorithm requires to specify $q_\epsilon \in [0, 100]$ and a decreasing sequence of thresholds $\epsilon_1 \geq \epsilon_2 \geq \cdots \geq \epsilon_T$ for T iterations.

for *i=1* **to** N **do**
  repeat
    Generate θ from the prior $\pi(\cdot)$
    Generate x^{sim} from $\mathscr{M}$ using θ
  until $d(x^{sim}, x^0) \leq \epsilon_1$
  $d^{(i)} = d(x^{\text{sim}}, x^0)$
  $\theta_1^{(i)} \leftarrow \theta$
  $\omega_1^{(i)} \leftarrow 1/N$
end
$\Sigma_1 \leftarrow 2 * \text{weighted-Covariance}(\theta_1, \omega_1)$
for *t=2* **to** T **do**
  $\epsilon_t = \max(q_\epsilon\text{-th percentile of } d, \epsilon_t)$
  for *i=1* **to** N **do**
    repeat
      Draw θ^* from among θ_{t-1} with probabilities ω_{t-1}
      Generate θ from $K_t(\theta^*, \Sigma_{t-1})$
      Generate x^{sim} from $\mathscr{M}$ using θ
    until $d(x^{sim}, x^0) \leq \epsilon_t$
    $d^{(i)} = d(x^{\text{sim}}, x^0)$
    $\theta_t^{(i)} \leftarrow \theta$
    $\omega_t^{(i)} \leftarrow \pi(\theta) / (\sum_{k=1}^{N} \omega_{t-1}^{(k)} K_t(\theta|\theta_{t-1}^{(k)}, \Sigma_{t-1}))$
  end
  Normalize $\omega_t^{(i)}$ over $i = 1, \ldots, N$
  $\Sigma_t \leftarrow 2 * \text{weighted-Covariance}(\theta_t, \omega_t)$
end

Algorithms with explicit acceptance scheme are usually more expensive computationally speaking as there is no way to know how many forward simulations will be performed until acceptance is reached. Moreover, this can have an impact on load balance when parallelizing the inference procedure.

ABCpy algorithms are usually compounded of the following main steps, as illustrated in Algorithm 4.2 :

1) Sample the parameters from the prior or from existing set of parameters;

2) simulate the model to generate data and compute the distance between observed and simulated data;

3) calculate the weight of parameters;

4) normalize the weights and calculate a covariance matrix.

Steps 1) and 4) are usually lightweight to compute so they are not parallelized. Step 2) on the other hand when the forward simulation of the model is done can be very heavy to compute. Step 3) implies a quadratic complexity regarding the number of parameters, so it is parallelized as well.

There is dependence between iterations. Data from iteration $t-1$ are necessary to compute iteration t, so it is not possible to parallelize over T. On the other hand, the computation of $\theta_t^{(i)}$ can be done independently. ABC algorithms are thus good candidates for parallelization using the map-reduce approach [46]. In the map-reduce parallelism, a master node sends task to execute to workers (the map phase) and once a worker finish to compute its task, it sends its result to the master (the reduce phase). While $\theta_t^{(i)}$ computations are independent, worker-to-worker communications are not necessary. The parallel implementation of ABCpy splits the parallel work into tasks which computes x^{sim} using $\mathscr{M}$ until the acceptance criterion is met and compute the weights $\omega_t^{(i)}$.

ABCpy offers two types of backend to run the tasks. First, a backend based on Apache Spark [125] which is an advanced implementation of map-reduce. A second option is the backend based on MPI. As MPI only offers low-level one-to-one, one-to-many and many-to-many communication operations, ABCpy offers a custom implementation of map-reduce based on those primitives. Within the MPI backend, one node is selected as the master node and attributes work to worker nodes. In MPI, there is no notion of nodes. Instead, the MPI runtime run processes that have a rank attributed, and each process can be bound to a certain number of cores. There is thus one process, that is assigned the role of master, which splits the work into approximately equal parts and assigns the parts to the worker processes.

The parallel model Tetras we want to integrate to ABCpy is MPI-based, which means it needs an MPI runtime which is not available when working with Spark. Thus, we concentrate on the implementation of the MPI backend of ABCpy. To distributes the work, the master process has

to split the work and distribute it on worker processes. To do that, a first naive approach is to split the total number of tasks in batch of equal size. For example, let's say we want to draw $m = 20000$ samples at each iteration of the APMCABC algorithm and distribute it over $n = 100$ processes (or workers). It is possible to assign $m/n = 200$ tasks to each worker. However, as possibly multiple sets of parameters have to be drawn for each $\theta_t^{(i)}$ before the acceptance criterion is met, different workers may have a different amount of work to achieve. And it is not possible to know in advance how much work each worker will have to achieve due to the stochastic nature of the process. This can lead to load imbalance between workers.

Another option is to attributes dynamically work to the worker processes by distributing $n < m$ tasks to the n workers. Then, when a worker finishes its tasks, it sends its results to the master and request a new task. This way, the risk of a load imbalance is lower. The difference between the naive and dynamic allocation of tasks for the MPI backend is illustrated in Figure 4.2.

Other techniques exist to run large number of tasks that exhibit heterogeneous workloads, such as Pilot-Job systems [114, 85]. In this paradigm, a placeholder task is run on resources requested to the scheduler of the cluster (Slurm form example) and is responsible for running tasks within those resources. This allows for application-level resource management. However, while ABCpy historically only offers parallelism based on MPI or Spark, only those options have been considered. Moreover, if using more sophisticated techniques could allow for more scheduling flexibility, it would as well imply more software layers, and thus more complexity in the maintenance and deployment of the package.

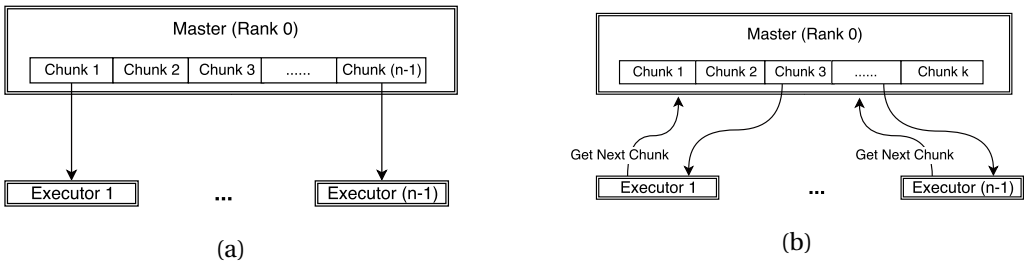
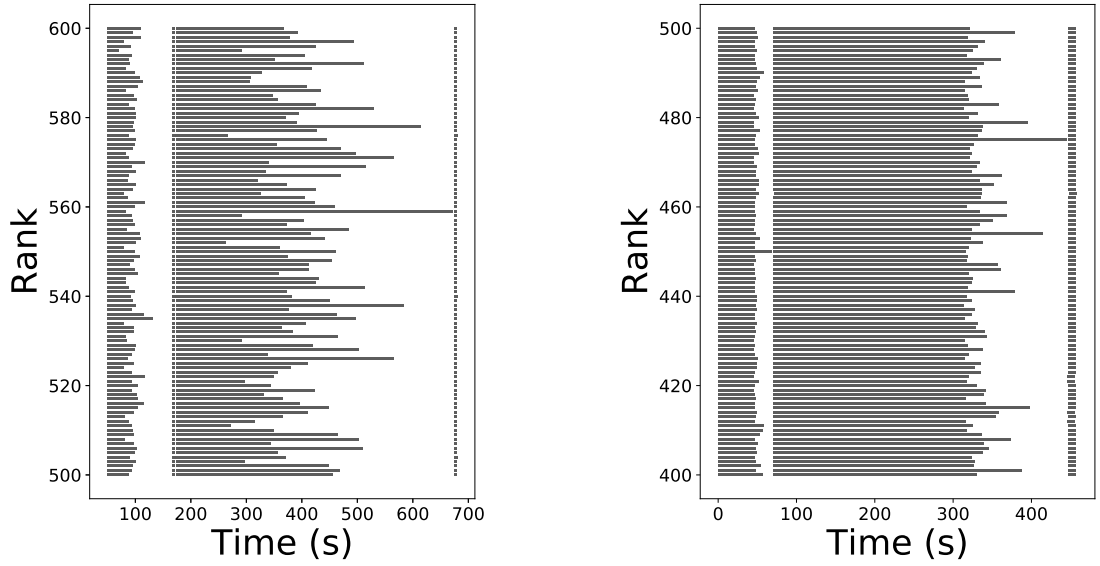


Figure 4.2: Comparison between straightforward (a) and dynamic (b) MPI backend workflow. Figure from [55].

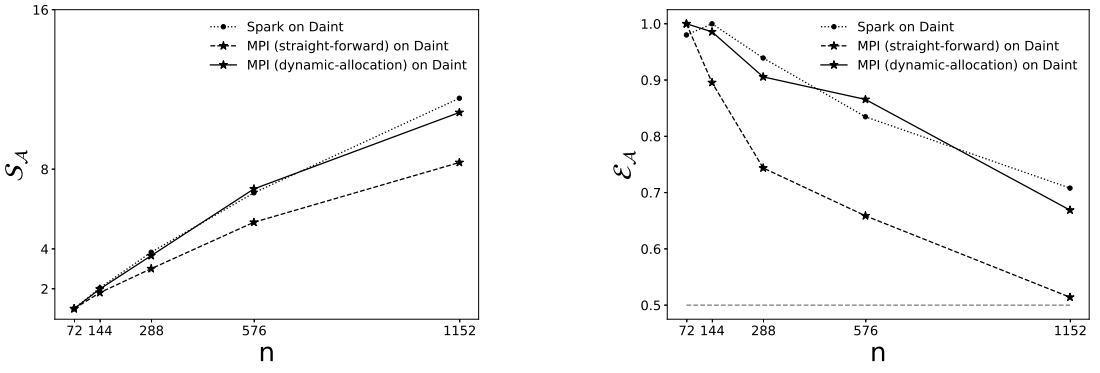
To evaluate the performances of the backend implementations, inference on weather prediction benchmark model known as the Lorenz model [83] have been performed on the Piz Daint supercomputer of the CSCS. Figure 4.3 displays the total amount of time spent by each worker process, either with naive or with dynamic task allocation with the PMCABC algorithm. Figure 4.4 shows the performance behaviour of Spark, MPI straightforward and MPI dynamic backends.

As stated above, ABC algorithms with explicit acceptance step like PMCABC comes with an intrinsic imbalance when parallelizing them. Indeed, some tasks can take way longer to terminate than others, and this can become difficult to mitigate when a large number of



(a) (b)

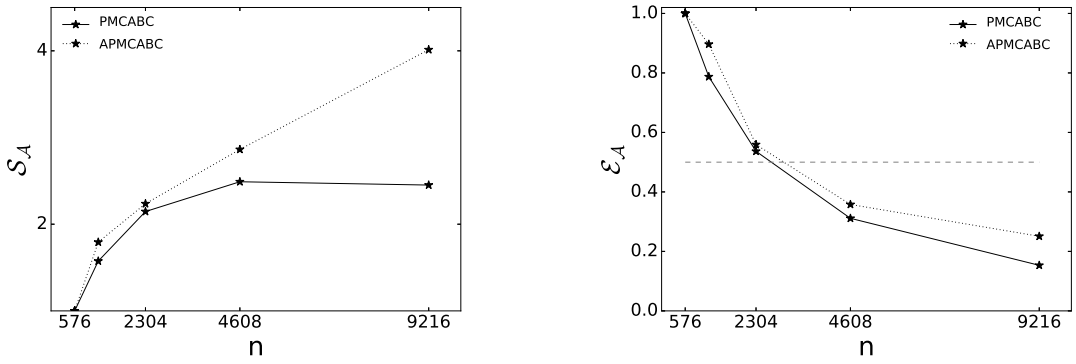
Figure 4.3: Imbalance of the PMCABC algorithm using MPI straightforward (a) and dynamic allocation (b) backend for the Lorenz95 model (T=1024). Note the large difference in the time-scale (in seconds) on the horizontal axis. Figure from [55].



(a) (b)

Figure 4.4: Speedup and efficiency of the PMCABC algorithm for Lorenz95 model using Spark and MPI backend with different number of cores n. Figure adjusted from [55].

workers are used. Algorithms with a probabilistic acceptance step, like APMCABC, are, on the other hand, much more scalable. This is illustrated in Figure 4.5 where inference on the Lorenz model is performed on a large number of cores and a dynamic allocation is used with PMCABC and APMCABC algorithms.



(a) (b)

Figure 4.5: Comparison of speedup and efficiency for PMCABC and APMCABC using the Lorenz95 model (T=1024) with different number of cores n. Figure adjusted from [55].

4.3.3 Nested parallelization

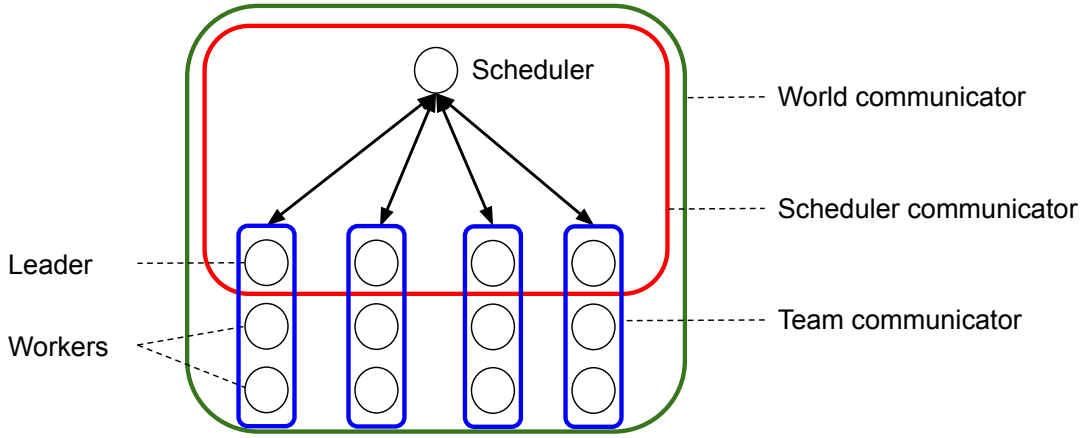


Figure 4.6: Nested parallelization: Description of the communication architecture of the nested MPI parallelization for ABCpy.

As mentioned above, ABCpy provides the user with seamless parallelization of ABC algorithms using MPI or Spark. Modern cluster nodes usually have multiple cores, and by default the MPI backend runs one simulation of the model per core. Yet, in case the model supports basic

multithreading at the level of a single machine, the backend can be accordingly configured to achieve this.

However, previous versions of ABCpy did not support models that were parallelized using MPI, which is the case for the studied volcano model. The challenge in enabling MPI model support is the fact that MPI code uses an object called MPI communicator to control communication. In the Apache Spark backend, this communicator is just not available due to the standard system setup and thus not usable in standard installations. In the MPI backend, the communicator is available but used by the backend itself that has to coordinate the parameter distribution and forward simulations. Thus, we made code contributions to ABCpy that enabled support for MPI parallelized models, broadening the field of applications beyond the volcanic model discussed here.

Technically, MPI uses an object called communicator in order to control communication between different ranks. Therefore, in order to achieve nested parallelization ABCpy creates two types of communicators: each forward model uses a team communicator to parallelize the computation on the ranks allocated by the backend object; moreover, the scheduler communicator is used by the overall master (the scheduler) to control the whole execution. The architecture is visualized in Figure 4.6. Note that one process of each team communicator is part of the scheduler communicator as well in order for communication to be successful.

By default in MPI, a global communicator exists, which is called the *world communicator*. Each process has a unique identifier (the rank) within this communicator. It is then possible to split this communicator in custom sub-communicators in which MPI operations can be performed. Note that when splitting a communicator, the encompassing communicator is not destroyed, but communicators that are subsets of the parent are created. Communicators are created using the *comm_split* primitive, which takes as parameters a communicator to split, a *key* and a *colour*. The key is a unique identifier that will serve to attribute a rank to the process in the new communicator. It can be the rank of the parent communicator. The colour is a value that serves to identify in which communicator the process will be placed. Every process in the same parent communicator calling the split primitive with the same colour will be placed in the same new communicator.

Concretely, a first split is applied to create the team communicators. In the current implementation, the process 0 is the master (or scheduler) process. Each process with a rank > 0 is placed in a team communicator. Then, a new communicator encompassing the master process and the process of rank 0 of each team communicator is created. Leader of each team communicator will then ask for tasks to the scheduler, distribute and gather the work within their team communicator and sends the results back to the scheduler when finished.

To be integrated into ABCpy nested parallelization, MPI parallelized models should not use the world communicator to communicate, but rather use a custom communicator that is passed as a parameter to the model. This is something that has to be taken into account while MPI codes can be developed without this in mind. Tetras code had to be adapted to this end.

Moreover, while ABCpy is a Python package and Tetras is a C++ code, it is necessary to have a way to pass data structures between Python and C++ (parameters, results, MPI communicator). This is achieved using SWIG [10, 9] which allows calling C and C++ code from scripting languages, including Python, and exchange data structures like MPI communicators and arrays of raw data between Python and C/C++.

4.3.4 Inference of Tetras parameters

For the purpose of the present study, the model $\mathscr{M}$ is parametrized in terms of two of the plume parameters, namely the initial plume velocity U_0 and the radius of the plume at the vent of volcano R_0, collectively defined as $\theta = (U_0, R_0)$. In this work, we assume the initial temperature, the initial gas mass fraction of exsolved volatiles and the diffusion coefficients to be known. However, the model could also be parametrized in terms of those parameters.

Initial temperature T_0 and initial gas mass fraction of exsolved volatiles n_0 have an influence on the plume height. For the purpose of this work, their values are kept constant, respectively $T_0 \approx 1200K$ and $n_0 \approx 0.01$. Diffusion in atmosphere D_a and diffusion in plume D_p have been empirically chosen as $D_a = 300m^2 \cdot s^{-1}$ and $D_p = 1500m^2 \cdot s^{-1}$.

By assuming that $\mathscr{M}$ is true, then we can simulate the deposition of tephra at those 72 locations, denoted as x^{sim}. If we have observed datasets x^0, can we quantify the uncertainty or estimate the model parameters ? As Tetras is a stochastic model in nature, the observed dataset could have been simulated using a whole spectrum of values for θ with different likelihood. To quantify this stochastic uncertainty in the parameters simulating the observed dataset, we develop a likelihood-free approximate Bayesian inference scheme.

4.3.5 Distance learning

Traditionally, distance between x^{sim} and x^0 are defined by summing over Euclidian distances between all possible pairs composed of one simulated and one observed data point in the corresponding datasets. Recently, distances for ABC has also been defined through accuracy of possible classification of x^{sim} and x^0 [67] or by Wasserstein distance [16], under the assumption that the data points in each datasets are identical and independently distributed and they are present in a large number in both x^{sim} and x^0. We notice here that we only have one data point in the observed dataset for a volcanic eruption field study and also due to the very expensive simulation model we can only have few data points in the simulated dataset. Hence, here we concentrate on the definition of distances through Euclidian distance while we only have one data point in both x^{sim} and x^0.

While performing ABC for inference, problems may arise in cases where the data x is high-dimensional. In fact, the number of simulations needed before you get close enough to the observation increases with the dimension of the data space. Therefore, a common practice in ABC literature is to define d as Euclidian distance between a lower-dimensional summary

statistics $S : x^{\text{sim}} \mapsto S(x^{\text{sim}})$, so that $d(x^{\text{sim}}, x^0) < \gamma$ would be replaced by

$$d(S(x^{\text{sim}}), S(x^0)) < \gamma$$

in Equation (4.2), boiling down to obtaining an approximation to the following likelihood function

$$p_{d,\gamma}(S(x^0)|\theta) = \int p(x^{\text{sim}}|\theta)\mathbb{K}_\gamma(d(S(x^{\text{sim}}), S(x^0)) < \gamma)dx^{\text{sim}} \tag{4.3}$$

so that now (θ, x^{sim}) are jointly sampled from a distribution proportional to $\pi(\theta)p_{d,\gamma}(S(x^0)|\theta)$ when performing ABC inference.

Reducing the data to suitably chosen summary statistics may also yield more robust inference with respect to noise in the data. Moreover, if the statistics is sufficient, then the above modification provides us with a consistent posterior approximation [51], meaning that we are still guaranteed to converge to the true posterior in the limit $\gamma \to 0$. As sufficient summary statistics are not known for most of the complex models, the choice of summary statistics remains a problem [44] and they have been previously chosen in a problem-specific manner [18, 57, 67].

For volcanic eruption model, y cannot be easily transformed into summary statistics $S(y)$ as there is a complex spatial dependence involved between the deposited tephra at each location. Hence, here we consider two possible ways of learning a distance directly between two datasets x_1 and x_2 rather than between the extracted summary statistics. The first entails constructing a Mahalanobis distance,

$$d_M(x_1, x_2) = \sqrt{(x_1 - x_2)^T M(x_1 - x_2)} \tag{4.4}$$

where M is a $d \times d$ positive semi-definite matrix.

The second approach uses instead a neural network to transform non-linearly the dataset in a new space; this is usually referred to as *deep* metric-learning and is a well-developed field in computer vision literature [62]. The learned distance is the Euclidian distance between the learned embeddings

$$d_{NN}(x_1, x_2) = ||g_w(x_1) - g_w(x_2)||_2 \tag{4.5}$$

where $g_w(\cdot)$ denotes the transformation applied by the network with weights w and $||\cdot||_2$ denotes the L^2 norm.

We don't go into the detail of the distance learning here and refer the interested reader to the original paper [91] for details on this matter. Simply put, the goal is to find a distance between observations x_1 and x_2 that corresponds to the Euclidian distance between the parameters θ_1 and θ_2 that were used to generate the observations (called the true distance).

The method used to learn the Mahalanobis distance is called the Sparse Distance metric learning (SDML) algorithm [95]. Two methods to learn the weights of the neural network are considered, the contrasive [68] and triplet [101]. A technique called summary statistics learning is also considered in [91] but is considered less performant. The triplet loss distance is selected as the most performant distance.

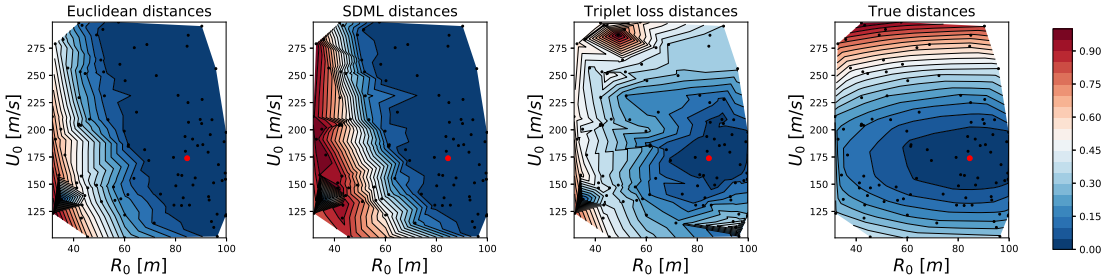


Figure 4.7: Comparison of metrics: We compare Euclidian distance d_E and the distances learned with the discussed techniques on the 300 elements of the training set, computing them between an observation point x^0 and the other 99 samples in the test set, taken to be a reference. x^0 was simulated using $U_0 = 173.87m \cdot s^{-1}, R_0 = 84.55m$ (red dot). All the distances are scaled between [0,1]. The small black points denote the position of the reference data, from which the contour plot is obtained by triangulation. Euclidian distance is the distance between summary statistics of x^{sim} and true distance is the Euclidian distance between the parameters, reported for reference.

In Figure 4.7, comparison between the Euclidian distance between the parameters generating the datasets ('true distance') with the learned distance functions on the corresponding datasets, namely the Mahalanobis one with SDML algorithm and the triplet loss distances; the Euclidian distance between outputs of the model is also reported. The comparison is made in the following way: 400 parameters-simulation pairs have been generated, with parameters drawn uniformly on the interval $[30,100]m$ for R_0 and $[100,300]m \cdot s^{-1}$ for U_0. This dataset is split into a training one (with 300 samples) and a test one (with 100 samples). Distances are learned on the training set, and then all the distances between a chosen element in the training set x^0 and the 99 remaining samples in the same test set ('reference samples') are computed; the learned distances in parameter space are then plotted by using the corresponding parameter value for each observation. x^0 was simulated using $\theta_0 = (173.87m \cdot s^{-1}, 84.55m)$.

We see that the minimum values of the distances are much more concentrated around the true parameter value for the triplet loss distance in comparison to the SDML one and the Euclidian. Note also that the neural network based distance is able to partially reproduce the behaviour of the distance between the true parameter values.

4.3.6 Posterior inference

To draw Z samples approximating the posterior distribution $p(\theta|x^0)$, we keep all the tuning parameters for the APMCABC fixed at the default values suggested in ABCpy package, except the acceptance rate cutoff, which was chosen to be 0.03. Different number of steps and samples are used for the inference on the simulated and real data; check Section 4.3.9 for more details. We consider independent uniform prior distributions for the parameters with a pre-specified range for each of them, $U_0 \sim U[100,300]\,m\cdot s^{-1}$, $R_0 \sim U[30,100]\,m$. To explore the parameter space of $\theta = (U_0, R_0)$, we use a two-dimensional truncated multivariate Gaussian distribution as the perturbation kernel. APMCABC inference scheme centres the perturbation kernel at the sample it is perturbing and updates the variance-covariance matrix of the perturbation kernel based on the samples learned from the previous step.

4.3.7 Parameter estimation

Given an observed dataset x^0, our main interest is to estimate the corresponding θ. In decision theory, Bayes estimator minimizes the posterior expected loss, $\mathbb{E}_{p(\theta|x^0)}(\mathscr{L}(\theta,\bullet)|x^0)$ for an already chosen loss function $\mathscr{L}$. If we have Z samples $(\theta_i)_{i=1}^Z$ from the posterior distribution $p(\theta|x^0)$, the Bayes estimator can be approximated as

$$\hat{\theta}_B = \arg\min_{\theta} \frac{1}{Z}\sum_{i=1}^{Z} \mathscr{L}(\theta_i,\theta). \tag{4.6}$$

As we consider the Euclidian loss function $\mathscr{L}(\theta,\theta') = (\theta-\theta')^2$, the Bayes estimator can be shown to be the posterior mean $\mathbb{E}_{p(\theta|x^0)}(\theta|x^0)$, corresponding to an approximate one $\hat{\theta} \approx \frac{1}{Z}\sum_{i=1}^{Z}\theta_i$.

4.3.8 Computational considerations

For this work, we used the supercomputer Piz Daint of the CSCS, where each compute node consisted of 36 Intel Broadwell Xeon E5-2695 v4 @ 2.10 GHz cores. Each forward MPI simulation was run on one node, resulting in the use of Z nodes (up to 500 in our case). With this setup, each forward simulation takes about 15 minutes to terminate, which makes it possible to run 6 iterations of the APMCABC algorithm in less than 2 hours.

We stress that ABC with distance or summary statistics learning comes at the expense of a larger computational cost with respect to directly using the Euclidian distance between model outputs in ABC, case in which no training step would be needed. When applying the learning approach, instead, you need to generate the training data, which consist of results of 400 forward simulations in our case. This is quite expensive given the model we are considering, but remains an approximately equivalent computational cost of one iteration of APMCABC when using 500 samples.

You then need to perform the training. Note that, once the training data is generated, the SDML technique requires a much shorter time for fitting than the time required for training the neural network in the other cases. However, in the overall balance, when compared with data generation time and the ABC inference time, the training step has a much smaller cost no matter the chosen method, and has to be performed only once. Therefore, it does not make sense to prefer a distance learning method over another just because of a shorter training time.

During inference, the computation of the new distance only requires multiplying output of the mechanistic model by some matrices with all distance learning techniques (as transforming some data with a neural network simply consists of matrix multiplications and the application of element-wise non-linearity functions). Therefore, the impact of distance learning on the computational complexity of the inference is very small, comparable to the use of hand-chosen summary statistics.

In general, the larger computational cost is balanced by a more efficient ABC inference scheme and a better approximation of the true posterior, given the same computational budget to the inference itself. We also remark that this approach can be thought of as a pre-processing technique, as it can be used with any ABC algorithm. Also, once the training has been performed, the same learned distance can be exploited for inference on several observations of the same physical process.

4.3.9 Results of parameters inference

To validate the performance of the inference scheme, we first try to infer the posterior distribution (using $Z = 100$ samples and 12 iterations of APMCABC scheme) and the Bayes estimate of the parameters given a dataset which was simulated from $\mathscr{M}(\theta)$ using a known parameter configuration, $\theta^* = (173.87m \cdot s^{-1}, 84.55m)$. In Figure 4.8, we plot the approximate posterior distribution inferred by APMCABC using the triple distance learned with best value of ϵ according to the above investigation, and the Bayes estimate for θ. We see the true parameter value $\theta^* = (173.87m \cdot s^{-1}, 84.55m)$ falls in a region with high posterior probability and the Bayes estimate $\hat{\theta} = (172.09m \cdot s^{-1}, 86.92m)$ being close to the true value.

After the validation of our inference scheme for a simulated dataset, we perform inference to learn the posterior distribution (using $Z = 500$ samples and 6 iterations of APMCABC scheme) and the Bayes estimate of the parameters given the tephra deposits at 72 ground locations associated with the 2450 BP Pululagua (Ecuador) volcanic eruption [117] in Figure 4.9. The Bayes estimate of θ and the posterior correlation between (U_0, R_0) are correspondingly $\hat{\theta} = (\hat{U}_0, \hat{R}_0) = (123.84m \cdot s^{-1}, 56.78m)$ and $Corr_{\text{post}}(U_0, R_0) = -0.79$, indicating that the similar deposition of tephra could have been caused by combinations of higher injection velocity and narrower radius at vent or a lower injection velocity and wider radius at vent.

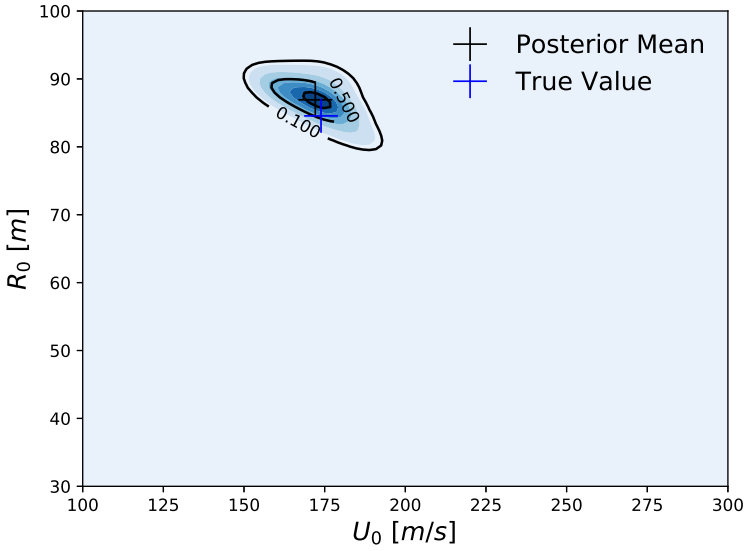


Figure 4.8: Inference on Simulated Data: Approximate posterior contour plot obtained through kernel density (bandwith=0.8) estimate from $Z = 100$ samples after 12 iterations of APMCABC scheme from the approximate posterior itself and Bayes estimate (black cross) of the parameters given a dataset which was simulated from $\mathscr{M}(\theta)$ using a known parameter configuration, $\theta^* = (173.87m \cdot s^{-1}, 84.55m)$ (blue cross). The prior on the parameters was chosen to be uniform in the region represented in the plot.

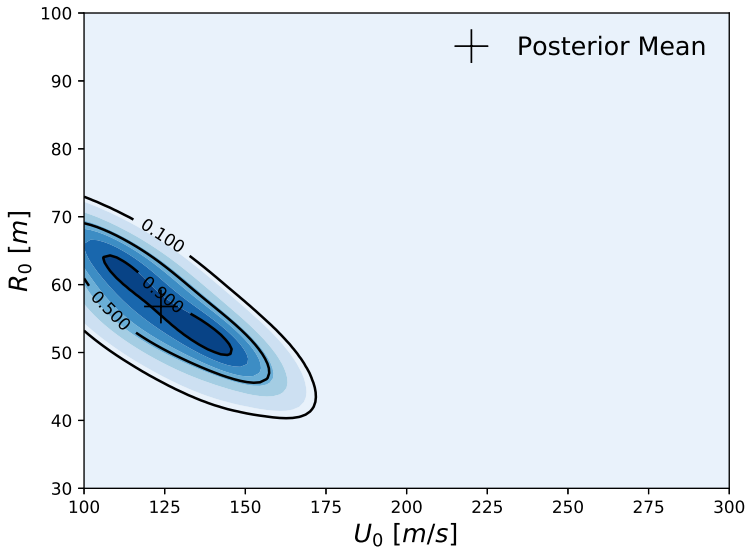


Figure 4.9: Inference on Real Data: Approximate posterior contour plot obtained through kernel density (bandwith=0.8) estimate from $Z = 500$ samples after 6 iterations of APMCABC scheme from the approximate posterior itself and Bayes estimate (black cross) of the parameters given the tephra deposits at 72 ground locations associated with the 2450 BP Pululagua volcanic eruption [117]. The prior on the parameters was chosen to be uniform in the region represented in the plot.

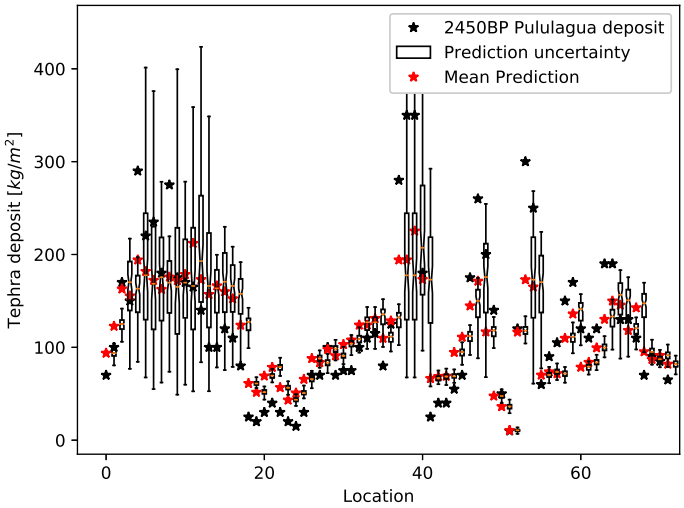


Figure 4.10: Posterior Prediction Check: To validate the numerical volcano model and the inference scheme we perform a posterior prediction check by simulating 100 datasets, each using a different parameter sample drawn from the posterior distribution. Here, we plot the tephra deposit from field observation (black) used for inference, the boxplot of the empirical predictive distribution (white patch) and the mean predicted dataset (red) at each location.

Next we do a posterior predictive check to validate our model and inference scheme. The main goal here is to analyse the degree to which the observed data deviate from the data generated from the inferred posterior distribution of the parameters. Hence we want to generate data from the model using parameters drawn from the posterior distribution. To do so, we first draw 100 parameter samples from the inferred approximate posterior distribution and simulate 100 datasets, each using a different parameter sample. We call this simulated dataset as the predicted dataset from our inferred posterior distribution and present the mean predicted dataset (red) compared with the observed dataset (black) in Figure 4.10.

Note that since we are dealing with the posterior distribution, we can also quantify uncertainty in our predictions. We plot the boxplot of the empirical predictive distribution (white patch) at each location to get a sense of uncertainty in the prediction. This shows a good prediction performance of the numerical model of volcanic deposition and the proposed inference scheme. The discrepancies in some of the locations can mainly be explained by the inability of the numerical model in providing a complete description of the process of volcanic eruption and by the uncertainty on the field measures.

4.4 Coupling the Nelder-Mead optimization algorithm and a SEER based model for inversion of eruption source parameters

In the previous section, we showed how to use ABC algorithms to infer parameters of a tephra dispersion model. Another technique used for the inversion of ESP is to apply an optimization algorithm that explores the parameter space in search of a good solution using some heuristics. A widely used technique is the inversion procedure included in Tephra2 [38], which applies the downhill simplex method of Nelder-Mead [89] to the Tephra2 model. Here, we aim at inverting ESP based on a SEER based model.

We consider here volcano-semianalytical-seer, which is lightweight to compute, and we project to include a heavier model like volcano-lagrangian-seer in the inversion scheme at a later time. Optimization or inference algorithms can be intrinsically parallel (as inference schemes presented in Section 4.3). In that case, the parallelization of the optimization scheme or a nested parallelization of the forward model in a parallel optimization scheme is possible. In other situations, as with the Nelder-Mead simplex method, the optimization scheme is purely sequential, and requires a parallel forward model with good strong scaling behaviour if using a heavy numerical model. As we showed in chapter 3, volcano-lagrangian-seer has a scaling behaviour which makes possible to use it either in a sequential or parallel optimization scheme.

We present here some preliminary results of the coupling of volcano-semianalytical-seer with the Nelder-Mead simplex algorithm. For this, we couple the forward model with the optimization framework of the Python package SciPy [116], which implements a range of optimization algorithms. Moreover, the Python interface makes the coupling with other optimization frameworks easy. Those early results are encouraging as convergence can be obtained with a reasonable number of forward simulations, but challenges remain in the sense that the algorithms seems to easily be trapped in a local optimum region.

4.4.1 The Nelder-Mead simplex method

The Nelder-Mead simplex is a method that aims at minimizing a function f defined on a space of N parameters. Each set of parameters represent a point in this space. Let's assume we have a set of $N+1$ distinct parameter sets, each set is a vertex of a simplex in the N-dimensional space. For example, in a two-dimensional space a simplex of three points, or a triangle, can be defined.

Starting from an initial simplex, the Nelder-Mead algorithm changes the point of the simplex that has the highest value at each iteration with a reflection, expansion, contraction or shrink operation until some convergence criterion is met. Algorithm 4.3 gives the Nelder-Mead algorithm.

Algorithm 4.3: The Nelder-Mead simplex algorithm. $\alpha, \gamma, \rho, \sigma$ are coefficients such as $\alpha > 0, \gamma > 1, 0 < \rho \leq 0.5$. Standard values are $\alpha = 1, \gamma = 2, \rho = 1/2, \sigma = 1/2$. In the used version of the simplex algorithm, vertices of a generated simplex are clipped if they are out of bound.

```
Construct a simplex of N+1 points x_1, x_2, ..., x_{N+1}
repeat
    Compute f on the simplex points and sort them as f(x_1) ≤ f(x_2) ≤ ... ≤ f(x_{N+1})
    Compute the center of gravity x_0 of x_1, x_2, ..., x_N
    Compute x_r = x_0 + α(x_0 − x_{N+1}), the reflection of x_{N+1}
    if f(x_1) ≤ f(x_r) ≤ f(x_N) then
        Replace x_{N+1} by x_r
    else if f(x_r) < f(x_1) then
        Compute x_e = x_0 + γ(x_r − x_0), the expansion of the simplex
        if f(x_e) ≤ f(x_r) then
            Replace x_{N+1} by x_e
        else
            Replace x_{N+1} by x_r
        end
    else
        Compute x_c = x_0 + ρ(x_{N+1} − x_0), the contraction of the simplex
        if f(x_c) < f(x_{N+1}) then
            Replace x_{N+1} by x_c
        else
            Shrink the simplex by replacing all points except the best (x_1) with
              x_i = x_1 + σ(x_i − x_1)
        end
    end
until convergence criterion is met
```

4.4.2 Coupling volcano-semianalytical-seer with the downhill simplex implementation of SciPy

Let's recall that volcano-semianalytical-seer takes the parameters $z_0, M, r_0, U_0, T_0, n_0$ as well as particle characteristics TGSD and density. In the present work, we consider z_0 and particle characteristics to be fixed. It remains a set of five parameters to optimize, which are respectively the total erupted mass, the initial radius of the plume, the initial temperature and the ratio of exoslved volatiles. The physical models (plume, umbrella cloud and settling velocity models) have been implemented in C++, historically to be used within the Teras model. Then, the model is solved by calling C++ functions from Python using SWIG, and using averaged values to solve volcano-analytical-seer (see Chapter 3 for details).

The optimization is made by minimizing an objective function that is passed to the *minimize* function of SciPy. In the present work, this function computes a MSE error between observed and simulated data. Experiments using the MAPE error have been made, but this type of error tends to always privilege underestimated simulation results, while the error is at maximum 100% when the result is underestimated. [38] use a χ^2 error. More investigation to determine an appropriate error measurement is needed. We pass bounds to the algorithm to constrain the choice of parameters within acceptable values[2]. In the used implementation of SciPy, Nelder-Mead coefficients are chosen as standard values, respectively $\alpha = 1, \gamma = 2, \rho = 1/2, \sigma = 1/2$ and the initial simplex is constructed as $x_i = (1+\delta) \cdot x_0$ given that x_0 is an initial parameter passed to the algorithm and $\delta = 0.05$. It is also possible to specify a custom initial simplex.

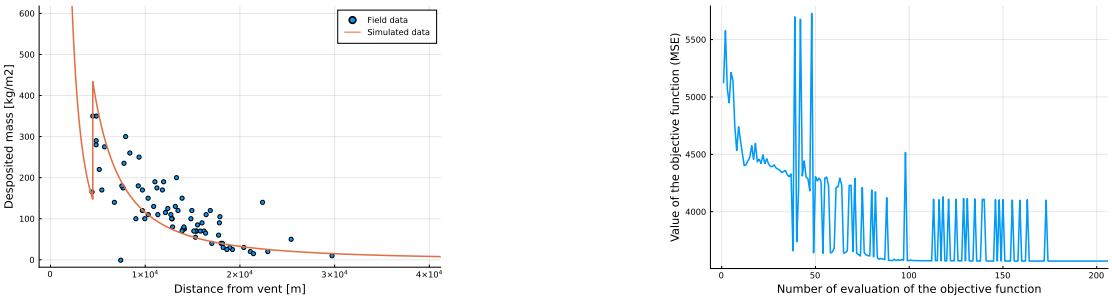
Preliminary results

In order to benchmark the inversion scheme, we run it against depositions of the Pululagua 2450BP eruption with various initial guesses passed to the algorithm. Figures 4.11 and 4.12 shows results of those inversions runs. For each, we specify the initial guess passed to the algorithm and the final result of the inversion. The bounds set for those runs are $M \in [3.5e11, 5.5e11]kg, U_0 \in [20, 400]m/s, r_0 \in [20, 200]m, n \in [0.01, 0.05], T_0 \in [1000, 1300]K$ and $z0 = 100m$. The continuity of deposition between plume and umbrella cloud is not imposed. Plume height H_t is also specified, but note that here plume height is an output of the model, not an input. An inversion with plume height as input would be possible by using the plume parameters inversion strategy described in Section 4.2. In that case, an inversion of plume parameters from plume height would be necessary at each iteration of the Nelder-Mead algorithm.

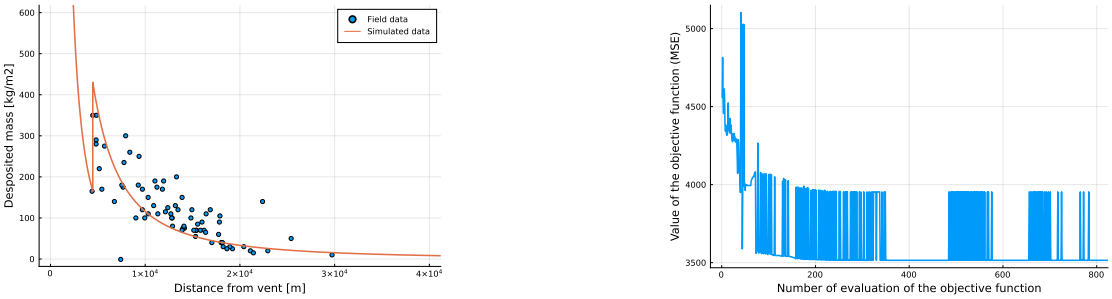
For each run, we show the deposition obtained with the inverted parameters, as well as the evolution of the value of the objective function at each evaluation. Note that it is not the evolution of the best value of the current simplex, but the value of every evaluation of the function. We recall that in the worst case, an iteration of the Nelder-Mead simplex can generate

[2] Support for bounds in the Nelder-Mead algorithm has only been added recently in SciPy. Version 1.7.0 released in June 2021 is necessary

up to three evaluations of the objective function (in the case of a reflection, a contraction and a shrink operation).



(a) Initial guess : $M = 4.5e11kg, U_0 = 170m \cdot s^{-1}, r_0 = 80m, n = 0.01, T_0 = 1200K, H_t \approx 26.4km, MSE \approx 5120$.
Inversion result : $M \approx 4.6e11kg, U_0 \approx 189.6m \cdot s^{-1}, r_0 \approx 108.8m, n \approx 0.01, T_0 \approx 1171.7K, H_t \approx 31km$. Best achieved $MSE \approx 3570$.

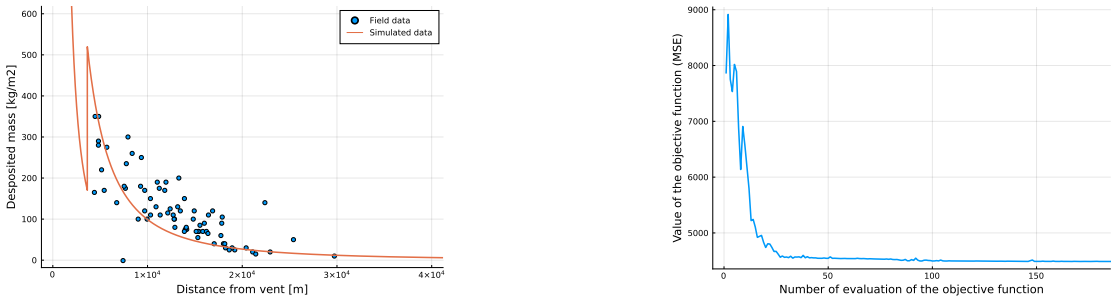


(b) Initial guess : $M = 4.5e11kg, U_0 = 150m/s, r_0 = 100m, n = 0.01, T_0 = 1200K, H_t \approx 28.7km, MSE \approx 4556$.
Inversion result : $M \approx 5.01e11kg, U_0 \approx 150.6m \cdot s^{-1}, r_0 \approx 120.7m, n \approx 0.01, T_0 \approx 1038.8K, H_t \approx 32.7km$. Best achieved $MSE \approx 3515$.

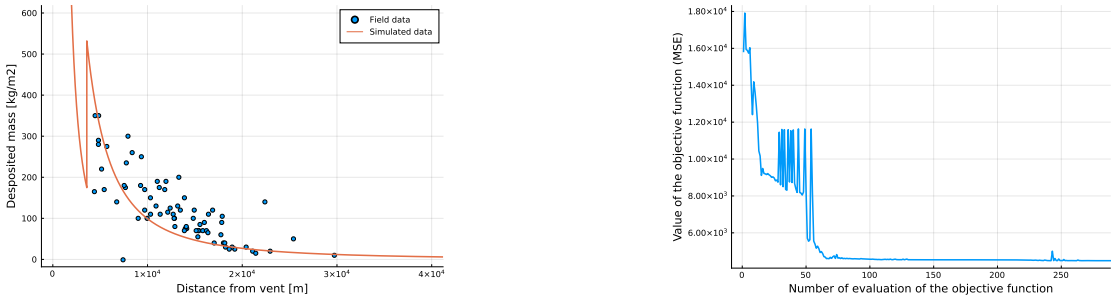
Figure 4.11: Comparison of inversions results using volcano-semianalytical-seer coupling with Nelder-Mead simplex algorithm. For each inversion run, we show the deposition obtained after the inversion (left) and and evolution of values of evaluations of the objective function (right).

First, we can observe that in the four cases we get visually close deposition patterns, but with rather different plume parameters. There are also two plume heights that emerges, $\approx 32km$ in Figures 4.11a and 4.11b and $\approx 27km$ in Figures 4.12a and 4.12b. We remark that obtained plume heights are compatible with results found in [117].

It seems that the algorithm has difficulty to escape a given region of the parameter space. It would then be interesting to explore other optimizations strategies or algorithms that would maybe be more adapted to this problem. For example, by applying multiple downhill simplex algorithms in parallel with different starting points, applying a simulated annealing or parallel simulated annealing or some other metaheuristic. The quality of the optimization procedure is, however, often problem dependent, and more investigations are needed to choose the appropriate technique.



(a) Initial guess : $M = 4.5e11kg, U_0 = 150m \cdot s^{-1}, r_0 = 50m, n = 0.01, T_0 = 1200, H_t \approx 21.3km, MSE \approx 7856$.
Inversion result : $M \approx 3.5e11kg, U_0 \approx 400m \cdot s^{-1}, r_0 \approx 141.5m, n \approx 0.05, T_0 \approx 1012.6K, H_t \approx 27.2km$. Best achieved $MSE \approx 4488$.



(b) Initial guess : $M = 4.5e11kg, U_0 = 80m \cdot s^{-1}, r_0 = 30m, n = 0.01, T_0 = 1200K, H_t \approx 16.1km, MSE \approx 15790$.
Inversion result : $M \approx 3.5e11kg, U_0 \approx 400m \cdot s^{-1}, r_0 \approx 60.3m, n \approx 0.01, T_0 \approx 1300K, H_t \approx 27.2km$. Best achieved $MSE \approx 4490$.

Figure 4.12: Comparison of inversions results using volcano-semianalytical-seer coupling with Nelder-Mead simplex algorithm. For each inversion run, we show the deposition obtained after the inversion (left) and and evolution of values of evaluations of the objective function (right).

The number of iterations to converge can greatly vary between inversions, but are always less than 1000 iterations. Thus, applying this inversion procedure using volcano-lagrangian-seer as the forward model instead of volcano-semianalytical-seer on 500 cores of Yggdrasil would take less than three hours for Pululagua and certainly less than two hours for Cotopaxi, if we consider that the number of evaluations to convergence would be of the same order of magnitude. This would take less than 9 hours on 900 cores if the same procedure was applied to the 1996 eruption of Ruapehu. This would, however, requires to implement Degruyter plume model in a more efficient way than the current Matlab implementation. This is again by supposing that the number of evaluations to convergence would remain of the same order of magnitude. See Section 3.4.1 for performance analysis of volcano-lagrangian-seer. Hybrid strategies could as well be conceived, where a first step of inversion would be done with volcano-semianalytical-seer to find one or multiple initial guess for a refinement with inversion using volcano-lagrangian-seer.

4.5 Conclusion

In this chapter, we introduced the notion of eruption source parameters and discussed three inversion techniques based on different optimization algorithms and forward models.First, we presented a simple Monte Carlo strategy that allows finding plume source parameters given a plume height. This technique works because the forward model is very fast to solve, and that it is necessary to find only one set of compatible parameters.

Then, we presented approximate Bayesian computation techniques and algorithms and the Python package ABCpy which offers a large range of ABC algorithms and is parallelized using Spark or MPI. We performed parameter inference on the Tetras model using ABCpy and a sophisticated distance learning technique. We presented how we contributed to the code of ABCpy to offer a nested parallelization architecture which allows integrating MPI parallelized models, such as Tetras, in the ABCpy framework. Advantages of statistical inference technique is that they are inherently parallel (it is possible to run hundreds or thousands of instances of the forward model in parallel at each iteration) and they allow for an estimation of the posterior distribution of parameters, which allow for uncertainty quantification, instead of finding a unique parameter set considered as optimal. They are, however, very computationally expensive (it is necessary to run hundreds or thousands of instances of the forward model at each iteration). We demonstrated that this task is now achievable in a rather short time for moderately heavy MPI models such as Tetras thanks to our nested parallelization scheme given that a high-end HPC infrastructure is available. The inversion procedure took approximately 1000 $nodes \times hour$ on Piz Daint (36000 $cores \times hour$), or 500 nodes (18000 cores) for two hours. The same procedure would take multiple days on a medium-range HPC infrastructure now routinely found in research centres and universities. We can estimate that this procedure would take less than three days on 500 cores of the Yggdrasil cluster of the Geneva's higher education institutions ($\frac{36000 cores \times hour}{500 cores} = 72 hours$, and by taking into account that processors in Yggdrasil are of newer generation than those of Piz Daint). This kind of inversion procedure

then still requires a rather important computing power, but it will become more and more affordable as the time passes.

Another difficulty in inference and optimization is the design of a distance or discrepancy measure between simulation outputs and / or field measurements. For the statistical inference, we relied on a sophisticated distance measure obtained by machine learning whose goal is to make the distance in the output space resemble the distance in the parameter space.

Finally, we presented preliminary results on an inversion strategy based on the Nelder-Mead simplex algorithm and our new volcano-semianalytical-seer model. The procedure is a sequential improvement of an objective function based on a geometrical argument. Its advantage compared to the statistical inference technique is to be less computationally intensive, but it outputs only one value of parameter considered optimal and provides no insight on the uncertainty quantification of the result. Another drawback seems the difficulty of the algorithm to escape local optimum of the objective function. However, given the reasonable amount of objective function evaluations and the parallelization of volcano-lagrangian-seer, this latter model could be used with such an inversion strategy on an HPC cluster.

A strength of our inversion strategies is to rely on a first principle plume model. Again we stress the flexibility of our design. Tetras and volcano-lagrangian-seer can be coupled with various plume models found in the literature and they can easily be integrated to various inversion schemes. The plume height seems to be the determining factor in the deposition profile, for future work, it could then be interesting to perform inversion taking into account the plume height and the simple inversion of plume parameters from plume height described in this chapter to lower the number of dimensions of the problem. It would then be possible to look for a range of compatible parameters with a given inverted plume height.

Other strategies could now be designed using hybrid techniques based on a first round of optimization based on volcano-semianalytical-seer and improvement based on volcano-lagrangian-seer. We could for example perform statistical inference using volcano-semianalytical-seer as forward model to determine a prior for inference using volcano-lagrangian-seer. We could as well find a starting point for an optimization algorithm using volcano-semianalytical-seer and refine the optimization using volcano-lagrangian-seer. We could finally investigate other metaheuristic optimization algorithms, some of them being applicable in parallel, such as the parallel simulated annealing. Then, the nested parallelization design integrated into ABCpy could prove useful to optimize a MPI parallelized model using a parallel metaheuristic.

5 Implementation of a multiscale TTDM

This chapter describes the multiscale implementation of a TTDM, including an aggregation process. The content of Section 5.4 has been originally published in

P. Künzli, J.-L. Falcone, E. Rossi, P. Albuquerque, and B. Chopard. "HPC Multiscale Simulation of Transport and Aggregation of Volcanic Particles". In: *2018 17th International Symposium on Parallel and Distributed Computing (ISPDC)*. 2018 17th International Symposium on Parallel and Distributed Computing (ISPDC). June 2018, pp. 25–32. DOI: 10.1109/ISPDC2018.2018.00013

and some implementation details present in Section 5.2 are found in the supplementary materials of

P. Künzli, K. Tsunematsu, P. Albuquerque, J.-L. Falcone, B. Chopard, and C. Bonadonna. "Parallel simulation of particle transport in an advection field applied to volcanic explosive eruptions". In: *Computers & Geosciences* 89 (2016), pp. 174–185. DOI: https://doi.org/10.1016/j.cageo.2016.02.005.

In the present chapter, we bring a stronger emphasis on implementation aspects.

5.1 Introduction

Real world phenomena involve processes of different natures, occurring at different spatial and temporal scales and involving different physical concepts. Each of those processes can be described by a model. The work of creating models to describe physical processes can be done with a large range of mathematical and computational tools, ranging from a simple equation to complex numerical models. Every model can rely on its own numerical description and physical concepts. By coupling models together, then considered as submodels, one can create a multiscale model, whose goal is to give an accurate representation of the real world phenomena considered. The concrete implementation of a multiscale model is a multiscale application.

Models for specific processes of a multiscale application are routinely published in specialized journals. If the design of the application is not thought since the beginning with flexibility in mind, it can be cumbersome to integrate a newly published model, and thus, make the multiscale model evolve with more up-to-date science difficult. When implementing a multiscale application, submodels can come in a variety of forms, usually a piece of software, either being a self-contained program such as a Python script, or implemented as a function or a class in a given language, for example in C++. The choice of the technology will depend on the model type and preferences and skills of the model developer. If part of the model is computationally intensive, it could be of interest to re-implement it with a different technology, for example on GPU, or by parallelizing it on a distributed memory architecture to target supercomputers.

It is then necessary to have a way to distinctively express the logic of data transfer between submodels, their concrete implementation in a computer language and their execution on a given infrastructure, which can range from a small microcontroller to the most powerful supercomputers coupled together.

Impact of volcanic ashes dispersal has been discussed previously in this document. Aggregation is an important physical phenomenon that impacts the sedimentation of volcanic particles [32]. Indeed, particle aggregation can modify particle sedimentation time. When particles stick together, they form larger objects that fall faster than the original ones. Not taking this phenomenon into account may lead to overestimate particle concentration in the atmosphere [29, 99]. For this reason, we coupled an aggregation model with the transport model.

When particles leave the volcanic plume, they are transported by wind, possibly over very long distances. Thus, meteorological data have to be integrated. We use data from the ERA-interim reanalysis provided by the ECMWF (European Centre for Medium-Range Weather Forecasts).

The aforementioned phenomena occur at various time and spatial scales: aggregation happens at scales of the order of the meter and the second; advection inside the plume tens of kilometres and hours to days; and advection in the atmosphere up to thousands of kilometres over weeks. To design such a multiscale application, we rely on the MMSF (Multiscale Modelling Software and Framework) set of tools and techniques. MMSF allow describing and simulating multiscale and multi-science phenomena in a standardized way [35, 37].

Tetras is based on a hybrid Eulerian-Lagrangian numerical scheme which leads to important workloads because it is necessary to simulate a huge number of particles to achieve statistical significance. The transport code is thus parallelized on a distributed memory architecture using MPI. Tetras's functionalities are implemented within the library `libpiaf` (Particles In Advection Field) which allows for representation of generic particles in an advection field and the library `libtetras`, which account for all the specificities of volcanic ash particles and related physical models. Tetras is implemented on top of those libraries. This allows for flexibility and gives the user the possibility to write its own simulation code.

Volcano-lagrangian-seer is based on a pure Lagrangian numerical model and also leads to important workloads. It has thus been parallelized as well using MPI. Data structures of `libpiaf` and `libtetras` have been reused to implement volcano-lagrangian-seer, but the core of the numerical model has been rewritten because it is fundamentally different from Tetras's numerical model. This implementation has been done through a header only library called `lagrangiantetras`, which is usable by custom client code. This architecture shows the flexibility of our approach and allows for hiding complex parallelism considerations that are embedded in `libpiaf` and `lagrangiantetras`.

For the multiscale implementation of the application, we rely on the MUSCLE-HPC coupling library [13], which is an HPC implementation of the MUSCLE coupling library [28, 26, 115]. This library allows implementing submodels in the form of C++ classes, with the MMSF operators being implemented as methods. The MMSF conduits are implemented as MPI communication, allowing good performances for tightly coupled submodels. We give a description of the multiscale implementation by building on the description of the single scale Tetras implementation.

Then, we look at the performance impact of coupling the particle transport at multiple scales and confirm that the impact of the usage of MUSCLE-HPC on the performances is low, and we show that using coarser time scale for the long-range transport allows for shorter computation time. Moreover, coupling multiple spatial domains requires a clever assignment to optimize computing resource usage. We thus discuss a strategy to model performances of submodels by a Discrete Event Simulation (DES) approach.

5.2 Tetras single scale implementation

This section describes the main data structures and algorithms used to implement the Lagrangian on grid method as well as the general structure of the single-scale implementation of Tetras. For each data structure, there is a partial representation of the full C++ implementation, where prototypes of the associated data and functions are given in C++ pseudo-code (for example `Array <Type>` is the C++ template notation and can be read, "Array of Type"). `Double2` and `Double3` are types that store together respectively two and three numbers. `MpiManager` that embeds and wraps MPI functionalities.

Code extracts are given in pseudo-C++. The syntax of C++ is not totally respected, and boilerplate code is neglected for simplicity. However, the logic of the objects construction and dependencies is respected, so that the code extract are close to code that is actually written. Functions members of classes are called "member functions" or "methods" in objected oriented programming, depending on the language. In C++, it is more common to use the term member function, while Java uses the term method. In this document, we choose to use the latter.

Listing 5.1 gives a simplified example of implementation of a particles advection code using

`libpiaf` and `libtetras`. It illustrates the use of the main data structures used to implement a TTDM. Listing 5.2 gives an example of parameter file that `FileManager` can read.

5.2.1 Particles

Particles (Listing 5.3) are described based on three main features: their family (specified by size and density, Listing 5.4), a position (relative to the site to which the particle is linked to) and their own velocity (the Lagrangian velocity). A multi-particle cellular automata method would only have stored the particle family, since velocity and position are directly attached to the domain. Volcano-lagrangian-seer uses a similar data structure, except that it considers the velocity and position to be relative to the domain's origin.

5.2.2 Velocities and advection field

Particles in the model are subject to an advection field. It consists of several velocities which are applied at points of the simulation space. Each of these velocities is described by a specific model. The parameters taken into account for the computation of a velocity are the position, the time, the time step and the particle family (or particle characteristics). The time step is useful to compute the random velocities used to generate diffusion.

The advection field is defined by a set of function providing components at any position, time, Δt (for diffusion) and particle characteristics. In object-oriented programming, we can define function objects, which are objects that we can call like a function. In C++, this is done by adding the operator `()` as a method of a class. We have created an abstract class called `SpeedFunctor` (functor is another name for function object in C++) with a virtual `()` operator taking as parameters : position, time, dt and particle family. Thus, we define the advection field by creating a set of function objects which will be used as advection field. It is then easy to modify its definition.

Each velocity of the model must then be defined as a child class of the abstract class `Velocity`. Then each velocity object is guaranteed to have an operator `()` with a known set of parameters, and can embed the necessary data structures.

Listing 5.5 shows the abstract structure `Velocity` and Listing 5.6 shows an example of the `TerminalVelocity` structures that inherits from `Velocity` which computes the terminal velocity of particles. It embeds a data structure of class `Atmosphere` which is used to get characteristics of the atmosphere (temperature, pressure, density) necessary to compute terminal velocity of particles.

Listing 5.1: Simplified example of a simulation code based on `libpiaf` and `libtetras`. In this example, we construct an advection field compounded of one constant velocity. To model the transport of volcanic particles, the task consist on the construction of a set of velocities that defines an advection field based on volcanological and atmospherical models. Those velocities captures the physics of the model.

```
class ConstantVelocity: Velocity{
  Atmosphere atmosphere
  Double3 operator()(
   Double3 position,
   double t,
   double dt ,
   TephraParticleFamily family
  ){
    return {1.0, 1.0, 1.0};
  }
}

main(){
  // Initialization
  MpiManager          mpiManager();
  FileManager         fm(mpiManager);
  EruptionParameters  ep          = fm.loadEruptionParameters("parameters.csv");
  Array<Velocity>     velocities  = [ConstantVelocity()];
  Domain              domain      (origin, size, dx, mpiManager);
  Terrain             terrain     (origin, size, dx, mpiManager);
  ParticleRepository  pr          (mpiManager);
  Simulator       simulator   (domain, dt, terrain, particleRepository,
                                mpiManager);

  domain.addParticleClasses       (ep.particleClasses())
  domain.addVelocities            (velocities)

  // Solving
  while(domain.containsParticles()){
      if(currentTime < eruptionTime) injectParticles();
      simulator.step();
      currentTime += dt;
  }

  // Finalization
  fm.write("output.h5", terrain);
}
```

Listing 5.2: An example of input file that `FileManager` of `libtetras` can read.

```
plumeModel,           degruyter
ventPosition,         0,          0,                    3000
# eruption events
#                     start time, duration
#                     8h30 ,      4h30
eruptionEvent,        30600,      16200
#                     15h,        2h
eruptionEvent,        54000,      7200
columnDiffusion,      1000
# particle classes
#                     phi,        wtpct,                density
family,               -8,         0.00000,              1100
family,               -7,         1.37562,              1100
family,               -6,         0.60811,              1100
...
family,               12,         0.00140,              2650
family,               13,         0.00020,              2650
family,               14,         0.00003,              2650
# plume ,             time,       file name
plume,                0,          plume-ruapehu-0.csv
# atmosphere
windModel,            files
tropopause,           16000
stratosphere,         24000
temperatureASL,       288
pressureASL,          101325
atmosphereDiffusion, 300
# wind
wind,                 0,          wind-ruapehu-0.csv
```

Listing 5.3: `Particle` data structure.

```
Particle :
 Data :
  int familyId
  Double3 displacement
  Double3 LagrangianSpeed
```

Listing 5.4: `TephraParticleFamily` data structure.

```
TephraParticleFamily :
 Data :
  int familyId
  double density
  double grainSize
  double weightPercent
  double diameter
  double mass
```

Listing 5.5: `Velocity` data structure.

```
Velocity :
 Function :
  Double3 operator()(
   Double3 position,
   double t,
   double dt ,
   TephraParticleFamily family
  )
```

Listing 5.6: `TerminalVelocity` data structure. The syntax `TerminalVelocity : Velocity` means that `TerminalVelocity` inherits from functions and data of `Velocity`.

```
TerminalVelocity : Velocity :
 Data :
  Atmosphere atmosphere
 Function :
  Double3 operator()(
   Double3 position,
   double t,
   double dt ,
   TephraParticleFamily family
  )
```

5.2.3 Domain

The simulation occurs inside a discretized domain (Listing 5.7) which is split into boxes of size Δx^3. Each particle remains linked to the closest site of the domain (the centre of each box). This way it is easy to track individual particles, to implement some aggregation function and to parallelize the implementation (by assigning groups of sites to different cores).

Each site in contact with the ground is tagged. Thus, if a particle enters a site in contact with the ground, the simulator transfers the particle to the terrain in order to be deposited. If a particle leaves the domain without entering a tagged site, the particle is transferred to a particle repository (described below).

Listing 5.7: `Domain` data structure.

```
Domain :
 Data :
  double dx
  Double3 position
  Array< ParticleFamily > particleFamilies
  Array< Array< Particle > > domain
  Array< Velocity > velocities
  Array< bool > isInGroundContact
  MpiManager mpiManager
```

Listing 5.8: `ParticleRepository` data structure.

```
ParticleRepository :
 Data :
  Array< Particle > storedParticles
  Array< ParticleFamily > particleFamilies
  MpiManager mpiManager
 Function :
  depositParticle( Particle )
```

5.2.4 Particle repository and terrain

A particle can leave the domain in two cases: either the particle is transported outside the domain, or the particle makes contact with the ground (the particle is deposited). Thus, we need a structure able to deal with these outgoing particles. This is the purpose of the particle repository and terrain structures.

A particle repository (Listing 5.8) is a structure that stores all particles that fall outside the domain. The terrain (Listing 5.9) is a substructure of the particle repository with the extra ability to identify the deposition position of the particles on the ground. When a particle is transferred to a particle repository (either a simple particle repository or a terrain), the particle inherits characteristics (position and velocity) from the domain. The particle then becomes a Lagrangian particle.

The repository will store the particle in its current state in a list. The main purpose is to verify the conservation of mass at the end of the simulation (each particle injected in the domain must be located in a repository / terrain or in the domain at the end of the simulation). This is also useful for the multiscale implementation, where it is necessary to intercept and store particles that leaves a simulation domain in order to send them to another one.

The terrain has the additional capability to compute where the particle will fall on the ground. It stores a digital terrain model as a square grid of elevations, where each element of the grid represents the elevation of a portion of the terrain. When a particle is transferred to the terrain, one must locate where the particle will fall on the ground. To this end, the terrain searches for the intersection between the particle trajectory and the terrain by moving the particle from a terrain portion to another, until the particle hits the ground. At this stage, the terrain stores the particle. So, at the end of the simulation, the terrain contains a list of Lagrangian particles deposited on the ground. There is also a data structure called *LightTerrain* that directly computes the deposition of particles on a grid in $kg \cdot m^{-2}$ to avoid storing individually all particles that deposit.

5.2.5 Simulator

The simulator (Listing 5.10) manages a domain in which particles are injected. These particles are transported within the domain by the advection field resulting from the various velocities.

Listing 5.9: `Terrain` data structure.

```
Terrain : ParticleRepository :
 Data :
  Array< Particle > storedParticles
  Array< ParticleFamily > particleFamilies
  double dx
  Double2 position
  Array< double > terrain
  MpiManager mpiManager
 Function :
  depositParticle( Particle )
```

Listing 5.10: `Simulator` data structure.

```
Simulator :
 Data :
  Domain domain
  double currentTime
  double dt
  Terrain terrain
  Repository repository
  MpiManager mpiManager
 Function :
  step()
```

The simulator takes into account the following entities: the domain, the time step Δt, the terrain in which the particles are deposited, a particle repository to store the other particles.

At each time step, the simulator computes the advection field and moves the particles accordingly. Each particle is moved individually and linked to a new site if necessary.

5.2.6 Parallel algorithm

Intercore communications

Each core works on a local subdomain which consists of a given number of blocks of the domain. This means that cores must send their neighbours the particles leaving their local blocks, and receive particles from these neighbours. Therefore, each core will prepare a packet of particles for each of its neighbours at each iteration (26 neighbours in three dimensions). These packets are then exchanged at the end of an iteration and particles are placed in the local domain of each core.

Parallel simulator

The parallel simulator applies the same algorithm as for the sequential simulator to each domain block. The behaviour of the parallel implementation differs mainly by two points. Firstly, there is the particle exchange step at the end of each iteration. Secondly, recall that the usage of an adaptive Δt implies the possibility that an iteration can be cancelled. However, in a

Listing 5.11: `SeerModel` data structure.

```
SeerModel :
 Data :
  VolcanicPlume plume
  Velocity plumeVelocity
  Velocity umbrellaCloudVelocity
  Array< ParticleFamily > particleFamilies
  function< double(SeerParticle, double) > plumeExitRate
  function< double(SeerParticle) > umbrellaCloudExitRate
  MpiManager mpiManager
 Function :
  move(SeerParticle&)
```

Listing 5.12: `SeerSimulator` data structure.

```
SeerSimulator :
 Data :
  double currentTime
  double dt
  Terrain terrain
  Double3 origin
  Double3 size
  SeerModel seerModel
  Array< ParticleFamily > particleFamilies
  Array< SeerParticle > particles
  Array< Velocity > velocities
  MpiManager mpiManager
 Function :
  step()
```

parallel context, each core needs to know the success state of the iteration of all cores. Indeed, a constraint violation can occur in any domain block. The success state being a boolean value, we need to perform a parallel logical "and" over each local value. Therefore, we have to perform a reduction operation and distribute the result to all the cores. This is done using the `MPI_AllReduce` primitive.

For the parallel implementation, the computation of an iteration is completed by each core. Then the result of the reduction allows to determine if this iteration can be validated or not.

5.3 Volcano-lagrangian-seer implementation

To implement volcano-lagrangian-seer, we have been able to reuse a large part of the useful data structures of Tetras : particles, velocities, file manager, eruption parameters management

Listing 5.13: `SeerParticle` data structure.

```
SeerParticle : Particle :
 Data :
  bool sourceTermParticle
```

and terrain. However, this model relies on a pure Lagrangian scheme and does not need to rely on an actual grid in memory to represent a domain. The simulator manages a list of Lagrangian particles. Thus, the parallelization is made simpler because there is no particle exchange implied between cores. Particles are equally distributed across cores when they are created and remain managed by the same core until they reach the ground. This comes at the price that particle interactions or computation of atmospheric ash load are not available and not easily doable with this model.

It has, however, been necessary to implement new components in addition to the new simulator. First, there is the `ExitRate` data structure. Indeed, we present in the Chapter 3 how exit rates are quantified, but as for the velocities, we decided to implement them in the form of functions that are exposed to the end user. Here, rather than defining a class with an abstract `()` operator as for velocities, we define exit rates (one is defined for the plume, another for the umbrella cloud) to be standard functions created through lambda expressions. Those expressions allow in C++ to construct unnamed functions that can be bound to variables. Moreover, they are able to capture variables within the scope at the creation of the function (i.e., they can embed data as a regular C++ object). The end result is close to what we obtain with the velocities : objects that encapsulate necessary data and that can be called as functions. This is illustrated on lines 21 and 27 of Listing 5.14.

The exit rates for plume takes as input a particle and an orientation, because the exit rate in the plume is dependent on exit orientation in the case of a weak plume (see Algorithm 3.2 in Chapter 3). For the umbrella cloud, however, only the particle is necessary (see Algorithm 3.1 in Chapter 3).

Then, those exit rates are passed to the `SeerModel` (Listing 5.11) data structure, which manages behaviour of particles inside the plume and umbrella cloud. This component embed as well a data structure describing the plume geometry and characteristics. In the current implementation, `SeerModel` does not manage directly particles. Instead, `SeerSimulator`(Listing 5.12) manage particles that can be tagged as lying inside the plume or the umbrella cloud or not (Listing 5.13). When particles exit the plume or umbrella cloud, they are subject to the advection field of the atmosphere until they reach the ground.

The usage of function objects for velocities and exit rates definitions allows exposing in the client code the physics of the model, without the need of modifying the underlying numerical model and parallelism. This way of doing thus increase the flexibility of the tool. This has been demonstrated by the possibility of implementing volcano-lagrangian-seer by reusing the definition of velocities from Tetras.

Listing 5.14: Summary of volcano-lagrangian-seer implementation. The syntax `[=]` in lines 21 and 27 means that variables used in the function are automatically captured by copy. That means that they are stored in the corresponding function object. The result is close to a function object as in the case of the velocities, but avoid the necessity of explicitly creating a class. The physics is mainly captured by velocities, as in Tetras, but it is necessary to pass a `VolcanicPlume` data structure to the model, because the simulator needs to know the geometry of the plume and umbrella cloud, which is not the case in Tetras.

```
1  class ConstantVelocity: Velocity{
2    Double3 operator()(
3     Double3 position,
4     double t,
5     double dt ,
6     TephraParticleFamily family
7    ){
8      return {1.0, 1.0, 1.0};
9    }
10 }
11
12 main(){
13   // Initialization
14   MpiManager          mpiManager();
15   FileManager         fm(mpiManager);
16   EruptionParameters ep         = fm.loadEruptionParameters("parameters.csv");
17   Array<Velocity>     velocities = [ConstantVelocity()];
18   Terrain             terrain      (origin, size, dx, mpiManager);
19   VolcanicPlume       plume        (ventPosition, wind, u0, r0, n0, t0);
20
21   function<double(SeerParticle, double)> plumeExitRate = [=](SeerParticle part,
22                                                              double direction){
23     // this function should return the probability
24     // for a particle to exit the plume at each second
25     return 1.0;
26   }
27   function<double(SeerParticle)> ucExitRate = [=](SeerParticle part){
28     // this function should return the probability
29     // for a particle to exit the umbrella cloud at each second
30     return 1.0;
31   }
32
33   SeerModel     seerModel(ep.particleClasses(), plume, plumeExitRate,
34                           ucExitRate, mpiManager);
35   SeerSimulator simulator(seerModel, velocities, origin, size, mpiManager);
36
37   // Solving
38   while(seerModel.containsParticles()){
39     if(currentTime < eruptionTime) injectParticles();
40     simulator.step();
41     currentTime += dt;
42   }
43
44   // Finalization
45   fm.write("output.h5", terrain);
46 }
```

5.4 A multiscale TTDM based on Tetras and MUSCLE-HPC

When particles are entrained inside the volcanic plume, they can stick together to form larger particles. Some of them are transported over a very long range by the wind. In the atmosphere, we suppose that particle concentration is too low so that their interaction can be neglected.

In the plume, speed of particles can be several orders of magnitude higher than when transported by the wind. Moreover, aggregation is a computationally intensive task, occurring at the time scale of a second. The full volcano simulation is therefore a multiscale process. A way to deal with it is to simulate each of the phenomena separately, and transfer data between them.

This problem is general to modelling science. This area of research aims at modelling more and more complex physical processes. One of the main paradigm that has emerged in the last decades to meet this challenge is multiscale modelling, where a physical process is viewed as a set of submodels, each one simulating a model at a specific time and temporal scale while exchanging data with other submodels. Various theoretical and operational multiscale frameworks have been proposed, [64] gives a review of those tools.

One approach is the Multiscale Modelling and Simulation Framework (MMSF) [27, 37], which defines a theoretical formulation of submodels and their data exchange and tools to implement the pairing and execution of submodels. A solution to implements a MMSF multiscale model is to rely on MUSCLE2 [26] which is a library that allows establishing communication channels between submodels written in Python, Java, C/C++ or Fortran, called conduits. This has the advantage of the wider compatibility, but the communication channels are TCP connections, which can prevent from using high bandwidth available on clusters. In this work, we rely on the MUSCLE-HPC framework [13]. This library is only compatible with C++ submodels, while submodels have to be implemented as C++ classes, but communications occurs through MPI communication. Which allows implementing multiscale models fully compliant with MMSF theoretical foundations while using the plain network capacity of modern clusters.

In the previous chapters, we have focused and described the transport processes of particles. In this section, we start by describing the aggregation model. Then, we describe how the multiscale application is implemented using MUSCLE-HPC as well as the implementation of the aggregation process.

5.4.1 Aggregation

Particle aggregation creates larger agglomerates starting from single and smaller objects. When they aggregate, volcanic particles tend to fall faster. Then, not taking this phenomenon into account can lead to the overestimation of atmospheric particle concentration, and underestimation of particle deposition close to the vent [32].

As a first step, we used an aggregation model inspired by water droplet aggregation. Note that

there is ongoing work to produce models more suited to volcanic particle aggregation [11]. The modular design of an MMSF application allows us to substitute submodels when more accurate ones are available.

The creation of a new aggregate depends on the capacity of particles to interact, i.e., to get physically in contact, and to stick together. In the present work, the theoretical description of particle aggregation is based on the general concepts contained in the so-called Population Balances Equation [96]. The underlying assumption is that there is a density function that describes the number of particles inside a population with a given selected property, "mass" in our case. This density function gives the number of particles with a mass in the interval $[m, m+dm]$. The equations that govern the interaction of droplets are the so-called Smoluchowski Coagulation Equations (SCE) [105]. This set of ordinary integro-differential equations describes the evolution in time of a population of particles of mass m in a given control volume:

$$\frac{dn(t,m)}{dt} = \frac{1}{2}\int_0^m K(t,m-\epsilon,\epsilon)n(t,m-\epsilon)n(t,\epsilon)d\epsilon - n(t,m)\int_0^\infty K(t,m,\epsilon)n(t,\epsilon)d\epsilon \quad (5.1)$$

The key quantities that appear in equation (5.1) are :

- the distribution $n(t,m)$ of the number of particles per unit volume with mass m at time t;
- the aggregation kernel $K(t,m,\epsilon)$, which contains all the information about collision rates and sticking efficiency for a two-particle collision of mass m and ϵ. According to the general description of an aggregation process, $K(t,m,\epsilon)$ is usually the product of the collision rate $\beta(t,m,\epsilon)$, i.e., the flow rate of particles colliding with the object under analysis, and the sticking efficiency $\alpha(t,m,\epsilon)$, i.e., the probability of a successful collision.

It is worth mentioning that Equation (5.1) assumes that objects are only classified according to their mass. This approach is sufficiently exhaustive to describe the interaction of water droplets, since the product of aggregation is modelled as a larger sphere with the total mass of the interacting droplets of the same density. However, since solid aggregates are more complex than droplets, a more detailed perspective will be adopted in a future improvement of the present model. The solution of Equation (5.1) for a realistic case is non-trivial and analytic solutions are only available for simple cases. We adopted here the Fixed-Pivot Technique [77] to obtain the numerical solution of the SCE. It is based on the discretization of the mass density function over a fixed binning. The higher the number of bins, the lower the numerical diffusion introduced by the discretization process.

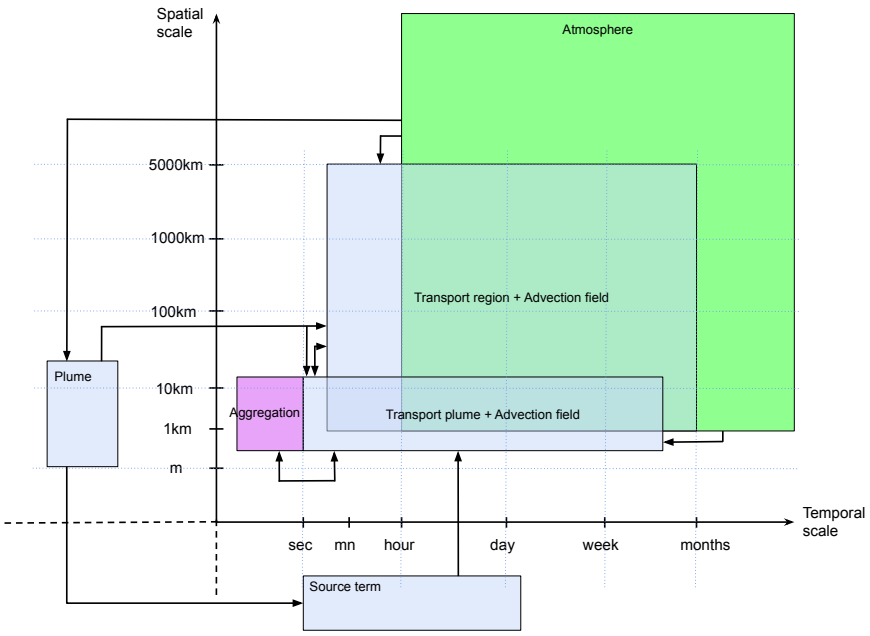
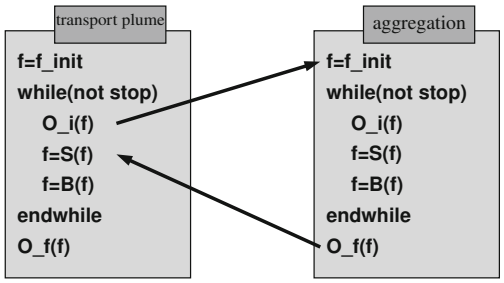
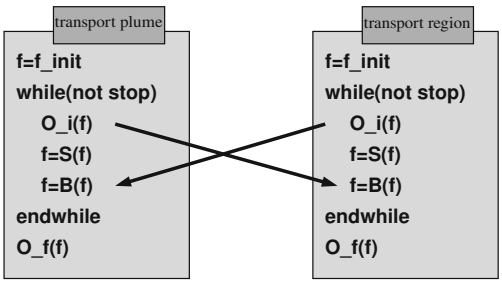


Figure 5.1: SSM of the multiscale tephra transport and aggregation application. The advection field submodels overlay transport submodels.



(a) Call-release between *transport plume* and *aggregation*



(b) Interact template between *transport plume* and *transport region*

Figure 5.2: Examples of two coupling templates.

5.4.2 Multiscale coupling

In our multiscale volcano application, compared to Tetras, we added the submodel *aggregation* and divided *transport* in two submodels *transport plume* and *transport region*. Which, in total, makes an application compound of the following submodels :

aggregation Computation of the transformation of a distribution of particles during a given time.

plume Compute the characteristics of the volcanic plume.

source term Manage the insertion of particles inside the simulation domain.

transport plume Transport of volcanic particles at the scale of the volcanic plume.

transport region Transport of particles over a long range by the wind.

atmosphere Representation of all useful characteristics of the atmosphere, such as wind velocity or atmospheric pressure.

advection field Definitions for the advection field.

With the following coupling templates :

plume* interaction with *advection fields is a dispatch coupling, where O_f of plume is connected to F_{init} of advection fields.

plume* interaction with *source term is a dispatch coupling.

source term* interaction with *transport plume is an interact coupling, where O_i of *source term* is connected to B of *transport plume*.

atmosphere* and *advection field *atmosphere* retrieves all the data from the ECMWF servers and provides them to *advection field*. O_f of *atmosphere* is used for F_{init} of *advection field*, which is referred to as a dispatch coupling.

advection fields* and *transport plume* or *transport region *advection field* computes at each iteration components of the advection field for *transport plume*. O_i of *transport plume* is used for F_{init} of *advection field*, which is named a call coupling. O_f of *advection field* is then used for S of *transport plume*, which is referred to as a release coupling. The situation is the same between *advection field* and *transport region*.

aggregation* and *transport plume *aggregation* computes a new distribution of particles for each site at each iteration of *transport plume*. It is again a call-release coupling.

transport plume* and *transport region *transport plume* sends particles and completion status at each iteration to *transport region*, and *transport region* sends completion status to *transport plume*. We thus have a so-called interact coupling template.

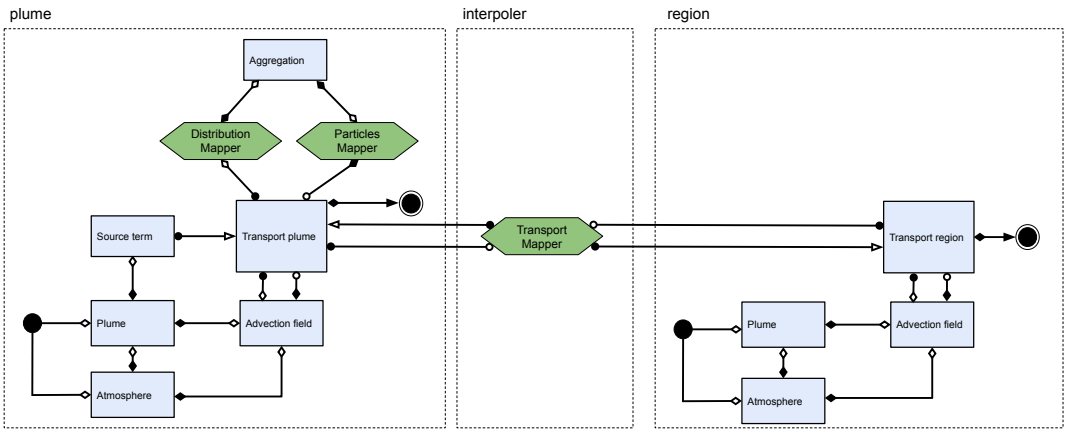


Figure 5.3: gMML representation of the full transport and aggregation multiscale application. When submodels are duplicated, that means that multiple instances are created. There are two model inputs and outputs because parameters are reads and results produced by two distinct processes. *Plume* submodel output is necessary for *advection field* of *transport region* because it is necessary to compute the advection field in the umbrella cloud.

Mappers are components which adapt data between submodels. In our case, we have three different mappers :

Distribution mapper Transforms a list of particles to a distribution of particles.

Particles mapper Transforms a distribution of particles to a list of particles.

Transport mapper Accumulates particles from *plume* in order to deal with different time step in *plume* and *region*.

Multiscale implementation

The next step is to actually couple the submodels. MUSCLE-HPC [13] allows us to couple submodels implemented as C++ classes with communication between submodels occurring through MPI. This allows a full use the network capacity of parallel machines.Transport submodels and mappers are implemented through MUSCLE-HPC API. Concretely, that means that submodels and mappers are C++ classes that inherit from the `MpiSubmodel` abstract class of MUSCLE-HPC. MMSF operators are defined as abstract methods `F_init()`, `S()`, `B()`, `Oi()`, `Of` and `isConverged()` of the class `MpiSubmodel` that the user should implement. The method `simulate()` will call them successively (Listing 5.15).

Listing 5.15: `simulate()` method of the `MpiSubmodel` class that implements the SEL. Operators are succesively executer until the model is converged. The user of the framework must define submodels as derived classes of the class `MpiSubmodel` and implements the operators `F_init()`, `S()`, `B()`, `Oi()`, `Of()` and `isConverged()`.

```
void MpiSubmodel::simulate(){
    F_init();
    while(!isConverged()){
        S();
        B();
        Oi();
    }
    Of();
}
```

For instance, in the case of transport submodels (*transport plume* and *transport region*) :

void F_init() Create the simulation domain and simulator data structures.

void S() Update advection field from *advection field* and move particles accordingly. For *plume* only: inject particles from *source term*, and apply aggregation (send particles to the distribution mapper, receive new particles from the particles mapper).

void B() Receive new particles from transport mapper for *transport region*.

void Oi() Send leaving particles to transport mapper for *transport plume*. Write state of the domain on disk if needed.

void Of() Write particles deposited on the ground and state of the domain on disk if needed.

boolean isConverged() Check if local (*transport plume*) and remote (*transport region*) convergences are reached.

Then, the method `simulate()` will call them successively, following the SEL model. The Listings 5.17 and 5.16 shows a partial implementation of the multiscale application. The class `MpiSubmodel` also provides the following methods :

```
ConduitEntrance<DataType> getConduitEntrance<DataType>(String);
ConduitExit<DataType>     getConduitExit<DataType>(String);
```

They are used to retrieve conduit entrances and exits from their names, which should match those defined in the coupling file (Listing 5.18).

The role of the transport mapper is to couple both transport models with different time steps. Indeed, Δt of *transport region* can be larger than that of *transport plume*. To achieve that, the mapper accumulates particles leaving *transport plume* in a buffer and move them to the appropriate physical time when requested by *transport region*. When particles are leaving *transport plume*, they store their velocity. This allows the mapper to do its job.

Listing 5.16: `f_ou t` is a conduit entrance because it has to be seen as the entrance of a pipe in which we write. The opposit applies to `f_in`, which has to be seen as the exit of a pipe from which we read. We use the methods `getConduitEntrance()` of the `MpiSubmodel` class that allows to retrieve conduit connector corresponding to the given name. There is also a function `getConduitExit()` that is used to retrieve the output of a conduit.

```
class TransportPlume: MpiSubmodel{

  bool isConverged(){
    return isLocalConverged && isRemoteConverged;
  }

  F_init(){
    // Initialization
    ep         = fm->loadEruptionParameters("parameters.csv");
    velocities = [ConstantVelocity()];
    domain = new Domain(origin, size, dx, mpiManager);
    terrain = new Terrain(origin, size, dx, mpiManager);
    simulator = new Simulator(domain, dt, terrain, particleRepository,
                                    mpiManager);
    domain->addParticleClasses       (ep.particleClasses());
    domain->addVelocities            (velocities);

    // get conduit entrance and exit
    f_out = getConduitEntrance<char>("f_out");
    f_in = getConduitEntrance<char>("f_in");
  }

  S(){
    // Solving
    // With optional aggregation step
    if(currentTime < eruptionTime) injectParticles();
    if(aggregate) domain->aggregate();
    simulator->step();
    currentTime += dt;
  }

  B(){
    // update local convergence and receive remote convergence
    if(!domain->containsParticles()) isLocalConverged = true;
    auto res = f_in->read();
    isRemoteConverged = deserialize(res);
  }

  Oi(){
    // send particles exiting the domain and local convergence
    f_out->write(serialize(pr.getParticles(), isLocalConverged));
  }

  Of(){
    // write the result on disk
    fm->write("output_plume.h5", terrain);
  }

}
```

Listing 5.17: Implementation of the multiscale volcano application with three MUSCLE-HPC submodels. Classes `TransportPlume`, `TransportRegion` and `TransportInterpolation` have to implement the methods `F_init()`, `S()`, `B()`, `Oi()` and `Of()` to respect the `MpiSubmodel` interface.

```
main(){
  MMSF_MPI mmsf();

  //--------- create kernels ---------------------
  MpiSubmodel * plume = new TransportPlume();
  MpiSubmodel * region = new TransportRegion();
  MpiSubmodel * interpoler = new TransportInterpolation();

  //--------- register kernels -------------------
  mmsf.addSubmodel(plume, "plume");
  mmsf.addSubmodel(region, "region");
  mmsf.addSubmodel(interpoler, "interpoler");

  //--------- setup conduits connections ----------
  mmsf.loadCouplingFile();

  //--------- start simulation -------------------
  mmsf.compute();

  delete plume, continent, interpoler;
}
```

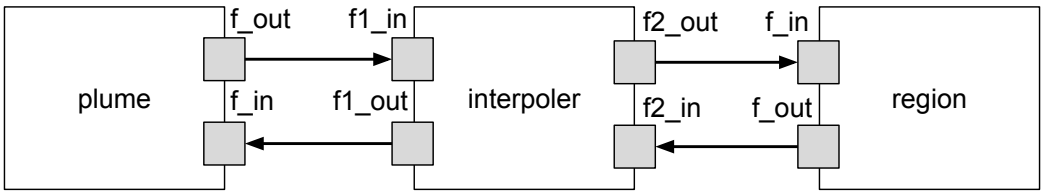


Figure 5.4: MUSCLE-HPC coupling.

Transport domain is split into volumes called sites and the aggregation model describes the evolution in time of populations of particles in a control volume. We then apply aggregation to each site of *transport plume* at each iteration.

In the MMSF terminology, ports are interconnected by conduits, which have to be defined in a configuration file specifying the coupling topology, number of cores and command line parameters for each submodel. An example of coupling file for the multiscale volcanic particle transport model is given in listing 5.18. The `conduit` instructions allow us to connect ports of submodels, the `cores` instructions provide the number of cores allocated to each submodel and the `cmdline` ones enable passing command line parameters to submodels. The resulting submodels, ports and conduits are depicted in Figure 5.4.

Listing 5.18: Example of MUSCLE-HPC coupling file.

```
conduit1<char>: plume.f_out -> interpoler.f1_in
conduit2<char>: region.f_out -> interpoler.f2_in
conduit3<char>: interpoler.f1_out -> plume.f_in
conduit4<char>: interpoler.f2_out -> region.f_in
cores<plume>: 6
cores<region>: 24
cores<interpoler>:1
cmdline<plume>:-x 30000 -y 30000 ...
cmdline<region>:-x 100000 -y 100000 ...
cmdline<interpoler>:
```

MUSCLE-HPC creates a process that acts as manager of the submodels to which other models have to connect. All submodel must not necessarily run within the same MPI runtime, but it is necessary to run a manager to which submodels will connect. It is thus necessary to allocate one more process than the actual sum of cores indicated in the coupling file. The MUSCLE-HPC multiscale model depicted in Listing 5.17 would be run in the following way with the configuration file of Listing 5.18 by supposing that the executable is named `volcano` and the configuration file `couplingfile` :

```
mpirun -np 32 volcano --coupling-file couplingfile --main --all
```

The parameter `-main` means that the manager process is run within this MPI execution and `-all` that all submodels are run. While there are 31 cores requested for the submodels in the configuration file, it is necessary to run 32 MPI processes.

Aggregation

While some submodels are coupled using specifically designed API like MUSCLE-HPC, some coupling templates do not necessarily imply using such an API, and are even better implemented by simple function calls. This is the case of the coupling between *transport plume* and *aggregation*, as they share the same spatial domain. *aggregation* works on a distribution of particles, while *transport plume* works on point particles. It is then necessary to convert from one representation to the other. This is why we placed mappers between *transport plume* and *aggregation*.

aggregation is a Fortran code, taking a distribution of particles as input, and giving another distribution as output. We have thus integrated this Fortran code inside a C++ wrapper function taking the role of the mapper which converts from particles to distribution and the other way around. To convert from particles to distribution, we count the number of particles of each class in a site. To produce particles from the distribution, we try to keep existing particles before injecting particles at random places in the site if new particles have to be created.

The implementation of the aggregation model used in this work tends to produce output particle distributions with slightly less mass than the input. Then, to avoid losing mass with

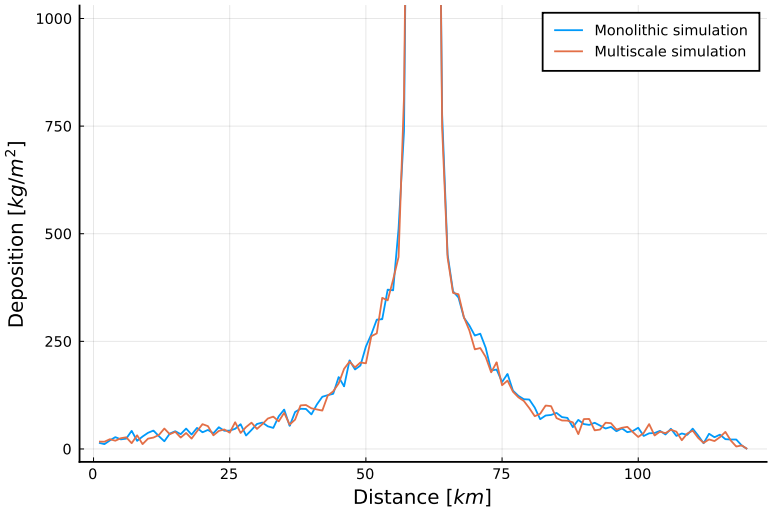


Figure 5.5: Comparison of simulation results between a monolithic and a multiscale simulation of a hypothetical volcanic eruption. In the monolithic case, simulation domain is of size $60km \times 60km \times 30km$ with $\Delta t = 1s$. In the multiscale simulation, *transport plume* simulate particles in a domain of size $10km \times 10km \times 20km$ with $\Delta t = 1s$. When particles exit *transport plume*, they are transferred to *transport region* where $\Delta t = 3s$ and of size $60km \times 60km \times 30km$.

multiple aggregation steps, the wrapper scales the output distribution to match the input mass.

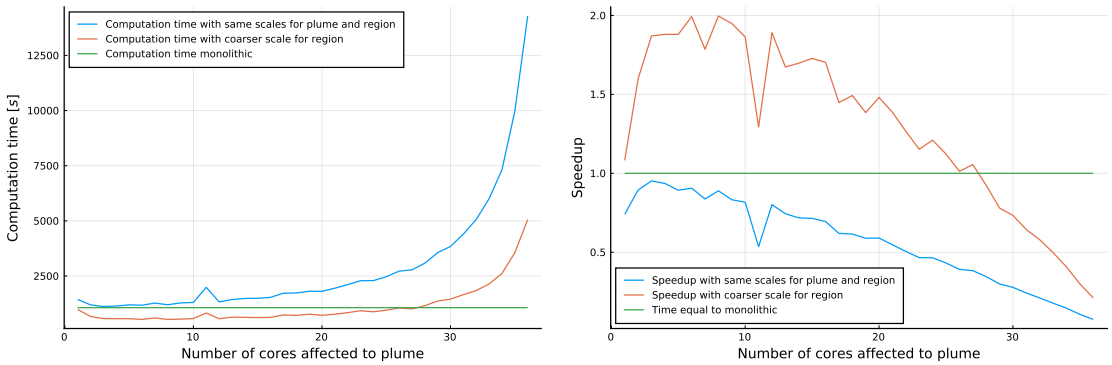
5.4.3 Submodels placement

Once the MMSF application designed, its submodels developed and possibly parallelized, the question of the placement of the submodels on available computing resources remains.

Let's consider the two transport submodels which are tightly coupled by the interact template (*transport plume* and *transport region*). For example, suppose that the Δt of both submodels are the same. One has to choose the allocation of resources so that one iteration of *plume* takes the same time as one iteration of *transport region*. Now suppose that Δt of *transport plume* is $1s$ and Δt of *transport region* is $3s$. One now has to choose the allocation of resources so that when *transport region* executes one iteration, *transport plume* executes three.

Simulations are done with a hypothetical eruption and we do not compare the results to field deposits. Simulations are done with a vent located at $2500m$, a total mass of particles injected of $4.5 \times 10^{11} kg$, and plume parameters $U_0 = 200m \cdot s^{-1}, L_0 = 50m, n_0 = 0.01, T_0 = 1200K$ diffusion in the volcanic plume of $500m^2 \cdot s^{-1}$ and diffusion in the atmosphere of $200m^2 \cdot s^{-1}$. All performance evaluations of this section have been performed on the Yggdrasil cluster of the Geneva's higher education institutions.

First, to validate that the multiscale implementation does not have a huge impact on simulation results, we ran a monolithic simulation as a reference with a domain of size $60km \times 60km \times 30km$ with $\Delta t = 1s$ and we compare with results obtained when *transport plume* simulate a domain of size $10km \times 10km \times 20km$ with $\Delta t = 1s$ and *transport region* simulate a domain of size $60km \times 60km \times 30km$ with $\Delta t = 3s$. Figure 5.5 shows a comparison between the two simulations. This allows to conclude that the change in the observed deposition is minimal.



(a) (b)

Figure 5.6: Comparison of execution time and speedup between monolithic, multiscale with same time scale in *transport plume* and *transport region* and multiscale with separate time scales in *transport plume* and *transport region*. The total number of allocated cores is 40, one is dedicated to MUSCLE-HPC manager, one to the transport mapper and the 38 remaining to submodels *transport plume* and *transport region*. The horizontal axis indicates the number of cores allocated to *transport plume*. Thus, on the left, one core is allocated to *transport plume* and 37 to *transport region*, while on the right 37 cores are allocated to *transport plume* and one to *transport region*.

Then, we observed the effect of the multiscale coupling on performances. For this, we first ran a multiscale simulation where *transport plume* and *transport region* simulates domains respectively of size $10km \times 10km \times 20km$ and $60km \times 60km \times 30km$ with both $\Delta t = 1s$ and $\Delta x = 500m$ and vary the allocation of cores from one core to *transport plume* and 37 cores to *transport region* to 37 cores to *transport plume* and one core to *transport region*. We then did the same thing, but with $\Delta t = 1s$ for *transport plume* and $\Delta t = 3s$ for *transport region*. It is possible to choose a higher Δt for *transport region* because the speed of particles is lower in this area.

The obtained execution times and speedup compared to the monolithic simulation are shown in Figure 5.6. First, we can conclude that the impact of the multiscale coupling on performance is low, because with the appropriate placement of submodels, the simulation with identical Δt in both submodels is achieved in almost the same time as the monolithic simulation. Then, while the particles in the *transport region* submodel deposits faster when Δt is higher, it is possible to achieve best performances with separates Δt. The speedup is approximately

two in the best case, when the number of cores assigned to plume is between 6 and 8. This demonstrates the importance of a clever choice of submodels placement.

It is important to note that this performance gain should be considered in a wider context. Here, there is a factor 3 between values of Δt in the two transport subdomains. It could be possible to further differentiate the subdomains, for example by having a subdomain dedicated to the transport in the plume, one to medium range transport and one to long range transport. This would allow for a Δt value more than an order of magnitude higher in the long range transport subdomain compared to the plume transport subdomain. This would allow for largely increased performance gains compared to a monolithic implementation.

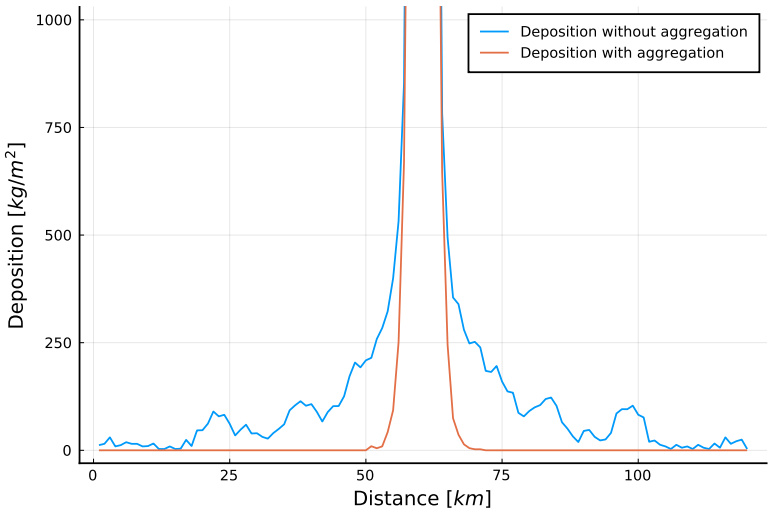


Figure 5.7: Comparisons of multiscale simulations results where aggregation is enabled or not. *transport plume* (where aggregation occurs) is of size $5km \times 5km \times 5km$ with $\Delta x = 100m$ and $\Delta t = 0.2s$ while *transport region* is of size $60km \times 60km \times 30km$ with $\Delta x = 500m$ and $\Delta t = 1s$.

Then, we ran the simulation with *transport plume* of size $5km \times 5km \times 5km$ with $\Delta x = 100m$ and $\Delta t = 0.2s$ and *transport region* is of size $60km \times 60km \times 30km$ with $\Delta x = 500m$ and $\Delta t = 2s$ with aggregation enabled in *transport plume*. Simulation results are shown in Figure 5.7. The aggregation model has not been carefully calibrated for volcanic ash aggregation, but we observe that particles deposit much closer to the vent location, which is compatible with smaller particles aggregation into larger ones, and the total observed mass on the ground match the injected mass. Which means that there is no mass loss resulting from the coupling.

Figure 5.8 shows the obtained execution time with aggregation enabled by varying the number of cores allocated to *transport plume* and *transport region* as previously. We observe that the overall execution time is lower than previously, even if applying aggregation model implies a much higher computational load per iteration. This is because larger particles falls faster. Then, we observe that the ideal number of cores allocated to *transport plume* is again around 10 cores, even with a higher computational load per iteration for the *transport plume* submodel.

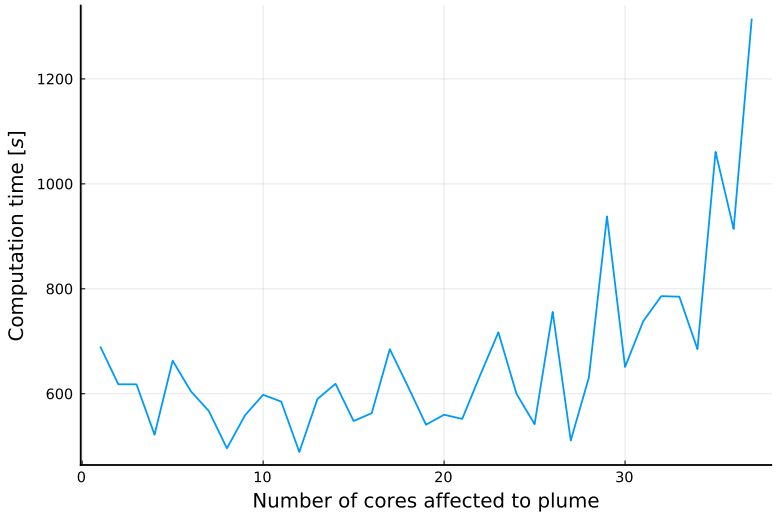
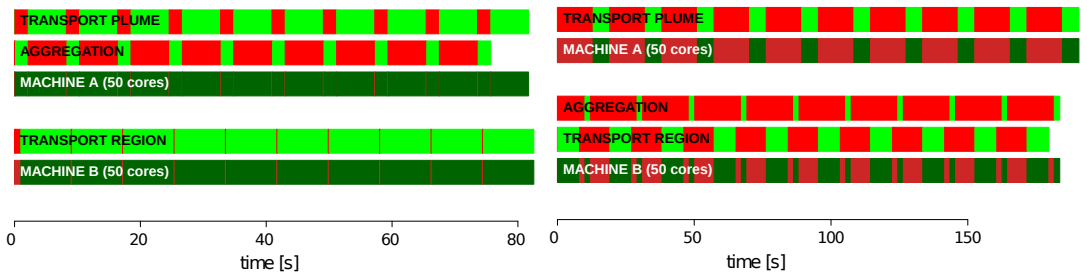


Figure 5.8: Comparison of execution times for a multiscale simulation where aggregation is enabled inside *transport plume*. The total number of allocated cores is 40, one is dedicated to MUSCLE-HPC manager, one to the transport mapper and the 38 remaining to submodels *transport plume* and *transport region*. The horizontal axis indicates the number of cores allocated to *transport plume*. Thus, on the left, one core is allocated to *transport plume* and 37 to *transport region*, while on the right 37 cores are allocated to *transport plume* and one to *transport region*. Note that the time is not compared to a monolithic run because the current implementation does not support application of aggregation in the plume only with a monolithic code.



(a) Placement with *plume* and *aggregation* on different machines

(b) Placement with *plume* and *aggregation* on the same machine

Figure 5.9: Comparison of two hypothetical placements of *plume*, *region* and *aggregation* on two 50-core machines. Red segments indicate idle CPU, green segment indicate useful computation.

This could be attributed to the fact that parallelism in *transport plume* is lower than in *transport region* due to the smaller domain and particles being spatially concentrated in a specific area, making useless to allocate too many cores to *transport plume*.

5.4.4 Performance analysis through Discrete Event Simulation

As we saw, allocation of submodels to available resources is a crucial question. It is common in the HPC community to provide performance analysis and performance models for parallel codes, it is then possible to predict execution time of each submodels and then optimize their placement. We designed a simulator based on Discrete Event Simulation (DES) which aims to evaluate submodel placement on computing resources.

This DES technique allows us to simulate the interaction between submodels and thus the execution time of a full multiscale application. The submodels are represented by state machines, whose states represent the current step inside the submodel execution loop (see Figure 5.2). We can predict the time of each step of a submodel by relying on a performance model and a model of the hardware on which the submodel runs, or with statistical measures of the execution time of the submodel on a specific hardware. This approach can also be used to evaluate the placement of submodels on multiple heterogeneous machines.

In Figure 5.9, we illustrate two placements of the three main submodels (*transport plume*, *transport region* and *aggregation*) when using two 50-core machines. At each iteration, *transport plume* must send data from all its domain to *aggregation* and *aggregation* must be executed entirely before *transport plume* can continue, as required by the call-release coupling. Here, we suppose that Δt of *transport plume* is the same as Δt of *transport region*. Thus, *transport region* must wait for *transport plume* and *aggregation* to finish at each iteration.

Due to the tight coupling and the call-release template between *transport plume* and *aggregation* which implies that only one of the two submodels can run at any given time, it is natural

to assume that allocating the same computing resources to both submodels is a good option. We compare this situation with a hypothetical situation where *aggregation* is placed on the same machine as *transport region*. In that case, we see that performances will be greatly limited. Communication time will limit performances and idle CPU time will increase. We can see that increasing the number of cores only for *transport region* or for both *transport plume* and *aggregation* will not reduce computation time because the other submodel will limit the performance.

5.5 Conclusion

In this chapter, we gave a detailed description of the main data structures implied in the implementation of Tetras and volcano-lagrangian-seer. We have shown that the adopted software architecture based on generic libraries and function objects allows for a good reusability of the code, while it has been possible to reuse and easily adapt the code defining the physics of Tetras to develop volcano-lagrangian-seer, developing only a new underlying parallel numerical model.

The dispersal of volcanic ashes during eruptions is a phenomenon involving multiple physical processes which occurs at multiple temporal and spatial scales. We can decompose such a phenomenon into multiple submodels. MMSF proposes solutions for the modelling, the implementation, and the coupling of this type of multiscale system.

The submodels needing large computational effort can benefit from parallelization in order to solve larger problems in a reasonable time. This is the case of our transport model, which has been parallelized on distributed memory architecture using MPI.

We focused on the design and implementation of the multiscale aspect of the application and showed how this task has been achieved following the MMSF methodology and using the MUSCLE-HPC API. We showed that the implementation of our multiscale application using MUSCLE-HPC has a low impact on performances, and that the adaptation of the scales of the transport models allow for increased performances. Then, we addressed the problem of submodel allocation on computing resources through a DES strategy. We gave as well the description of the aggregation model and shown some simulation results enabling this aggregation process.

Future work would be necessary to improve the DES simulation tool and verify its prediction ability with various simulation cases. Finally, improvements on the physical submodels will have to be achieved. Thanks to the modular design of the application at each level, this can be done smoothly without interacting with the development of other components of the model.

6 Conclusions and perspectives

Volcanic eruptions are dramatic large-scale events that can endanger human life or have a long-range impact due to the transport of fine volcanic ash. Their dynamics can be described by various physical models, possibly coupled together, such as models describing the dynamic of volcanic plumes, fall speed of particles or the ability of particles to stick together. It is, however, very difficult to get accurate observations of those phenomena, due to the extreme conditions in which they occur, their unpredictability and their interaction with meteorological phenomena. It is thus particularly interesting to design a simulation tool for such phenomenon with a clear decomposition in terms of components working at distinct scales coupled together. This allows updating specific parts of the multiscale model when research progress is made.

Another problem is the ability to reconstruct eruption characteristics (namely ESP, eruption source parameters) from external observations, such as the deposit pattern of particles on the ground. This task is called inversion of eruption parameters, and is done by running a large number of simulations with different sets of parameters and applying a clever strategy to update those parameter sets until finding the best possible solution.

In this thesis, we addressed those problems by designing our volcanic simulation tools as multiscale applications, coupling different submodels using a well-defined formalism. We developed parallel numerical models usable for inversion strategies, as well as a nested parallelization scheme which allows integrating MPI parallel applications in parallel optimization algorithms.

6.1 Summary and contributions

In Chapter 2 we presented Tetras (TEphra TRAnsport Simulator), a tephra transport simulator based on a hybrid Lagrangian-on-grid numerical model that applies an advection-diffusion process to point particles. We introduced the multiscale MMSF formalism and used it to describe the architecture of Tetras and described the volcanological models on which we rely for this description. A theoretical description of the performance of Tetras is also given in the

form of a performance model. The strong points of Tetras are its modular design, its underlying numerical model allowing for fast computation of atmospheric concentration, tracking of individual particles and implementation of particle interaction and a clever underlying parallelism. We identified a drawback in the model, in the fact that proximal deposition tends to be underestimated under the plume corner with the pure advection-diffusion approach. This led us to reevaluate the description of the dynamic of the particles in this area.

We ended up with a new way of describing the source term of the advection-diffusion process by quantifying the flux of particles exiting the plume and umbrella cloud and considering a 3D representation of those objects. We called this model the SEER (Spatially Extended Exit Rate) model and result in a new category of tephra transport models based on this source term. By solving analytically a simplified geometry of the model, we raised the question of the continuity of the rate of particle sedimentation under the plume corner. We also developed a simplified semi-analytical version of the model able to produce results for eruptions occurring in no wind condition with a fast resolution time called volcano-semianalytical-seer. We developed as well a full Lagrangian version of a tephra transport model based on SEER that we called volcano-lagrangian-seer. This allowed us to demonstrate the modularity of the strategy adopted to develop Tetras, where the physics is mainly embedded in a series of function objects which describes the velocity of particles. We were able to reuse the physical definition of Tetras and develop a new numerical model with a new parallelization scheme. We studied empirically the scaling of the application and observed a good strong scaling behaviour. This is important because it makes the usage of volcano-lagrangian-seer usable together with non-parallel optimization algorithms in order to do inversion.

In Chapter 4, we addressed the problem of ESP inversion. First, we did so by coupling Tetras with the ABCpy approximate Bayesian computation package. ABCpy runs a high number of forward models and refine approximations using ABC iterative algorithms. The package is parallelized using Spark or MPI. While Tetras is parallelized using MPI, it has been necessary to develop a nested parallelization scheme. This work has made possible the integration of any MPI model into ABCpy. Then, we made some preliminary investigation by coupling volcano-semianalytical-seer with the Nelder-Mead simplex algorithm. Indeed, the fast-solving time of volcano-semianalytical-seer allows integrating it in a sequential optimization strategy.

Finally, in Chapter 5 we described in more details the implementation of Tetras and volcano-semianalytical-seer. We gave the description of the full multiscale volcano application using MMSF and implements it using MUSCLE-HPC. We demonstrated the importance of submodel placement on computing resources and showed that the multiscale aspect of the implementation allows optimizing resource usage by varying the scale of submodels and allocate the appropriate quantity of resources. We addressed the question of submodel placement with a Discrete Event Simulation (DES) strategy.

6.2 Perspectives

A variety of directions could be explored following this work.

- We showed in Chapter 3 that particular conditions of particles velocities and exit rates have to be met to guarantee a continuous deposition pattern between particles sedimenting from the plume and from the umbrella cloud in the absence of diffusion. It would be interesting to investigate if this condition is actually met in reality, or if atmospheric diffusion prevent from observing a discontinuity on the field. It is, however, difficult to answer this question, while explosive eruptions produce extreme conditions, it is not possible to measure directly the rate of sedimentation of particles from the plume border and umbrella cloud base.

- Our multiscale volcano application uses a Lagrangian framework in the plume and applies an Eulerian aggregation model that works on particle distributions of equal density. We could compare our strategy to alternative approaches such as [11] where an aggregation model is coupled with a BPT plume model, itself used as a source term of the NAME transport model (with the insertion of particles at the top of the plume) or with other integral plume models that integrates aggregation such as [60] and [86]. We recall as well that we can update the aggregation model when models better suited to volcanic particles are available thanks to our modular design.

- The modular multiscale design of our application allows replacing submodels without changing the entire implementation. While there exist now plumes and umbrella cloud models that account for particle aggregation [60, 86], we could use those models as the source term of our numerical transport models (Lagrangian-on-grid and Lagrangian) using the SEER framework. That is to say, injecting particles in the transport model from all over the 3D surface defined by the plume and umbrella cloud models with specific insertion rates defined as well by the plume and umbrella cloud models.

- While a Lagrangian framework is interesting for transport at short and medium range because it allows tracking individual particles, simulating enough particles at long range to global scale can become computationally very intensive. It could then be interesting to couple our Lagrangian or semi Lagrangian transport model with an Eulerian transport model for long-range transport, by considering the Lagrangian transport as a source term of the Eulerian transport.

- Volcano-lagrangian-seer is a Lagrangian model parallelized on CPU using MPI. It is possible to parallelize this model on GPU. A difficulty would be to evenly attribute particles to GPU threads all along the simulation to maintain a good device occupancy. This could be done by attributing groups of particles of different density and sizes and injected at various time points to GPU threads. If good performance is achieved, this would allow performing fast forward simulations on GPU and inversion on GPU or GPU cluster if a parallel inference or optimization scheme is used.

- Volcano-semianalytical-seer does not include advection of particles by wind, and can currently only be used for a strong plume eruption in no wind condition, which is quite uncommon. It would be possible to add a fast numerical resolution if the wind is stratified and constant in time.

- Some development effort can be made to increase the flexibility and usability of our tools. For example, the usage of a digital elevation model could be added as a submodel to automate integration of topography from sources such as the ASTER global digital elevation model of the USGS [63] and the atmosphere submodel adapted to act as a wrapper of various reanalysis databases such as the NCEP/NCAR from the NOAA [75] or ERA5 from ECMWF [69, 15], or even from weather prediction models.

- The presented DES approach to model performances is still at a proof a concept stage, more development to produce a usable tool and validation with actual multiscale simulations implementations are needed.

- Various investigations can be made regarding inversion. Inversion within this work is done on plume model parameters, such as the radius, velocity and temperature at vent. It is possible to produce a plume with a given height with multiple combination of those parameters (for example, keeping the same plume height by decreasing the radius and increasing the temperature). We should then conduct a sensibility analysis to determine if highly different deposition patterns can be obtained with a given plume height and various plume parameters. If not, then it would make more sense to consider the plume height as a parameter and obtain initial velocity, temperature and radius through the Monte Carlo inversion described in Chapter 4. This would dramatically decrease the size of the parameter space to explore and consequently make the problem much easier. A study of the parameter space should as well be conducted to determine if multiple local optimum exists. Then, various optimization algorithms could be investigated to determine which ones have the best convergence behaviour for this problem. Hybrid strategies could also be adopted, such as a first exploration of the parameter space with a fast solving forward model before conducting more in depth local optimization using a heavy forward model.

- Finally, due to the flexible structure of our implementation and its ability to deal with generic types of particles and source term, we could explore the possibility to model transport of other types of passive particles, such as pollen. We could still benefit from the parallel implementation and generic design whilst changing the target application. Modelling of active particles, such as radionuclides, would require the adaptation of the particles data structure and the addition of processes currently not integrated in the model.

A The advection-diffusion process with Lagrangian particles

A.1 Diffusion

This section shows how random speeds used to generate diffusion are calculated.

We know from the central limit theorem that a random walk produces a Gaussian distribution with variance

$$\sigma^2 = \frac{t}{\Delta t}\epsilon^2 \tag{A.1}$$

where t is the elapsed time since the beginning of the random walk, ϵ the size of a step and Δt the time step. In our case we have

$$\epsilon = u_r \Delta t \tag{A.2}$$

Thus

$$\sigma^2 = u_r^2 \Delta t \cdot t \tag{A.3}$$

According to the Einstein relation, diffusion can be written

$$D = \frac{\sigma^2(t)}{2d \cdot t} \tag{A.4}$$

where d is the number of dimensions considered. We can now write the associated random speed

$$u_r = \sqrt{\frac{2d \cdot D}{\Delta t}} \tag{A.5}$$

We finally select with uniform probability a direction for the random velocities. In the one

dimensional case (1D) we have

$$\begin{aligned} r &= 1 \text{ or } -1 \\ \vec{u}_r &= u_r \cdot r \end{aligned} \tag{A.6}$$

where r is chosen randomly with probability 0.5. Then in the two dimensional case (2D) we have

$$\begin{aligned} &\theta \in [0; 2\pi] \\ &\vec{u}_r = u_r \begin{pmatrix} \cos(\theta) \\ \sin(\theta) \end{pmatrix} \end{aligned} \tag{A.7}$$

where θ is drawn with uniform probability. And in the three dimensional case (3D) we have

$$\begin{aligned} &\theta \in [0; 2\pi] \\ &\phi = \arccos(2r - 1) \text{ with } r \in [0; 1] \\ &\vec{u}_r = u_r \begin{pmatrix} \cos(\theta)\sin(\phi) \\ \sin(\theta)\sin(\phi) \\ \cos(\phi) \end{pmatrix} \end{aligned} \tag{A.8}$$

where θ and r are drawn with uniform probability. Then, we can choose to generate an isotropic diffusion in 3D, or to combine a 2D and a 1D diffusion in order to produce different diffusions horizontally and vertically.

A.2 Validation of the advection-diffusion process

In order to validate the advection-diffusion process using the simulator, we inject N particles in a domain at position $\vec{r}_0 = (x_0, y_0, z_0)$ at time t_0. We then apply a constant velocity field $\vec{v}_0 = (v_x, v_y, v_z)$ during a time t with a diffusion coefficient D. We compare the simulation results with the analytical solution of the diffusion equation. The particle distribution can be expressed as

$$n(x, y, z, t) = \frac{N}{[4\pi(t - t_0)]^{3/2}\sqrt{D_x D_y D_z}} \cdot \exp\left[-\frac{\|\vec{\xi}\|^2}{4}\right] \tag{A.9}$$

where $\vec{\xi} = \left(\frac{x - x_0 - v_x(t-t_0)}{\sqrt{D_x(t-t_0)}}, \frac{y - y_0 - v_y(t-t_0)}{\sqrt{D_y(t-t_0)}}, \frac{z - z_0 - v_z(t-t_0)}{\sqrt{D_z(t-t_0)}}\right)$

We perform a simulation with $N = 10^6$, $t_0 = 0[tu]$, $t = 10[tu]$, $\vec{r}_0 = (0,0,0)[lu]$, $\vec{v} = (1,1,1)[\frac{lu}{tu}]$, $D = 0.5[\frac{lu^2}{tu}]$, $\Delta t = 0.1[tu]$ [1]. We recall that diffusion arises from the random velocities $\vec{u}_r$ applied to each particle at each time step as given by Equation (A.8) for the three dimensional case.

[1] tu and lu stand respectively for time unit and length unit

Then we compare the simulated particle distribution to the analytical distribution at time $t = 10[tu]$ along on a line through $(10,10,10)[lu]$ parallel to the x-axis. This comparison is shown on Figure A.1. We observe a very good correspondence between experimental results and the analytical solution (the same holds along any line). Figure A.2 shows a horizontal slice through the middle of the simulated particle distribution at time $t = 10[tu]$, illustrating that our diffusion is isotropic as expected.

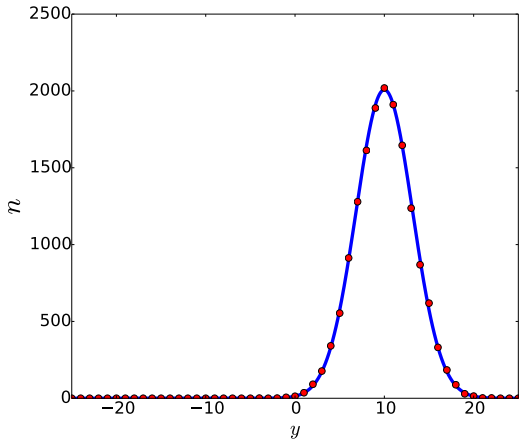


Figure A.1: The simulated particle distribution (red dots) compared to the analytical distribution (blue line) at time $t = 10[tu]$ along on a line through $(10,10,10)[lu]$ parallel to the y-axis.

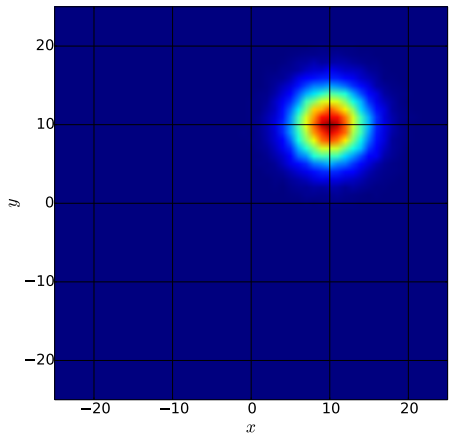


Figure A.2: Horizontal cut of the simulated particle distribution at time $t = 10[tu]$ with a plane through $(10,10,10)[lu]$ parallel to the xy-plane.

A.3 Number of particles

A dedicated sensitivity analysis was performed to optimize the number of particles used in the simulations. In fact, the total number of particles corresponding to the erupted mass and the particle characteristics of Ruapehu and Pululagua eruptions would be of the order of magnitude of 10^{22} to 10^{23} particles. Current computers cannot deal with such a number of particles injected in the simulator.

We therefore resorted to a statistical sampling approach. We injected a much smaller number of particles, and we then scaled the output to get the correct number of particles.

An obvious question is how many particles should we inject in order to obtain accurate results ? In our case, we are interested in the number of particles deposited on the ground. Our output is stored in the form of a matrix containing the mass of particles per area.

As the generated diffusion is a stochastic process, outputs differ from each other. However, for a given physical configuration, the dispersion of the simulation results is relatively small provided the sample consists of enough particles.

We use the procedure described in [103] to determine the number of particles to use. We run s simulations with n particles. To estimate the similarity between two outputs p and q ($p, q \in \{1, \ldots, s\}$), we compute the correlation coefficient

$$\rho_{pq}^{(n)} = \frac{cov(R_p, R_q)}{\sigma_p \sigma_q} = \frac{E\{(R_p - \mu_p)(R_q - \mu_q)\}}{\sigma_p \sigma_q} \tag{A.10}$$

where R is a vector containing the output matrix, μ the mean value and σ the standard deviation of R. There are $s(s-1)/2$ distinct pairs of results. We consider the average

$$\bar{\rho}^{(n)} = \frac{\sum_{p<q}^{s} \rho_{pq}^{(n)}}{s(s-1)/2} \tag{A.11}$$

which is a measure of the relative stability of the results.

For a given number of outputs, we observe the number of particles n necessary for $\bar{\rho}^{(n)}$ to be greater than a given threshold (e.g., ≥ 0.99). Beyond this threshold, we consider the outputs to be stable. It is important to note that this accuracy estimation greatly depends on the physical configuration of the simulation (size of the domain, physical model applied and its parametrization), it is then not possible to determine the required number of particles before running simulations.

For each n with values in the range 1800 to $18 \cdot 10^6$ particles, we ran $s = 20$ simulations. We computed the $s(s-1)/2 = 190$ values $\rho_{pq}^{(n)}$ and then their average $\bar{\rho}^{(n)}$. The value of $\bar{\rho}^{(n)}$ as a function of n is shown on Figure A.3.

We see that a 0.99 correlation is reached with slightly more than 10^6 particles (red dot on Figure A.3) for the Pululagua simulation. The same order of magnitude holds for Ruapehu. We chose to run our simulations with 10^6 particles per family, which gives a total of $18 \cdot 10^6$ particles with 18 particle families for 2450BP Pululagua simulations and $23 \cdot 10^6$ particles for 1996 Ruapehu simulations.

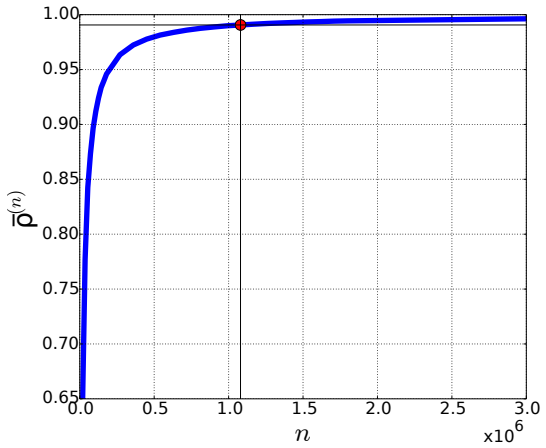


Figure A.3: Mean correlations between simulation outputs for given numbers of injected particles for the Pululagua simulation.

Bibliography

[1] C. Albert, R. K. Hans, and A. Scheidegger. “A simulated annealing approach to approximate Bayesian computations”. In: *Statistics and Computing* 25 (2015), pp. 1217–1232.

[2] G. M. Amdahl. “Validity of the single processor approach to achieving large scale computing capabilities”. In: *Proceedings of the April 18-20, 1967, spring joint computer conference*. AFIPS ’67 (Spring). New York, NY, USA: Association for Computing Machinery, Apr. 18, 1967, pp. 483–485. DOI: 10.1145/1465482.1465560.

[3] G. Bagheri and C. Bonadonna. “Chapter 2 - Aerodynamics of Volcanic Particles: Characterization of Size, Shape, and Settling Velocity”. In: *Volcanic Ash*. Ed. by S. Mackie, K. Cashman, H. Ricketts, A. Rust, and M. Watson. Elsevier, 2016, pp. 39–52. DOI: 10.1016/B978-0-08-100405-0.00005-7.

[4] G. H. Bagheri, C. Bonadonna, I. Manzella, P. Pontelandolfo, and P. Haas. “Dedicated vertical wind tunnel for the study of sedimentation of non-spherical particles”. In: *Review of Scientific Instruments* 84.5 (2013), p. 054501. DOI: 10.1063/1.4805019.

[5] F. Barberi, M. Coltelli, A. Frullani, M. Rosi, and E. Almeida. “Chronology and dispersal characteristics of recently (last 5000 years) erupted tephra of Cotopaxi (Ecuador): implications for long-term eruptive forecasting”. In: *Journal of Volcanology and Geothermal Research* 69.3 (1995), pp. 217–239. DOI: https://doi.org/10.1016/0377-0273(95)00017-8.

[6] S. Barsotti and A. Neri. “The VOL-CALPUFF model for atmospheric ash dispersal: 2. Application to the weak Mount Etna plume of July 2001”. In: *Journal of Geophysical Research: Solid Earth* 113 (B3 Mar. 2008), B03209. DOI: 10.1029/2006JB004624.

[7] S. Barsotti, A. Neri, and J. S. Scire. “The VOL-CALPUFF model for atmospheric ash dispersal: 1. Approach and physical formulation”. In: *Journal of Geophysical Research: Solid Earth* 113 (B3 Mar. 2008), B03208. DOI: 10.1029/2006JB004623.

[8] M. A. Beaumont. “Approximate Bayesian computation in evolution and ecology”. In: *Annual review of ecology, evolution, and systematics* 41 (2010). Publisher: Annual Reviews, pp. 379–406.

[9] D. M. Beazley. “Automated scientific software scripting with SWIG”. In: *Future Generation Computer Systems* 19.5 (July 1, 2003), pp. 599–609. DOI: 10.1016/S0167-739X(02)00171-1.

[10] D. M. Beazley et al. “SWIG: An Easy to Use Tool for Integrating Scripting Languages with C and C++.” In: *Tcl/Tk Workshop*. Vol. 43. 1996, p. 74.

[11] F. Beckett, E. Rossi, B. Devenish, C. Witham, and C. Bonadonna. “(submitted) Modelling the size distribution of aggregated volcanic ash and implications for operational atmospheric dispersion modelling”. In: *Atmospheric Chemistry and Physics Discussions* (May 31, 2021). Publisher: Copernicus GmbH, pp. 1–39. DOI: 10.5194/acp-2021-254.

[12] F. M. Beckett, C. S. Witham, S. J. Leadbetter, R. Crocker, H. N. Webster, M. C. Hort, A. R. Jones, B. J. Devenish, and D. J. Thomson. “Atmospheric Dispersion Modelling at the London VAAC: A Review of Developments since the 2010 Eyjafjallajökull Volcano Ash Cloud”. In: *Atmosphere* 11.4 (Apr. 2020). Number: 4 Publisher: Multidisciplinary Digital Publishing Institute, p. 352. DOI: 10.3390/atmos11040352.

[13] M. B. Belgacem and B. Chopard. “MUSCLE-HPC: A new high performance API to couple multiscale parallel applications”. In: *Future Generation Computer Systems* 67 (2017), pp. 72–82. DOI: https://doi.org/10.1016/j.future.2016.08.009.

[14] M. B. Belgacem, B. Chopard, J. Borgdorff, M. Mamoński, K. Rycerz, and D. Harezlak. “Distributed Multiscale Computations Using the MAPPER Framework”. In: *Procedia Computer Science*. 2013 International Conference on Computational Science 18 (Jan. 1, 2013), pp. 1106–1115. DOI: 10.1016/j.procs.2013.05.276.

[15] B. Bell, H. Hersbach, P. Berrisford, P. Dahlgren, A. Horányi, J. Muñoz Sabater, J. Nicolas, R. Radu, D. Schepers, A. Simmons, C. Soci, and J.-N. Thépaut. *ERA5 hourly data on single levels from 1950 to 1978 (preliminary version)*. Copernicus Climate Change Service (C3S) Climate Data Store (CDS). URL: https://cds.climate.copernicus.eu/cdsapp#!/dataset/reanalysis-era5-single-levels-preliminary-back-extension?tab=overview (visited on 05/12/2021).

[16] E. Bernton, P. E. Jacob, M. Gerber, and C. P. Robert. “Approximate Bayesian computation with the Wasserstein distance”. In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 81.2 (2019). Publisher: Wiley Online Library, pp. 235–269.

[17] S. Biass and C. Bonadonna. “A quantitative uncertainty assessment of eruptive parameters derived from tephra deposits: the example of two large eruptions of Cotopaxi volcano, Ecuador”. In: *Bulletin of Volcanology* 73.1 (Jan. 1, 2011), pp. 73–90. DOI: 10.1007/s00445-010-0404-5.

[18] M. G. Blum, M. A. Nunes, D. Prangle, S. A. Sisson, et al. “A comparative review of dimension reduction methods in approximate Bayesian computation”. In: *Statistical Science* 28.2 (2013), pp. 189–208.

[19] C. Bonadonna and B. F. Houghton. “Total grain-size distribution and volume of tephra-fall deposits”. In: *Bulletin of Volcanology* 67.5 (Jan. 2005), pp. 441–456. DOI: 10.1007/s00445-004-0386-2.

[20] C. Bonadonna, J. C. Phillips, and B. F. Houghton. “Modeling tephra sedimentation from a Ruapehu weak plume eruption”. In: *Journal of Geophysical Research: Solid Earth* 110 (B8 Aug. 2005), B08209. DOI: 10.1029/2004JB003515.

[21] C. Bonadonna, M. Pistolesi, R. Cioni, W. Degruyter, M. Elissondo, and V. Baumann. "Dynamics of wind-affected volcanic plumes: The example of the 2011 Cordón Caulle eruption, Chile". In: *Journal of Geophysical Research: Solid Earth* 120.4 (Apr. 2015), 2014JB011478. DOI: 10.1002/2014JB011478.

[22] C. Bonadonna, L. J. Connor, C. B. Connor, and L. M. Courtland. *Tephra2*. 2014.

[23] C. Bonadonna and A. Costa. "Modeling tephra sedimentation from volcanic plumes". In: *Modeling Volcanic Processes*. Cambridge University Press, 2013. DOI: 10.1017/CBO9781139021562.009.

[24] C. Bonadonna, A. Folch, S. Loughlin, and H. Puempel. "Future developments in modelling and monitoring of volcanic ash clouds: outcomes from the first IAVCEI-WMO workshop on Ash Dispersal Forecast and Civil Aviation". In: *Bulletin of Volcanology* 74.1 (Aug. 2011), pp. 1–10. DOI: 10.1007/s00445-011-0508-6.

[25] C. Bonadonna and J. C. Phillips. "Sedimentation from strong volcanic plumes". In: *Journal of Geophysical Research: Solid Earth* 108 (B7 July 2003), p. 2340. DOI: 10.1029/2002JB002034.

[26] J. Borgdorff, M. Mamonski, B. Bosak, K. Kurowski, M. B. Belgacem, B. Chopard, D. Groen, P. V. Coveney, and A. G. Hoekstra. "Distributed multiscale computing with MUSCLE 2, the Multiscale Coupling Library and Environment". In: *Journal of Computational Science* 5.5 (2014), pp. 719–731. DOI: https://doi.org/10.1016/j.jocs.2014.04.004.

[27] J. Borgdorff, J.-L. Falcone, E. Lorenz, C. Bona-Casas, B. Chopard, and A. G. Hoekstra. "Foundations of distributed multiscale computing: Formalization, specification, and analysis". In: *Journal of Parallel and Distributed Computing* 73.4 (2013), pp. 465–483. DOI: https://doi.org/10.1016/j.jpdc.2012.12.011.

[28] J. Borgdorff, M. Mamonski, B. Bosak, D. Groen, M. B. Belgacem, K. Kurowski, and A. G. Hoekstra. "Multiscale Computing with the Multiscale Modeling Library and Runtime Environment". In: *Procedia Computer Science*. 2013 International Conference on Computational Science 18 (Jan. 1, 2013), pp. 1097–1105. DOI: 10.1016/j.procs.2013.05.275.

[29] R. J. Brown, C. Bonadonna, and A. J. Durant. "A review of volcanic ash aggregation". In: *Physics and Chemistry of the Earth, Parts A/B/C*. Volcanic ash: an agent in Earth systems 45–46 (2012), pp. 65–78. DOI: 10.1016/j.pce.2011.11.001.

[30] M. I. Bursik, R. S. J. Sparks, J. S. Gilbert, and S. N. Carey. "Sedimentation of tephra by volcanic plumes: I. Theory and its comparison with a study of the Fogo A plinian deposit, Sao Miguel (Azores)". In: *Bulletin of Volcanology* 54.4 (Apr. 1, 1992), pp. 329–344. DOI: 10.1007/BF00301486.

[31] S. Carey and R. S. J. Sparks. "Quantitative models of the fallout and dispersal of tephra from volcanic eruption columns". In: *Bulletin of Volcanology* 48.2 (June 1986), pp. 109–125. DOI: 10.1007/BF01046546.

[32] S. N. Carey and H. Sigurdsson. “Influence of particle aggregation on deposition of distal tephra from the MAy 18, 1980, eruption of Mount St. Helens volcano”. In: *Journal of Geophysical Research: Solid Earth* 87 (B8 1982), pp. 7061–7072. DOI: http://dx.doi.org/10.1029/JB087iB08p07061.

[33] T. J. Casadevall. *Volcanic ash and aviation safety; proceedings of the First international symposium on Volcanic ash and aviation safety*. USGS Numbered Series 2047. Series: U.S. Geological Survey Bulletin. U.S. G.P.O., 1994.

[34] M. Cerminara, T. Esposti Ongaro, and L. C. Berselli. “ASHEE-1.0: a compressible, equilibrium-Eulerian model for volcanic ash plumes”. In: *Geoscientific Model Development* 9.2 (Feb. 18, 2016). Publisher: Copernicus GmbH, pp. 697–730. DOI: 10.5194/gmd-9-697-2016.

[35] B. Chopard, J. Borgdorff, and A. G. Hoekstra. “A framework for multi-scale modelling”. In: *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 372.2021 (2014). Publisher: The Royal Society _eprint: http://rsta.royalsocietypublishing.org/content/372/2021/20130378.full.pdf. DOI: 10.1098/rsta.2013.0378.

[36] B. Chopard and M. Droz. *Cellular Automata Modeling of Physical Systems*. Cambridge: Cambridge University Press, 1998. DOI: 10.1017/CBO9780511549755.

[37] B. Chopard, J.-L. Falcone, P. Kunzli, L. Veen, and A. Hoekstra. “Multiscale modeling: recent progress and open questions”. In: *Multiscale and Multidisciplinary Modeling, Experiments and Design* 1.1 (2018), pp. 57–68. DOI: 10.1007/s41939-017-0006-4.

[38] L. J. Connor and C. B. Connor. “Inversion is the key to dispersion: understanding eruption dynamics by inverting tephra fallout”. In: *Statistics in Volcanology*. Geological Society of London, 2006. DOI: 10.1144/IAVCEI001.18.

[39] L. Connor, C. Connor, L. Courtland, and A. Saballos. *Tephra2 Users Manual*. 2011.

[40] R. Constantinescu, A. Hopulele-Gligor, C. B. Connor, C. Bonadonna, L. J. Connor, J. M. Lindsay, S. Charbonnier, and A. C. M. Volentik. “The radius of the umbrella cloud helps characterize large explosive volcanic eruptions”. In: *Communications Earth & Environment* 2.1 (Jan. 2021), p. 3. DOI: 10.1038/s43247-020-00078-3.

[41] A. Costa, G. Macedonio, and A. Folch. “A three-dimensional Eulerian model for transport and deposition of volcanic ashes”. In: *Earth and Planetary Science Letters* 241.3 (2006), pp. 634–647. DOI: 10.1016/j.epsl.2005.11.019.

[42] A. Costa, A. Folch, and G. Macedonio. “A model for wet aggregation of ash particles in volcanic plumes and clouds: 1. Theoretical formulation”. In: *Journal of Geophysical Research: Solid Earth* 115 (B9 Sept. 2010), B09201. DOI: 10.1029/2009JB007175.

[43] A. Costa, A. Folch, and G. Macedonio. “Density-driven transport in the umbrella region of volcanic clouds: Implications for tephra dispersion models”. In: *Geophysical Research Letters* 40.18 (Sept. 2013), pp. 4823–4827. DOI: 10.1002/grl.50942.

[44] K. Csilléry, M. G. Blum, O. E. Gaggiotti, and O. François. “Approximate Bayesian computation (ABC) in practice”. In: *Trends in Ecology & Evolution* 25.7 (2010), pp. 410–418.

[45] G. A. Davidson. “Gaussian versus top-hat profile assumptions in integral plume models”. In: *Atmospheric Environment (1967)* 20.3 (1986), pp. 471–478. DOI: https://doi.org/10.1016/0004-6981(86)90087-9.

[46] J. Dean and S. Ghemawat. “MapReduce: Simplified Data Processing on Large Clusters”. In: *Communications of the ACM* 51.1 (2008). Publisher: ACM, pp. 107–113.

[47] D. P. Dee, S. M. Uppala, A. J. Simmons, P. Berrisford, P. Poli, S. Kobayashi, U. Andrae, M. A. Balmaseda, G. Balsamo, P. Bauer, P. Bechtold, A. C. M. Beljaars, L. van de Berg, J. Bidlot, N. Bormann, C. Delsol, R. Dragani, M. Fuentes, A. J. Geer, L. Haimberger, S. B. Healy, H. Hersbach, E. V. Hólm, L. Isaksen, P. Kållberg, M. Köhler, M. Matricardi, A. P. McNally, B. M. Monge-Sanz, J.-J. Morcrette, B.-K. Park, C. Peubey, P. de Rosnay, C. Tavolato, J.-N. Thépaut, and F. Vitart. “The ERA-Interim reanalysis: configuration and performance of the data assimilation system”. In: *Quarterly Journal of the Royal Meteorological Society* 137.656 (2011). Publisher: John Wiley & Sons, Ltd., pp. 553–597. DOI: 10.1002/qj.828.

[48] W. Degruyter and C. Bonadonna. “Impact of wind on the condition for column collapse of volcanic plumes”. In: *Earth and Planetary Science Letters* 377–378 (Sept. 2013), pp. 218–226. DOI: 10.1016/j.epsl.2013.06.041.

[49] W. Degruyter and C. Bonadonna. “Improving on mass flow rate estimates of volcanic eruptions”. In: *Geophysical Research Letters* 39.16 (Aug. 2012), p. L16308. DOI: 10.1029/2012GL052566.

[50] P. Del Moral, A. Doucet, and A. Jasra. “An adaptive sequential Monte Carlo method for approximate Bayesian computation”. In: *Statistics and Computing* 22.5 (2012). Publisher: Springer, pp. 1009–1020.

[51] X. Didelot, R. G. Everitt, A. M. Johansen, D. J. Lawson, et al. “Likelihood-free estimation of model evidence”. In: *Bayesian Analysis* 6.1 (2011), pp. 49–76.

[52] C. C. Drovandi and A. N. Pettitt. “Estimation of Parameters for Macroparasite Population Evolution Using Approximate Bayesian Computation”. In: *Biometrics* 67.1 (2011). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1541-0420.2010.01410.x, pp. 225–233. DOI: 10.1111/j.1541-0420.2010.01410.x.

[53] T. Dürig, M. T. Gudmundsson, F. Dioguardi, M. Woodhouse, H. Björnsson, S. Barsotti, T. Witt, and T. R. Walter. “REFIR- A multi-parameter system for near real-time estimates of plume-height and mass eruption rate during explosive eruptions”. In: *Journal of Volcanology and Geothermal Research* 360 (July 1, 2018), pp. 61–83. DOI: 10.1016/j.jvolgeores.2018.07.003.

[54] R. Dutta, M. Schoengens, J. Onnela, and A. Mira. “ABCpy: A user-friendly, extensible, and parallel library for Approximate Bayesian Computation”. In: *Proceedings of the Platform for Advanced Scientific Computing Conference*. ACM, June 2017.

[55] R. Dutta, M. Schoengens, L. Pacchiardi, A. Ummadisingu, N. Widmer, P. Künzli, J.-P. Onnela, and A. Mira. "ABCpy: A High-Performance Computing Perspective to Approximate Bayesian Computation". In: *arXiv:1711.04694 [stat]* (Feb. 25, 2021).

[56] J.-L. Falcone, B. Chopard, and A. Hoekstra. "MML: towards a Multiscale Modeling Language". In: *Procedia Computer Science*. ICCS 2010 1.1 (May 1, 2010), pp. 819–826. DOI: 10.1016/j.procs.2010.04.089.

[57] P. Fearnhead and D. Prangle. "Constructing Summary Statistics for Approximate Bayesian Computation: Semi-Automatic Approximate Bayesian Computation". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 74.3 (2012). Publisher: Wiley Online Library, pp. 419–474.

[58] A. Folch. "A review of tephra transport and dispersal models: Evolution, current status, and future perspectives". In: *Journal of Volcanology and Geothermal Research* 235–236 (Aug. 2012), pp. 96–115. DOI: 10.1016/j.jvolgeores.2012.05.020.

[59] A. Folch, A. Costa, and G. Macedonio. "FALL3D: A computational model for transport and deposition of volcanic ash". In: *Computers & Geosciences* 35.6 (June 2009), pp. 1334–1342. DOI: 10.1016/j.cageo.2008.08.008.

[60] A. Folch, A. Costa, and G. Macedonio. "FPLUME-1.0: An integral volcanic plume model accounting for ash aggregation". In: *Geoscientific Model Development* 9.1 (2016), pp. 431–450. DOI: 10.5194/gmd-9-431-2016.

[61] A. Folch, L. Mingari, N. Gutierrez, M. Hanzich, G. Macedonio, and A. Costa. "FALL3D-8.0: a computational model for atmospheric transport and deposition of particles, aerosols and radionuclides – Part 1: Model physics and numerics". In: *Geoscientific Model Development* 13.3 (2020), pp. 1431–1458. DOI: 10.5194/gmd-13-1431-2020.

[62] W. Ge. "Deep metric learning with hierarchical triplet loss". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 269–285.

[63] D. Gesch, M. Oimoen, J. Danielson, and D. Meyer. "VALIDATION OF THE ASTER GLOBAL DIGITAL ELEVATION MODEL VERSION 3 OVER THE CONTERMINOUS UNITED STATES". In: *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. XXIII ISPRS Congress, Commission IV (Volume XLI-B4) - 12–19 July 2016, Prague, Czech Republic. Vol. XLI-B4. ISSN: 1682-1750. Copernicus GmbH, June 13, 2016, pp. 143–148. DOI: 10.5194/isprs-archives-XLI-B4-143-2016.

[64] D. Groen, J. Knap, P. Neumann, D. Suleimenova, L. Veen, and K. Leiter. "Mastering the scales: a survey on the benefits of multiscale computing software". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 377.2142 (Apr. 8, 2019). Publisher: Royal Society, p. 20180147. DOI: 10.1098/rsta.2018.0147.

[65] M. T. Gudmundsson, R. Pedersen, K. Vogfjörd, B. Thorbjarnardóttir, S. Jakobsdóttir, and M. J. Roberts. "Eruptions of Eyjafjallajökull Volcano, Iceland". In: *Eos, Transactions American Geophysical Union* 91.21 (May 2010), pp. 190–191. DOI: 10.1029/2010EO210002.

[66] M. Guffanti, G. C. Mayberry, T. J. Casadevall, and R. Wunderman. "Volcanic hazards to airports". In: *Natural Hazards* 51.2 (June 2008), pp. 287–302. DOI: 10.1007/s11069-008-9254-2.

[67] M. U. Gutmann, R. Dutta, S. Kaski, and J. Corander. "Likelihood-free inference via classification". In: *Statistics and Computing* 28.2 (2018). Publisher: Springer, pp. 411–425.

[68] R. Hadsell, S. Chopra, and Y. LeCun. "Dimensionality reduction by learning an invariant mapping". In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. Vol. 2. IEEE, 2006, pp. 1735–1742.

[69] H. Hersbach, B. Bell, P. Berrisford, G. Biavati, A. Horányi, J. Muñoz Sabater, J. Nicolas, C. Peubey, R. Radu, I. Rozum, D. Schepers, A. Simmons, C. Soci, D. Dee, and J.-N. Thépaut. *ERA5 hourly data on single levels from 1979 to present*. Copernicus Climate Change Service (C3S) Climate Data Store (CDS). URL: https://doi.org/10.24381/cds.adbb2d47 (visited on 05/12/2021).

[70] M. Herzog and H.-F. Graf. "Applying the three-dimensional model ATHAM to volcanic plumes: Dynamic of large co-ignimbrite eruptions and associated injection heights for volcanic gases". In: *Geophysical Research Letters* 37.19 (2010). _eprint: https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/2010GL044986. DOI: https://doi.org/10.1029/2010GL044986.

[71] M. Herzog, J. M. Oberhuber, and H.-F. Graf. "A Prognostic Turbulence Scheme for the Nonhydrostatic Plume Model ATHAM". In: *Journal of the Atmospheric Sciences* 60.22 (Nov. 1, 2003). Publisher: American Meteorological Society Section: Journal of the Atmospheric Sciences, pp. 2783–2796. DOI: 10.1175/1520-0469(2003)060<2783:APTSFT>2.0.CO;2.

[72] R. E. Holasek and S. Self. "GOES weather satellite observations and measurements of the May 18, 1980, Mount St. Helens eruption". In: *Journal of Geophysical Research: Solid Earth* 100 (B5 May 1995), pp. 8469–8487. DOI: 10.1029/94JB03137.

[73] C. J. Horwell and P. J. Baxter. "The respiratory health hazards of volcanic ash: a review for volcanic risk mitigation". In: *Bulletin of Volcanology* 69.1 (Apr. 2006), pp. 1–24. DOI: 10.1007/s00445-006-0052-y.

[74] A. Jones, D. Thomson, M. Hort, and B. Devenish. "The U.K. Met Office's Next-Generation Atmospheric Dispersion Model, NAME III". In: *Air Pollution Modeling and Its Application XVII*. Ed. by C. Borrego and A.-L. Norman. Springer US, 2007, pp. 580–589. DOI: 10.1007/978-0-387-68854-1_62.

[75] E. Kalnay, M. Kanamitsu, R. Kistler, W. Collins, D. Deaven, L. Gandin, M. Iredell, S. Saha, G. White, J. Woollen, Y. Zhu, A. Leetmaa, R. Reynolds, M. Chelliah, W. Ebisuzaki, W. Higgins, J. Janowiak, K. C. Mo, C. Ropelewski, J. Wang, R. Jenne, and D. Joseph. "The NCEP/NCAR 40-Year Reanalysis Project". In: (Mar. 1, 1996), p. 124.

[76] W. C. Krumbein. "Size frequency distributions of sediments". In: *Journal of Sedimentary Research* 4.2 (Aug. 1, 1934). Publisher: GeoScienceWorld, pp. 65–77. DOI: 10.1306/D4268EB9-2B26-11D7-8648000102C1865D.

[77] S. Kumar and D. Ramkrishna. "On the solution of population balance equations by discretization—I. A fixed pivot technique". In: *Chemical Engineering Science* 51.8 (1996), pp. 1311–1332. DOI: https://doi.org/10.1016/0009-2509(96)88489-2.

[78] D. Kunii and O. Levenspiel. *Fluidization Engineering*. Butterworth-Heinemann, 1991.

[79] P. Künzli, J.-L. Falcone, E. Rossi, P. Albuquerque, and B. Chopard. "HPC Multiscale Simulation of Transport and Aggregation of Volcanic Particles". In: *2018 17th International Symposium on Parallel and Distributed Computing (ISPDC)*. 2018 17th International Symposium on Parallel and Distributed Computing (ISPDC). June 2018, pp. 25–32. DOI: 10.1109/ISPDC2018.2018.00013.

[80] P. Künzli, K. Tsunematsu, P. Albuquerque, J.-L. Falcone, B. Chopard, and C. Bonadonna. "Parallel simulation of particle transport in an advection field applied to volcanic explosive eruptions". In: *Computers & Geosciences* 89 (2016), pp. 174–185. DOI: https://doi.org/10.1016/j.cageo.2016.02.005.

[81] M. Lenormand, F. Jabot, and G. Deffuant. "Adaptive approximate Bayesian computation for complex models". In: *Computational Statistics* 28.6 (2013). Publisher: Springer, pp. 2777–2796.

[82] J. Lintusaari, M. U. Gutmann, R. Dutta, S. Kaski, and J. Corander. "Fundamentals and Recent Developments in Approximate Bayesian Computation". In: *Systematic Biology* 66.1 (Jan. 1, 2017), e66–e82. DOI: 10.1093/sysbio/syw077.

[83] E. Lorenz. "Predictability: A Problem Partly Solved". In: *Proceedings of the Seminar on Predictability, 4-8 September 1995*. Vol. 1. Backup Publisher: European Center on Medium Range Weather Forecasting. Shinfield Park, Reading: European Center on Medium Range Weather Forecasting, 1995, pp. 1–18.

[84] A. Masselot and B. Chopard. "A lattice Boltzmann model for particle transport and deposition". In: *EPL (Europhysics Letters)* 42.3 (1998), p. 259. DOI: 10.1209/epl/i1998-00239-3.

[85] A. Merzky, M. Turilli, M. Maldonado, M. Santcroos, and S. Jha. "Using Pilot Systems to Execute Many Task Workloads on Supercomputers". In: *Job Scheduling Strategies for Parallel Processing*. Ed. by D. Klusáček, W. Cirne, and N. Desai. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2019, pp. 61–82. DOI: 10.1007/978-3-030-10632-4_4.

[86] M. de'Michieli Vitturi and F. Pardini. "PLUME-MoM-TSM 1.0.0: A volcanic columns and umbrella cloud spreading model". In: *Geoscientific Model Development Discussions* 2020 (2020), pp. 1–46. DOI: 10.5194/gmd-2020-227.

[87] B. R. Morton, G. I. Taylor, and J. S. Turner. "Turbulent gravitational convection from maintained and instantaneous sources". In: *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences* 234.1196 (Jan. 24, 1956). Publisher: Royal Society, pp. 1–23. DOI: 10.1098/rspa.1956.0011.

[88] *NCEP/NCAR Reanalysis 1: NOAA Physical Sciences Laboratory*. National Centers for Environmental Prediction/National Weather Service/NOAA/U.S. Department of Commerce. 1994, updated monthly. NCEP/NCAR Global Reanalysis Products, 1948-continuing. URL: https://psl.noaa.gov/data/gridded/data.ncep.reanalysis.html (visited on 05/12/2021).

[89] J. A. Nelder and R. Mead. "A Simplex Method for Function Minimization". In: *The Computer Journal* 7.4 (Jan. 1, 1965), pp. 308–313. DOI: 10.1093/comjnl/7.4.308.

[90] J. M. Oberhuber, M. Herzog, H.-F. Graf, and K. Schwanke. "Volcanic plume simulation on large scales". In: *Journal of Volcanology and Geothermal Research* 87.1 (Dec. 1, 1998), pp. 29–53. DOI: 10.1016/S0377-0273(98)00099-7.

[91] L. Pacchiardi, P. Künzli, M. Schöngens, B. Chopard, and R. Dutta. "Distance-learning For Approximate Bayesian Computation To Model a Volcanic Eruption". In: *Sankhya B* (Jan. 2020). DOI: 10.1007/s13571-019-00208-8.

[92] W. Petersen and P. Arbenz. *Introduction to Parallel Computing: A practical guide with examples in C*. Oxford Texts in Applied and Engineering Mathematics 9, Jan. 2004.

[93] T. Pöschel and T. Schwager. *Computational Granular Dynamics*. Springer, 2005. DOI: 10.1007/3-540-27720-X.

[94] A. T. Prata, L. Mingari, A. Folch, G. Macedonio, and A. Costa. "FALL3D-8.0: a computational model for atmospheric transport and deposition of particles, aerosols and radionuclides – Part 2: Model validation". In: *Geoscientific Model Development* 14.1 (Jan. 25, 2021). Publisher: Copernicus GmbH, pp. 409–436. DOI: 10.5194/gmd-14-409-2021.

[95] G.-J. Qi, J. Tang, Z.-J. Zha, T.-S. Chua, and H.-J. Zhang. "An efficient sparse metric learning in high-dimensional space via l1-penalized log-determinant regularization". In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, 2009, pp. 841–848.

[96] D. Ramkrishna. *Population Balances: Theory and Applications to Particulate Systems in Engineering*. Academic Press, 2000.

[97] F. Reckziegel, A. Folch, and J. Viramonte. "ATLAS-1.0: Atmospheric Lagrangian dispersion model for tephra transport and deposition". In: *Computers & Geosciences* 131 (Oct. 1, 2019), pp. 41–51. DOI: 10.1016/j.cageo.2019.05.008.

[98] C. P. Robert and G. Casella. *Monte Carlo Statistical Methods*. Springer-Verlag New York, Inc., 2005.

[99] W. I. Rose and A. J. Durant. "Fate of volcanic ash: Aggregation and fallout". In: *Geology* 39.9 (Sept. 1, 2011), pp. 895–896. DOI: 10.1130/focus092011.1.

[100] E. Rossi, C. Bonadonna, and W. Degruyter. "A new strategy for the estimation of plume height from clast dispersal in various atmospheric and eruptive conditions". In: *Earth and Planetary Science Letters* 505 (2019), pp. 1–12. DOI: https://doi.org/10.1016/j.epsl.2018.10.007.

[101] F. Schroff, D. Kalenichenko, and J. Philbin. "Facenet: A unified embedding for face recognition and clustering". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 815–823.

[102] H. F. Schwaiger, R. P. Denlinger, and L. G. Mastin. "Ash3d: A finite-volume, conservative numerical model for ash transport and tephra deposition". In: *Journal of Geophysical Research: Solid Earth* 117 (B4 2012). DOI: https://doi.org/10.1029/2011JB008968.

[103] S. Scollo, M. Prestifilippo, M. Coltelli, R. A. Peterson, and G. Spata. "A statistical approach to evaluate the tephra deposit and ash concentration from PUFF model forecasts". In: *Journal of Volcanology and Geothermal Research* 200.3 (Mar. 2011), pp. 129–142. DOI: 10.1016/j.jvolgeores.2010.12.004.

[104] C. Searcy, K. Dean, and W. Stringer. "PUFF: A high-resolution volcanic ash tracking model". In: *Journal of Volcanology and Geothermal Research* 80.1 (Jan. 1998), pp. 1–16. DOI: 10.1016/S0377-0273(97)00037-1.

[105] M. V. Smoluchowski. "Drei Vortrage uber Diffusion, Brownsche Bewegung und Koagulation von Kolloidteilchen". In: *Zeitschrift fur Physik* 17 (1916), pp. 557–585.

[106] A. F. Stein, R. R. Draxler, G. D. Rolph, B. J. B. Stunder, M. D. Cohen, and F. Ngan. "NOAA's HYSPLIT Atmospheric Transport and Dispersion Modeling System". In: *Bulletin of the American Meteorological Society* 96.12 (2015). Place: Boston MA, USA Publisher: American Meteorological Society, pp. 2059–2077. DOI: 10.1175/BAMS-D-14-00110.1.

[107] J. L. Suárez, S. García, and F. Herrera. "A Tutorial on Distance Metric Learning: Mathematical Foundations, Algorithms and Software". In: *arXiv preprint arXiv:1812.05944* (2018).

[108] S. Suga. "Numerical schemes obtained from lattice boltzmann equations for advection diffusion equations". In: *International Journal of Modern Physics C* 17.11 (Nov. 2006), pp. 1563–1577. DOI: 10.1142/S0129183106010030.

[109] T. Suzuki. "A Theoretical Model fo Dispersion of Tephra". In: *Arc Volcanism: Physics and Tectonics* (1983).

[110] M. S. T. Toni. "Approximate Bayesian Computation Scheme for Parameter Inference and Model Selection in Dynamical Systems." In: *Journal of the Royal Society Interface* 31.6 (2009), pp. 187–202.

[111] K. Tsunematsu. “New numerical solutions for the description of volcanic particle dispersal”. PhD thesis. University of Geneva, 2012.

[112] K. Tsunematsu, B. Chopard, J.-L. Falcone, and C. Bonadonna. “Comparison of Two Advection-Diffusion Methods for Tephra Transport in Volcanic Eruptions”. In: *Communications in Computational Physics* (2011). DOI: 10.4208/cicp.311009.191110s.

[113] K. Tsunematsu, J.-L. Falcone, C. Bonadonna, and B. Chopard. “Applying a Cellular Automata Method for the Study of Transport and Deposition of Volcanic Particles”. In: *Cellular Automata*. Ed. by H. Umeo, S. Morishita, K. Nishinari, T. Komatsuzaki, and S. Bandini. Lecture Notes in Computer Science 5191. Springer Berlin Heidelberg, Sept. 2008, pp. 393–400. DOI: 10.1007/978-3-540-79992-4_51.

[114] M. Turilli, M. Santcroos, and S. Jha. “A Comprehensive Perspective on Pilot-Job Systems”. In: *ACM Computing Surveys* 51.2 (Apr. 17, 2018), 43:1–43:32. DOI: 10.1145/3177851.

[115] L. E. Veen and A. G. Hoekstra. “Easing Multiscale Model Design and Coupling with MUSCLE 3”. In: *Computational Science – ICCS 2020*. Ed. by V. V. Krzhizhanovskaya, G. Závodszky, M. H. Lees, J. J. Dongarra, P. M. A. Sloot, S. Brissos, and J. Teixeira. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 425–438. DOI: 10.1007/978-3-030-50433-5_33.

[116] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[117] A. C. M. Volentik, C. Bonadonna, C. B. Connor, L. J. Connor, and M. Rosi. “Modeling tephra dispersal in absence of wind: Insights from the climactic phase of the 2450 BP Plinian eruption of Pululagua volcano (Ecuador)”. In: *Journal of Volcanology and Geothermal Research* 193.1 (June 2010), pp. 117–136. DOI: 10.1016/j.jvolgeores.2010.03.011.

[118] J. B. Wardman, T. M. Wilson, P. S. Bodger, J. W. Cole, and C. Stewart. “Potential impacts from tephra fall to electric power systems: a review and mitigation strategies”. In: *Bulletin of Volcanology* 74.10 (Sept. 2012), pp. 2221–2241. DOI: 10.1007/s00445-012-0664-3.

[119] J. T. White, C. B. Connor, L. Connor, and T. Hasenaka. “Efficient inversion and uncertainty quantification of a tephra fallout model”. In: *Journal of Geophysical Research: Solid Earth* 122.1 (2017), pp. 281–294. DOI: https://doi.org/10.1002/2016JB013682.

[120] T. Wilson, C. Stewart, H. Bickerton, P. Baxter, V. Outes, G. Villarosa, and E. Rovere. *Impacts of the June 2011 Puyehue Cordon-Caulle volcanic complex eruption on urban infrastructure*. GNS Science, 2012.

[121] T. Wilson, C. Stewart, J. Cole, D. Johnston, and S. Cronin. “Vulnerability of farm water supply systems to volcanic ash fall”. In: *Environmental Earth Sciences* 61.4 (Dec. 2009), pp. 675–688. DOI: 10.1007/s12665-009-0380-2.

[122] T. M. Wilson, C. Stewart, V. Sword-Daniels, G. S. Leonard, D. M. Johnston, J. W. Cole, J. Wardman, G. Wilson, and S. T. Barnard. “Volcanic ash impacts on critical infrastructure”. In: *Physics and Chemistry of the Earth, Parts A/B/C*. Volcanic ash: an agent in Earth systems 45–46 (2012), pp. 5–23. DOI: 10.1016/j.pce.2011.06.006.

[123] A. W. Woods. “The fluid dynamics and thermodynamics of eruption columns”. In: *Bulletin of Volcanology* 50.3 (June 1988), pp. 169–193. DOI: 10.1007/BF01079681.

[124] Q. Yang, E. B. Pitman, M. Bursik, and S. F. Jenkins. “Tephra deposit inversion by coupling Tephra2 with the Metropolis-Hastings algorithm: algorithm introduction and demonstration with synthetic datasets”. In: *Journal of Applied Volcanology* 10.1 (Jan. 26, 2021), p. 1. DOI: 10.1186/s13617-020-00101-4.

[125] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, and I. Stoica. “Apache Spark: a unified engine for big data processing”. In: *Communications of the ACM* 59.11 (Oct. 28, 2016), pp. 56–65. DOI: 10.1145/2934664.