Development, implementation and efficiency optimization of novel methods to accelerate kinetic Monte Carlo simulations of reactive systems

Giannis Savva

A dissertation submitted in partial fulfilment
of the requirements for the degree of
Doctor of Philosophy
of
University College London

Department of Chemical Engineering
June 2022

I, Giannis Savva, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

……

Acknowledgements

This journey wouldn’t have started if I didn’t have the support of Prof. Andreas Boudouvis to whom I’m deeply grateful. I would also like to express my sincere appreciation and gratitude to my PhD supervisor, Dr. Michail Stamatakis, for his support, expert advice and encouragement throughout the duration of my PhD.

I cannot thank enough my colleagues across UCL and beyond; the last four years have been unique from many perspectives and my interaction with each one of them contributed into making these years worth remembering. Thinking differently is definitely the most important non-technical skill I acquired thanks to my colleagues to whom I am especially indebted.

Special thanks and wholehearted gratitude to my friends and my family for their unconditional support and understanding. Thank you very much all.

Declaration form

Paper 1

G. D. Savva and M. Stamatakis, *Comparison of Queueing Data-Structures for Kinetic Monte Carlo Simulations of Heterogeneous Catalysts*, The Journal of Physical Chemistry A, 2020, 124, 38, 7843-7856.

1. **For a research manuscript that has already been published**
   a. **Where was the work published?** The Journal of Physical Chemistry A
   b. **Who published the work?** ACS Publications
   c. **When was the work published?** 1 September 2020
   d. **Was the work subject to academic peer review?** YES
   e. **Have you retained the copyright for the work?** NO

   ☒ I acknowledge permission of the publisher named under 1b to include in this thesis portions of the publication named as included in 1a.

3. **For multi-authored work, please give a statement of contribution covering all authors:**
   - **G.D.Savva:** Formal analysis, Investigation, Methodology, Software, Validation, Visualization, Writing - original draft, Writing - review & editing.
   - **M. Stamatakis:** Conceptualization, Funding acquisition, Project administration, Resources, Software, Supervision, Validation, Writing - review & editing.
4. **In which chapter(s) of your thesis can this material be found?** 3
5. **Candidate's e-signature:** Giannis Savva

   Date: 8-6-2022
6. **Supervisor~~/senior author(s)~~ e-signature:** Michail Stamatakis

   Date: 10-Jun-2022

Paper 2

S. Ravipati, G. D. Savva, I.-A. Christidi, R. Guichard, J. Nielsen, R. Réocreux, and M. Stamatakis, *Coupling the time-warp algorithm with the graph-theoretical kinetic Monte Carlo framework for distributed simulations of heterogeneous catalysts*, Computer Physics Communications, 2022, 270, 108148.

1. **For a research manuscript that has already been published**
   a. **Where was the work published?** Computer Physics Communications
   b. **Who published the work?** Elsevier
   c. **When was the work published?** 31 August 2021
   d. **Was the work subject to academic peer review?** YES
   e. **Have you retained the copyright for the work?** NO

   ☒ I acknowledge permission of the publisher named under 1b to include in this thesis portions of the publication named as included in 1a.

3. **For multi-authored work, please give a statement of contribution covering all authors:**
   - **S. Ravipati:** Formal analysis, Investigation, Methodology, Software, Visualization, Writing - original draft, Writing - review & editing.
   - **G. D. Savva:** Formal analysis, Investigation, Methodology, Software, Validation, Visualization, Writing - review & editing.
   - **I.-A. Christidi:** Funding acquisition, Methodology, Project administration, Software, Writing - review & editing.
   - **R. Guichard:** Methodology, Software, Writing – review & editing.
   - **J. Nielsen:** Methodology, Software, Writing – review & editing.
   - **R. Réocreux:** Formal analysis, Investigation, Writing - review & editing.
   - **M. Stamatakis:** Conceptualization, Funding acquisition, Investigation, Methodology, Project administration, Resources, Software, Supervision, Validation, Writing – review & editing.
4. **In which chapter(s) of your thesis can this material be found?** 4
5. **Candidate's e-signature:** Giannis Savva
   Date: 8-6-2022
6. **Supervisor~~/senior author(s)~~ e-signature:** Michail Stamatakis
   Date: 10-6-2022

Paper 3

G. D. Savva†, R. L. Benson†, I.-A. Christidi, and M. Stamatakis, *In-depth performance analysis of the Time-Warp algorithm for distributed, on-lattice Kinetic Monte Carlo simulations*, (tentative title), *in preparation*.

2. **For a research manuscript prepared for publication but that has not yet been published**:
    a. **Where is the work intended to be published?** Physical Chemistry Chemical Physics (PCCP)
    b. **List the manuscript's authors in the intended authorship order:** G. D. Savva†, R. L. Benson†, I.-A. Christidi, M. Stamatakis
    c. **Stage of publication:**
        ☒ Not yet submitted
        ☐ Submitted
        ☐ Undergoing revision after peer review
        ☐ In press
3. **For multi-authored work, please give a statement of contribution covering all authors:**
    - **G. D. Savva:** Formal analysis, Investigation, Methodology, Software, Validation, Visualization, Writing - review & editing.
    - **R. L. Benson:** Formal analysis, Investigation, Methodology, Validation, Visualization, Writing - original draft
    - **I.-A. Christidi:** Funding acquisition, Methodology, Project administration, Software, Writing - review & editing.
    - **M. Stamatakis:** Conceptualization, Funding acquisition, Investigation, Methodology, Project administration, Resources, Software, Supervision, Validation, Writing – review & editing.
4. **In which chapter(s) of your thesis can this material be found?** 4
5. **Candidate's e-signature:** Giannis Savva
    Date: 8-6-2022
6. **Supervisor~~/senior author(s)~~ e-signature:** Michail Stamatakis
    Date: 10-Jun-2022

† equal contribution

Paper 4

G. D. Savva, M. Stamatakis, *Tackling the temporal stiffness on well-mixed chemical systems: a kinetic Monte Carlo algorithm for on-the-fly scaling of fast reactions via cost-error optimization*, *in preparation*.

3. **For a research manuscript prepared for publication but that has not yet been published**:
   a. **Where is the work intended to be published?** Journal of Chemical Physics
   b. **List the manuscript's authors in the intended authorship order:** G. D. Savva, M. Stamatakis
   c. **Stage of publication:**
      ☒ Not yet submitted
      ☐ Submitted
      ☐ Undergoing revision after peer review
      ☐ In press
7. **For multi-authored work, please give a statement of contribution covering all authors:**
   - **G.D.Savva:** Formal analysis, Investigation, Methodology, Software, Validation, Visualization, Writing - original draft, Writing - review & editing.
   - **M. Stamatakis:** Conceptualization, Funding acquisition, Methodology, Project administration, Resources, Supervision, Writing - review & editing.
8. **In which chapter(s) of your thesis can this material be found?** 5
9. **Candidate's e-signature:** Giannis Savva
   Date: 8-6-2022
10. **Supervisor~~/senior author(s)~~ e-signature:** Michail Stamatakis
    Date: 10-Jun-2022

Abstract

On-lattice Kinetic Monte Carlo (KMC) is a powerful computational method that is widely used to study chemical reaction on catalytic surfaces. It is an exact method able to capture surface inhomogeneities, e.g. due to interactions among the participating species, and handle systems with complex chemistries. KMC is exact in the sense that the method itself does not introduce approximations of any kind. Therefore, the results produced from a KMC simulation depend exclusively on the input, i.e. the lattice, the chemical reaction model, and the kinetic and energetic parameters thereof. However, KMC simulations of realistic systems tend to be computationally demanding, mainly due to the inherently serial nature of KMC since the reaction events are scheduled and executed one at a time.

This thesis focuses on methods and approaches to accelerate KMC simulations of reactive systems. First, the focus is on the scheduling of KMC events undertaken by suitable queueing systems. Different data structures are developed, implemented and benchmarked to identify those that deliver the best computational performance. Next, detailed performance evaluation and optimisation studies are performed for a newly implemented algorithm that enables distributed, on-lattice, KMC simulations. Lastly, the focus turns towards well-mixed chemical reaction systems exhibiting timescale disparity, i.e. system in which some reactions occur much more frequently than others. To tackle timescale disparity, a novel method is developed that reduces (downscales) the appropriate rate constants on the fly in an optimal and data-driven way. The developed method also provides estimates for the error introduced by the downscaling procedure.

The approaches developed and benchmarked enable KMC simulations to reach temporal and spatial scales that were previously unattainable. Thus, these methodological advancements are expected to have significant positive impact in future studies of complex systems.

Impact Statement

The energy problem along with the environmental impact and pollution due to human and industrial activities are more concerning than ever. The field of catalysis, which focuses, among others, on the conversion of chemicals in an energy-efficient manner, may play a critical role against these global challenges. The traditional experimental pathway towards developing better catalysts for specific needs is, in principle, trial-and-error based, time consuming and expensive. Virtual experimentation, i.e. computational modelling, is becoming more and more dominant on catalysts development. Kinetic Monte Carlo (KMC) is one such computational method that is widely used to study the performance of catalysts for specific reactions. KMC however tends to struggle to reach industrially relevant length and time scales due to its serial nature.

This thesis focuses on various methods to accelerate KMC simulations of reactive systems that are relevant to catalysis. From the fundamental research point of view, the outcomes of this Thesis can (a) guide the development of more efficient and accurate modelling software that is indispensable in all computational studies, (b) make possible the computational simulation of industrially relevant timescales, enabling, at the same time, the study of reaction systems that were impossible to model before, (c) provide insights, better understanding of the underlying physics and eventually a way to perform multi-scale simulations of catalytic reactions, from the molecular to the reaction level. From the academic perspective, the research projects completed on this thesis are the first steps towards expanding the applicability of KMC to larger scales that have been impossible to reach previously, and further work would enrich the literature with pioneering studies. In addition, the impact of this research may be far reaching to other fields, especially to biological physics. From the industrial perspective, these developments will provide a significant contribution towards realising the vision of “rational catalyst design”. From that point onwards, the potential applications are virtually unlimited, as may be reasonably assumed based on the usage of catalysts in the production of a vast range of chemical products, e.g. pharmaceuticals, fuels, fertilizers, plastics, etc.

Table of Contents

List of Abbreviations

CRN	Common Random Number
CRP	Common Reaction Path
df	downscale factor
DM	Direct Method
FRM	First Reaction Method
GVT	Global Virtual Time
IAT	Inter-Arrival Time
KMC	Kinetic Monte Carlo
MD	Molecular Dynamics
Mod-NRM	Modified Next Reaction Method
NRM	Next Reaction Method
PU	Processing Unit
QE	Quasi-Equilibrated
RTC	Random Time Change
SS	Strong Scaling
SSI	Snapshot Saving Interval
TPD	Temperature-Programmed Desorption
TSS	TimeScale Separation
WGS	Water-Gas Shift
WS	Weak Scaling
ZGB	Ziff-Gulari-Barshad

1. Introduction

Simply stated, a catalyst is a substance that increases the rate of a chemical reaction without being consumed and can be homogeneous or heterogeneous. A homogeneous catalyst is in the same phase (often liquid or gaseous) as the reactants. On the other hand, a heterogeneous catalyst is not in the same phase as the reactants; in this case, the latter are usually in the gas or liquid phase, whereas the catalyst is a solid. The field using the latter category of catalysts, namely heterogeneous catalysis, is the cornerstone of the chemical industry where, regardless of application and process specifics, the principle is the same: a composite, solid material is used to facilitate the transformation of the input material to something more useful or less harmful. Probably, the most familiar example is the catalytic converters that are widely used in vehicles to catalyse reactions such as the oxidation of CO and reduction of NO_x species [1, 2]. Other examples include gas synthesis such as hydrogen [3] and ammonia (NH_3) production from nitrogen and hydrogen for fertilizers through the Haber–Bosch process [4]. Especially for the latter example, although widely used for more than 100 years, there is ongoing research [5, 6] on the various conditions under which ammonia is produced.

Due to the complex nature of the conversion process of reactants to products, it is expected that an efficient catalyst for a specific application is not going to perform the same for other processes. This issue gives rise to the need of catalyst development suitable for specific applications, a task that is certainly not trivial [7]. Traditional catalyst development is more trial-and-error oriented, an approach which may turn out very expensive, particularly time consuming and without any sort of guarantee for success. It should, then, be clear that the design of a catalyst requires a more rigorous approach based on an in-depth understanding of the physico-chemical phenomena that occur at the interface of the catalyst and the substrate. However, phenomena relevant to catalysis span many scales, both spatial and temporal. The necessary insights on the occurring phenomena could come from computational modelling that is based on physics and chemistry principles. To understand the chemistry at the atomistic level, the electronic structure needs to be resolved. Then, molecular interactions, that

largely determine catalytic functionality, have to be well understood so that accurate predictions could be made. At higher scales, specifically at the reactor level, other relevant phenomena such as the type of the flow of the reactants over the catalyst need to be modelled. Consequently, a reliable computational technique that would drive the design of modern catalysts needs to be multi-scale as the observed outcome is a combination of contributions from the atomistic to the macroscopic scale. In addition, using computational modelling, one may perform catalyst screening, namely, choose the most promising combination of materials that enhances the efficiency of a catalyst and then proceed to their experimental characterisation in order to assess their performance. This is obviously advantageous as compared to the trial-and-error approach.

At the atomistic scale, first-principles methods, such as density functional theory (DFT), are used to study the chemistry (bond formation and breaking) of the reactions of interest as well as the interactions between molecules [8, 9]. At a slightly higher scale, molecular dynamics (MD), able to resolve trajectories of molecules using Newtonian mechanics, may be utilised to simulate the evolution of a catalytic system by capturing the interactions, calculated by DFT for example, via a reactive force field [10, 11]. MD can also provide additional information [12] that could not be obtained by lower level methods, because of intrinsic assumptions or computational constraints. Moving further towards larger spatial scales, the kinetic Monte Carlo method, abbreviated as KMC, coarse-grains space and time and provides statistical averages by sampling over simulated configuration trajectories [13-17]. Again, information obtained by lower level methods, such as reaction rates, energy barriers and energetic contributions of the configurations of interest, is properly parametrised and incorporated into the KMC simulation. Finally, at the reactor scale, which is the scale of industrial activity in chemical engineering, computational fluid dynamics (CFD) methods are used to numerically solve the equations governing the transport of mass, momentum, heat and species and direct the reactor design process [18, 19]. Reactions may also be taken into account in CFD [20] but information about them is obtained from lower level methods. Apart from the indirect coupling of methods just mentioned, the exchange of information can be direct and done during the simulation, namely the two methods run concurrently, each one of them

calculating quantities relevant to their scale and communicate the relevant quantities when necessary [21-23].

Since the goal of the research done on catalysts is the prediction of their performance, kinetic modelling of the relevant chemical processes is inevitable. For this purpose, macroscopic rate equations, for example Langmuir-Hinshelwood models, may be used. However, the fundamental assumptions thereof, i.e. the random distribution of adsorbates on the surface, the absence of interactions between the adsorbates, as well as the single site type, are not valid for real catalytic surfaces [24, 25]. Therefore, for reliable results, one has to look for more sophisticated kinetic modelling strategies.

The kinetic Monte Carlo method, introduced above, is intrinsically able to capture the structural inhomogeneity of a catalyst since multiple site types can be considered. In addition, adsorbate interactions and multi-dentate species, i.e. occupying more than just one lattice site, can be incorporated in the kinetic model. The above elements, considered altogether, make the KMC method able to deal with real-life surface problems of advanced complexity [15]. Extensive reaction networks with a large number of reactions and an even larger number of intermediate chemical species can, in principle, be studied using KMC.

No method comes, however, without any limitations. Compared to MD, KMC is certainly capable of accessing longer timescales since it only models the events relevant to catalysis (adsorption, desorption, surface diffusion and reaction), which are, in principle, rare compared to atomic vibrations [7]. Yet, even rare events themselves may occur across different timescales [16], a fact that brings KMC in the same disadvantageous position as MD when it comes to simulating a representative system and providing useful information on catalytic performance. The latter issue, which appears in both well-mixed and on-lattice KMC simulations, is known as the "*timescale disparity"* problem or "*KMC stiffness"* and it arises when at least one of the reactions or elementary steps considered in the model occurs at a (much) higher frequency than the other ones, i.e. has a larger kinetic rate. The "severity" of the timescale disparity is determined by the difference of the execution frequencies of the fastest and slowest step. The greater this difference is, the more pronounced the timescale disparity problem becomes. Especially for on-lattice KMC, multiple reasons may contribute to that such as very low energy barrier, high pre-exponential factor or the density

of reactive configurations might be much higher as compared to that of other reaction steps. For example, if the diffusion of an adsorbed species on the surface is much faster than the reaction involving two adsorbed molecules of this species, the simulation spends most of time mixing the adsorbates on the surface, which might be unimportant from one point onwards, rather than performing reactions that are of interest. Tackling the timescale disparity to reduce the computational cost of the KMC simulation can, in principle, be done by reducing the execution frequency of the very fast reaction via the manual reduction of the corresponding rate constants. Depending on the type of KMC simulation, i.e. well-mixed, where the species are uniformly distributed on a volume, or on-lattice, where space is explicitly taken into account, various simulation strategies have been developed to address the issue. A more detailed account on those studies is provided on the relevant sections in the main body of the thesis.

The power of the KMC method comes from explicitly simulating every event in a sequential order. If one wants to have an efficient implementation of a KMC method for on-lattice simulation, it is imperative to make sure that, among other operations, the scheduling and execution of elementary reaction events is implemented in the most efficient way possible. Such procedures are usually undertaken by queueing system modules, and their implementation may have a huge impact on performance.

In on-lattice KMC simulations [26], the catalytic surface is represented using a suitable lattice with one or more site types [27]. The size of this lattice is usually on the order of a few tens of nanometers [17], which is sufficient for most applications, since larger lattices increase the computational cost and might not provide additional insight into the process studied. However, certain experimentally observed phenomena on catalytic surfaces [28, 29], such as oscillations due to surface reconstruction and the formation of patterns whose wavelengths are on the order of μm and mm, cannot be captured using such small lattices. To enable the simulation of even larger domains, one has to resort to distributed computing methods [30, 31] and divide the workload among many computational processes, so that the simulation becomes tractable within a reasonable amount of time.

All the above techniques share the same goal: to accelerate KMC simulations. The goal is achieved by (a) reducing the rate constants of fast

reactions, which results in approximate methods, and (b) using more efficient algorithms, more powerful hardware, or distributing the workload. From the physics perspective, in the latter approach, the sequence of steps executed by the simulation software remains unchanged, namely the methodology is left intact. What is modified, however, is the *way* these operations are executed. It is crucial to note that, regardless of the implementation, the results given by any approach belonging to category (b) above should be exact, i.e. entail no approximations. In certain cases, the results of approach (b) should even be numerically identical. The latter restriction is also a powerful way of evaluating the correctness of alternative implementations.

In the current thesis, and in line with what has already been described, we follow different approaches aiming to accelerate the Kinetic Monte Carlo simulations of reactive systems. After a brief overview of the fundamental of the KMC approach (Chapter 2), we focus on implementing different data structures that take care of the scheduling and execution of elementary events during on-lattice KMC simulations (Chapter 3). We further develop another data structure to address specific needs of a particular class of simulations [32]. In Chapter 4, we investigate the performance of the newly implemented Time-Warp algorithm into the Graph-Theoretic KMC framework [33] of the software package *Zacros* [27, 34, 35]. Following the performance benchmarks, we move on into a detailed study on the dependence of the performance on the user-defined parameters using a few but representative chemical reaction models. Further, Chapter 5 of the thesis focuses on developing an approximate method the scales the rate constants of the fast reactions in well-mixed chemical reaction systems. As a final note, we re-iterate that Chapters 3 and 4 of the thesis (data structures for event scheduling & execution, and performance benchmarks and optimisation of the Time-Warp algorithm) are concerned with exact methods for accelerating KMC simulations. In contrast, Chapter 5, the scaling methodology of the rate constants, provides an approximate algorithm to reduce the computational cost of KMC simulations while providing error estimates. Finally, Chapter 6 provides a summary and future directions on accelerating KMC simulations.

2. Kinetic Monte Carlo (KMC) Fundamentals

2.1. Introduction

A chemical reaction can be seen as the conversion of interacting molecules from reactant to product species via rearrangement of their atoms. The simulation of the dynamic evolution of such reactive processes can in principle be achieved by atomistic simulation methods like molecular dynamics (MD), whereby the numerical integration of Newton's equations of motion resolves the trajectory of each atom in the system, under the effect of the chosen interatomic potential and boundary conditions. However useful this method has been so far [36], it is limited by the fact that accurate and stable integration requires extremely small time steps (~10^{-15} s) in order to capture atomic vibrations which, in turn, limits the accessible timescales to a few microseconds [37]. Therefore, phenomena or processes that take place on longer timescales, e.g. slow (infrequent) chemical reactions, are inaccessible by MD methods.

Kinetic Monte Carlo is a stochastic computational method that, unlike MD, does not resolve the entire trajectory followed by individual atoms and molecules [26, 38]. In KMC, the motion of the molecules is spatially coarse-grained in the sense that vibrations, which are not of interest, are not resolved at all. What is of interest, especially in the field of catalysis, is the occupancy of each site on the catalytic surface, represented by a suitably defined lattice, and the statistics of transitions from reactant to product states. This crossing of states is simulated explicitly in MD along with all the vibrations that lead to it. In KMC instead, only the initial and final states are considered and information about the transition is provided as an input in the form of a rate constant. Given an initial surface (lattice) configuration, with occupied and vacant sites, the chemical system is propagated

in time through a discrete set of configurations with known transition rates among the various states. This different simulation scope enables KMC to reach much longer timescales than what is achievable by MD methods [39, 40]. The stochastic nature of this method lies in the fact that only the transition rates among the different configurations are known instead of the exact occurrence time of each transition; the latter is in fact a random variable that depends on the transition rate constant.

In the heterogeneous catalysis field, where the interest is on modelling surface reactions, one uses an appropriate computational lattice as a representation of the catalytic surface. The lattice is simply a collection of discrete sites that, in principle, may differ in type. The sites that compose the lattice can be either vacant or occupied by an adsorbate. Furthermore, any species participating in the chemical reaction system may adsorb to a site, desorb from a site, hop from one site to a neighbouring one, i.e. diffuse, and finally react with other adsorbates found on the lattice. Given a particular state of the lattice, α, we may, at least, theoretically, calculate the probability that the system "jumps" to a different state, β. All the accessible states compose the state space of the particular system, denoted with Ω. In addition, the transition rate from a state α to another state β is denoted by $k_{\alpha\beta}$. These kinetic constants can be computed from transition state theory expressions parametrised with ab initio results. The equation that describes the dynamics of the above system can be written as [26, 41]:

$$\frac{dP(\alpha,t)}{dt} = \sum_{\beta \neq \alpha} \left[k_{\beta\alpha}\, P(\beta,t) - k_{\alpha\beta}\, P(a,t) \right] \qquad (2.1)$$

where $P(\alpha,t)$ is the probability of finding the system at the state α in time t. This equation, referred to as the "*Master Equation*", captures the statistics of a KMC simulation. It is actually a balance equation: the positive term under the sum operator over all states β accounts for the probability of jumping to state α while being in any state β and the negative term accounts for the probability of leaving state α towards any other state β.

The master equation seems deceptively simple for its capability to describe a system in its entirety. Indeed, even for small lattices and simple chemical systems, the state space is composed of an enormous number of configurations that makes its numerical solution intractable. As an example, consider a lattice with 100 sites (let it be a 10×10 lattice), and a reactive system with two participating species, each of which occupying a single site once adsorbed on the lattice. In this case, the number of all possible states is approximately 5×10^{47}. If, instead, we choose a 20×20 lattice (400 sites), the number of states is on the order of 10^{191}. If more complexity is added by considering one more single-site species, thus three species in total, the configuration space ends up having more than 10^{240} states. Fortunately, one does not have to numerically solve the master equation. What is done, instead, is to produce trajectories that are statistically consistent with the master equation [42]. In other words, spatial KMC methods are simulating a single configuration trajectory that can be followed by the system under study. Statistical averages are obtained by performing the simulation multiple times and the averaged results are statistically equivalent to the results that would have been obtained if the master equation was solved [14].

2.2. KMC methods

KMC is a purely numerical method, which means no further approximations are introduced once the mechanistic details under study and the corresponding rate constants have been determined [14]. For the generation of one or multiple sequences of configurations, or trajectories, consistent with the underlying master equation, many algorithms have been developed that yield statistically equivalent results. The main idea is to determine the time of occurrence of the next elementary event and the type of process that will occur. In addition, for on-lattice systems, one needs to determine the position of the executed process on the lattice. The latter is usually done along the process selection procedure. Each one of these steps can be performed in multiple ways that leads to different algorithms. In the subsequent sections two of these methods are discussed: the Direct Method (DM) and the First Reaction Method

(FRM), both developed and proved equivalent by Daniel T. Gillespie in 1976 [43]. The Direct Method is also referred to as Variable Step Size Method, VSSM [26, 44]. Following Gillespie's methods, additional KMC methods have been developed as extensions of the FRM. Those are the Next Reaction Method (NRM) by Gibson and Bruck [45] and the Modified Next Reaction Method (Mod-NRM) by Anderson [46]. To facilitate the readability of the thesis, avoid repetition and improve continuity, the last two methods are introduced and discussed in chapter 5 due to their closer relevance with the work presented there.

Gillespie laid the foundations of both the Direct and the First Reaction Methods in his seminal work published in 1976 [43]. Then, he applied the Direct Method on multiple well-mixed, i.e. spatially homogeneous chemical reaction systems, such as the radioactive decay, the Lotka reaction, the Brusselator and the Oregonator [47]. In those systems, the evolution of the species' populations was tracked over time and different phenomena on the phase-space were demonstrated, e.g. limit cycles. On the contrary, in on-lattice systems, one is interested in production rates, or species coverages. For both well-mixed and on-lattice systems, the same principles apply when it comes to simulating them using a KMC method. Since Chapters 3 and 4 are related to the on-lattice KMC implementation of the First Reaction Method, in the following subsections, the methods are described in the context of on-lattice simulations and further details are provided in the appropriate chapter.

2.2.1. Direct Method

Suppose that we are given a system in a particular state and based on the defined reaction mechanism, we want to find the reaction (with its position on the lattice) that is going to happen next and the time that this reaction will occur. Given a lattice configuration, we identify all realisable elementary events so that each one corresponds to a particular set of lattice sites. For example, if our reaction system incorporates adsorption of a species, we identify and uniquely label all adsorptions to vacant sites on the lattice. Using this approach, a reaction and its position can be determined simultaneously.

Two uniformly distributed random numbers, r_1 and r_2, in the range of (0, 1) are needed in every KMC step in order to determine the next reaction and its time. The time advancement, t_{adv}, is calculated as:

$$t_{adv} = -\frac{ln(r_1)}{k_{tot}} \tag{2.2}$$

where *ln* denotes the natural logarithm and k_{tot} is the sum of all the reaction rates of the feasible events on our lattice. Then, the occurrence time, t_{KMC}, of the next event is:

$$t_{KMC} = t_{cur} + t_{adv} \tag{2.3}$$

where t_{cur} is the current KMC time, that equals to zero in the beginning of the simulation. We note that the determination of the occurrence time is completely decoupled from the reaction that is going to occur; the only quantity involved in the equation (2.2) is the sum of rate constants (apart from the random number which is not system dependent).

The second uniformly distributed random number r_2 is used for the determination of the next process, j_p, as:

$$\sum_{i=1}^{j_p-1} k_i < r_2 \times k_{tot} \leq \sum_{i=1}^{j_p} k_i \tag{2.4}$$

where k_i is the rate constant of the i^{th} realizable event on the lattice. Practically, the second random number is multiplied by the sum of the kinetic rates to form the quantity $r_2 \times k_{tot}$. We start summing the kinetic constants of all the realizable reactions until our running sum exceeds the quantity $r_2 \times k_{tot}$. The last reaction constant added will indicate the reaction that occurs next. An important point is that the order in which we sum the reaction rates does not *statistically* affect the simulation, in the sense that the latter will still reproduce the correct probabilistic dynamics as described by the master equation.

Computationally, one has to identify (and uniquely label) all feasible processes on the lattice, sum their rate constants to form k_{tot}, generate the time of the next event, sum individual kinetic rates k_i to find the next event and execute it. Post-execution, some new elementary processes are enabled and some are disabled. The sum of the kinetic rates, k_{tot}, has changed and should be updated. Lastly, the previous procedure is repeated and the system is propagated in time through a single trajectory, which is consistent with the master equation.

2.2.2. First Reaction Method

The main idea of this method is that a random inter-arrival time is generated for every feasible process on the lattice and the one with the minimum time is going to occur next. The waiting time t_i of each event i is calculated via the equation:

$$t_i = -\frac{ln(r_i)}{k_i} \tag{2.5}$$

where r_i is a random number from the uniform distribution in the unit interval and k_i is the rate constant of elementary event i. All these inter-arrival times are properly catalogued and a procedure to identify their minimum is executed. The minimum waiting time, $t_{min} = min(t_i)$, corresponds to a lattice process, j_{exe}, that is chosen as the next event to occur. The KMC clock then advances by t_{min}.

In contrast to the direct method, the first reaction method necessitates the generation of multiple random numbers per KMC step. In the beginning of the simulation, N random numbers are needed, where N is the total number of feasible processes identified on the lattice. In the subsequent steps, one does not have to generate new random inter-arrival times for all events; that would make this method very inefficient. Since the execution of an event incurs “local” changes around the neighbourhood of the executed event, only the waiting times of the affected events have to be recalculated. On the other hand, the waiting times of all other realisable events may remain intact and are still valid.

3. Accelerating the scheduling and execution of elementary reactions on lattice KMC

The work presented on this chapter has already been published. In accordance with the policy of the copyright holder, the main body, tables and figures that appear in this chapter are *Reproduced with permission from G. D. Savva and M. Stamatakis, J. Phys. Chem. A 124, 7843 (2020). Copyright 2020, American Chemical Society.*

3.1. Introduction and previous works

In general, kinetic Monte Carlo algorithms undertake the simulation of a trajectory that stems from the properties of the system under study. Regardless of implementation specifics, at every step of a KMC simulation we have a list of all the possible events that may happen on the lattice. We, then, need to randomly select an event along with its time of occurrence and execute it. Finally, we perform the necessary updates on both the lattice and the list of possible events. The bookkeeping of these procedures, namely the selection of a process, its execution and the post-execution updates, depends on the algorithmic implementation and is usually carried out using appropriate data-structures. For instance, Jansen has used a binary search tree to store the reactions while simulating systems with time-dependent rate constants [44]. Later, Lukkien and co-workers developed three KMC methods and have used a binary tree to store the possible reactions [48]. Along these lines, Gibson and Bruck developed another KMC algorithm relying on an indexed priority queue [45], which enables the retrieval of an arbitrary reaction in constant time. The same authors have also proposed the use of a complete tree data-structure with a divide-and-conquer search algorithm in order to achieve an execution time that scales logarithmically with respect to the total available reactions on the lattice. More recently, Chatterjee and Vlachos [49] reviewed the available KMC algorithms along with the efficiency of the various implementations. They report that implementations

involving linear search on the data-structure scale as $O(N_{queue})$ with N_{queue} being the total number of elements in the queueing data-structure. They first report further extensions to the linear search such as the two-level and n-level linear search that scale as $O(\sqrt{N_{queue}})$ and $nO(\sqrt[n]{N_{queue}})$ respectively. The binary and hash-table search are discussed as well. Chatterjee and Vlachos [49] conclude that sophisticated search algorithms can result in significant computational savings. In addition, due to the fact that reaction occurrence in a lattice KMC affects only a limited number of neighbouring sites, updates to the corresponding queueing data-structures are limited ("local updates"). Thus, exploiting this property also leads to computationally efficient simulations.

In the studies just noted, the main motivation behind the development of new data-structures for KMC was to improve the efficiency of the simulation, and this was shown to be achievable using some variant of a divide-and-conquer strategy (binary trees are essentially implementations based on this concept). However, a detailed comparative study on the performance of a wider variety and more modern data-structures as queueing systems for KMC is lacking. Such comparisons could be of interest to developers of kinetic Monte Carlo software, such as *Zacros* [50, 51], SPPARKS [52, 53], KMCLib [54], kmos [55], EON [56], CARLOS [57], MonteCoffee [58], and MoCKa [59] for materials and catalysis simulations, or Dizzy [60], StochPy [61], and CERENA [62] for biological reaction simulations, but also for users, who would like to adopt the most efficient implementations for their simulations. In this work, we present the implementation and comparison of four data-structures (i.e. the unsorted list, the binary heap, the pairing heap and the skip list), as alternative queueing systems for KMC simulations, and we further develop the skip list to address specific needs of such simulations. These four+one queueing systems are incorporated in the Graph-Theoretical kinetic Monte Carlo (GT-KMC) framework as implemented by the KMC software package *Zacros* [51, 63]. The performance of the queueing systems is evaluated using three different chemical reaction models: the Ziff-Gulari-Barshad (ZGB) CO oxidation model [64], a simplified water-gas shift (WGS) model based on Ref. [65] and a temperature-programmed-desorption (TPD) model of CO from a pure Cu surface [66]. As a comparison measure, we use the time taken to simulate a fixed number of KMC steps with each of the

different approaches. Aiming to assess the core performance of our algorithms in a variety of settings, we compile our code with and without optimisations and we present and compare results for both cases.

In the following sections, we present briefly the KMC method as implemented by *Zacros* and the development and implementation methodology for the data-structures (Section 3.2). We then proceed with the description of the chemical models used to benchmark our queueing systems along with our results in Section 3.3. Finally, Section 3.4 summarizes our results as well as their importance, while we provide some useful generic guidelines on the choice of data-structures depending on the chemical system under study.

3.2. Methodology

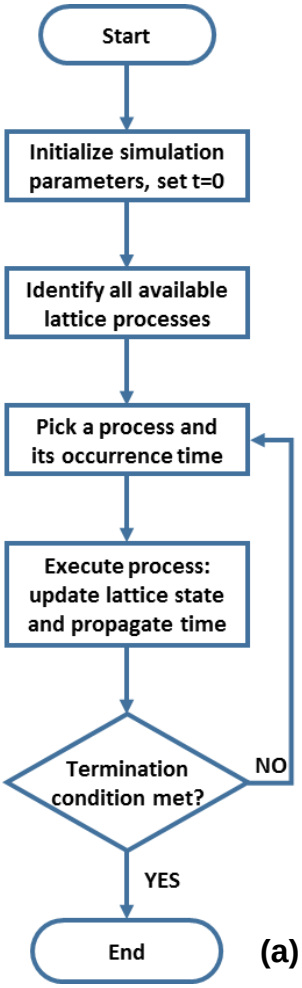
3.2.1. Kinetic Monte Carlo

The backbone of the on-lattice KMC method, schematically shown in Figure 1(a), consists of repeatedly identifying and randomly selecting a realizable elementary event among adsorption, desorption, diffusional hops and reaction on the surface as listed in Figure 1(c). As already mentioned, the widely used and statistically equivalent KMC methods are the Direct Method and First Reaction Method, by Gillespie [43]. The idea behind the Direct Method is to use two random numbers, one that picks the reaction r that is going to happen next and another that determines the time, τ^*, when this reaction is going to occur. On the other hand, the First Reaction Method makes use of the intuitive idea that the next reaction to be executed has to be the most imminent one, namely the one with the smallest waiting time. Therefore, a tentative reaction time τ_i is generated for every possible reaction i. Then, the reaction to execute and its occurrence time are both determined by picking the minimum time, τ^*, among all the reaction times τ_i. These two methods are able to stochastically simulate a system exactly and are equivalent [43], in the sense that the pair (r, τ^*) in both methods is drawn from the same probability distribution. *Zacros* [51, 63] implements the *First Reaction Method*, which is described in more detail below.

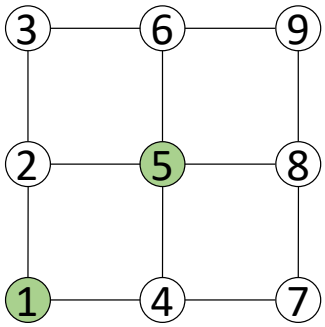
A KMC simulation is initiated by providing *Zacros* with an appropriate representation of the catalytic surface in the form of a lattice (e.g. as the one shown in Figure 1(b)), as well as an ensemble of chemical reactions along with the relevant reaction rates and energetic contributions to the lattice. Next, all elementary events are identified, as listed in Figure 1(c), and for each one of them a random occurrence time t_i is generated as

$$t_i = t_{cur} - \frac{1}{k_i} \ln(u) \tag{3.1}$$

where t_{cur} is the current KMC time (in the beginning $t_{cur} = 0$), k_i is the kinetic rate constant of that particular elementary event and u is a uniform random number in the range (0, 1). Subsequently, the minimum time among all the reaction times is retrieved and the reaction that it corresponds to is executed. Reaction execution



(a)



(b)

Label	Lattice Process
1	Desorption from site 1
2	Diffusion 1 → 2
3	Diffusion 1 → 4
4	Adsorption to site 2
5	Adsorption to site 3
6	Adsorption to site 4
7	Desorption from site 5
8	Diffusion 5 → 6
9	Diffusion 5 → 8
10	Diffusion 5 → 4
11	Diffusion 5 → 2
12	Adsorption to site 6
13	Adsorption to site 7
14	Adsorption to site 8
15	Adsorption to site 9

(c)

Figure 1: (a) Flowchart of the KMC method. **(b)** Non-periodic square lattice with sites 1 and 5 being occupied by an adsorbate. **(c)** List of all realisable elementary events of the lattice shown in (b).

includes updating the state of the lattice and the occurrence times of the affected elementary events only [45, 49].

Computationally, one has to create a process queue to store the absolute occurrence times, generated via equation (1), of all the available processes, and identify their minimum along with the individual event it corresponds to. Then, the chosen event, *j*, is executed and the lattice state and the process queue entries are updated. Updating the process queue includes:

(i) removing the elementary processes that are no longer realizable,
(ii) detecting the new possible elementary processes, assigning each of them a random time and inserting it in the queue, and
(iii) modifying the inter-arrival time of processes for which the rate constant changed due to adsorbate-adsorbate lateral interactions as a result of the execution of process *j*.

This procedure is iteratively repeated until a termination condition is met, e.g. a final KMC time is reached or a predefined number of KMC steps have been simulated. If numerous lattice processes are stored in the queue and long-range interactions are included in the modelling mechanism, inserting, deleting and updating elements may become computationally expensive.

Throughout the course of a KMC simulation, all the elementary lattice processes have a unique identifier, their label, which is generated upon their insertion in the process queue. Moreover, each lattice process holds all the information needed for its execution: (i) the reactants involved and (ii) the products and the lattice sites on which the former have to be added post-execution. These lattice processes are always referenced using their unique label and for that reason the update and removal operations of the process queue, as highlighted in the previous paragraph, take as input the label of the lattice process that needs to be updated or removed. Such labels constitute extra information that needs to accompany the occurrence time of each lattice process; whenever not possible to infer the label indirectly, it has to be stored in the queueing system in a way described in more detail in the following section.

3.2.2. Queueing system data-structures

In this subsection, we present the main architecture of the data-structures implemented as queueing systems. For the skip list data-structure, we elaborate on the motivation for further development as well as the modifications to the original data-structure. In the following discussion, we refer to the *occurrence times* as *elements*, unless a different definition is given [32].

3.2.2.1. Unsorted list

The simplest way to keep track of all the occurrence times is to store them in a one-dimensional array as they are being generated via equation (1). The array index serves as the elementary event identifier (label) since it uniquely describes an event. The retrieval of the minimum element requires to iterate over all the entries in the array, therefore, this operation scales as $O(N)$ where N is the number of elements the array contains. Given its label, removing an element is a constant time operation, namely, the time required to perform it does not depend on the size of the array. In addition, if no “gaps” are allowed in the array, that is, upon a removal, we move the last element to the position of the just-removed-element, then the insertion of a new element is done in constant time, since it is always inserted at the back of the list. Similarly, the modification of an existing value in the list is a constant time operation.

3.2.2.2. Binary heap

The binary heap (chapter 6 in Ref. [67]) is a data-structure that belongs to the broader family of binary tree structures, in which each node has at most two children. In addition, the binary heap is always complete in the sense that all levels, with the exception of the bottom one, are fully filled with elements (Figure 2). Moreover, every node in the binary heap has “priority” over all its children, a property that makes the binary heap partially ordered. However, no other conclusions may be inferred regarding elements in different branches and

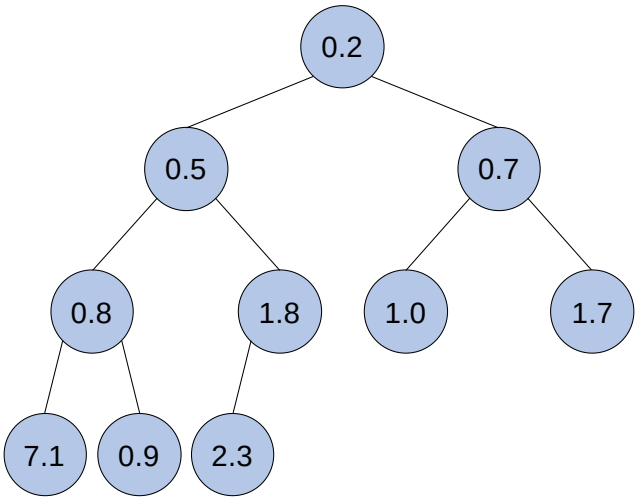


Figure 2: A binary heap with 10 elements where the bottom level is incomplete.

different levels. Due to the partial order, the minimum element is always on the top node of the binary heap and its retrieval is a constant time operation.

A new element is inserted as a new leaf at the bottom level to the left-most position. If the new element has priority over its parent, the two elements are swapped (refer to the Appendix I, section 1, for graphical representation of the operation). These swaps restore the partial order of the binary tree in $O(\log_2 N)$ computational time. On removal of an element, the rightmost node of the bottom level replaces the removed node. Since it is likely that partial order is now violated, the previously last element is swapped by its current parent or the minimum of its children depending on the priority it has over them. Likewise, on update, the new value floats upwards or sinks down based on its priority over its parent and children nodes. Both operations exhibit $O(\log_2 N)$ execution time scaling.

Since the binary tree is complete, it can be implemented using an array instead of pointers (see Appendix I, section 1); therefore, each element (occurrence time) has a unique index that serves as the element identifier, just like in the case described in subsection 3.2.2.1. Nevertheless, this array index does not correspond to the lattice process label. Other supporting arrays are used to ensure a one-to-one connection between a lattice process, its occurrence time and its position on the binary heap, equivalently, its position on the 1-D array representing the binary heap (refer to Appendix I, section 1 for more details).

Given its label, any element is retrievable in constant time and therefore no additional overhead is incurred on the update and removal operations.

3.2.2.3. Pairing heap

The pairing heap, introduced by Fredman and co-workers,[68] is a heap data-structure in which elements are stored partially sorted. Unlike the binary heap, the pairing heap allows for an arbitrary number of children per node as illustrated in Figure 3. However, due to this shape flexibility, the notion of *completeness* does not apply to the pairing heap; therefore, from the implementation point of view, a pairing heap requires pointers to connect the nodes. In addition, for an efficient implementation, the *binary tree representation* (chapter 10.4 in Ref. [67], [68]) is often used (see Appendix I, section 2, for details). Direct access to elements is provided by an indexing array with pointers, in a similar manner as the indexed priority queue of Gibson and Bruck [45].

To insert a new element in a pairing heap, we compare its value with the value of the top node. If the value of the new element is smaller than the top node, then the new element becomes the top node, otherwise the new element is added as the left-most child of the top node. The insertion operation involves just a numerical comparison, so it is a constant time operation. Due to the partial order property, finding the minimum is a constant time operation as well.

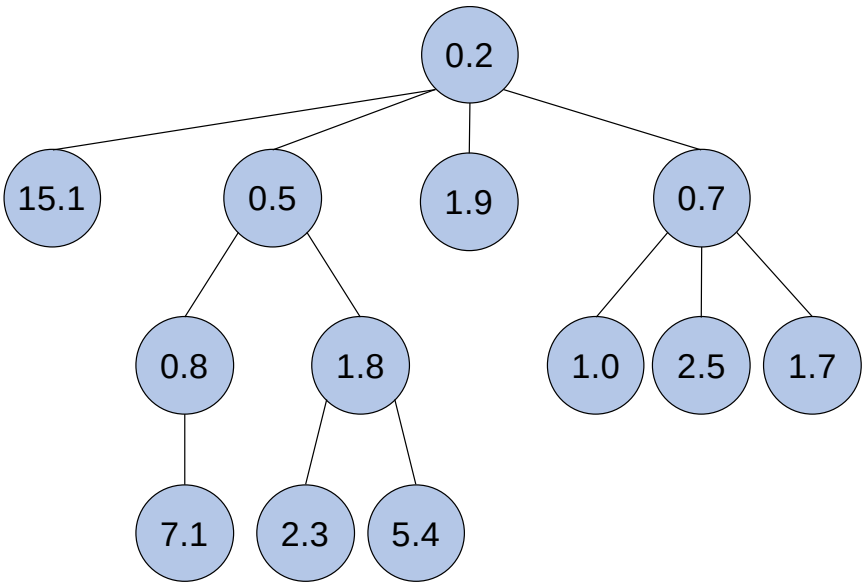


Figure 3: A pairing heap representation.

The deletion operation requires reconnecting together all the children (single nodes and entire subtrees) of the deleted node. However, the order in which the tree recombination is carried out is crucial for performance as discussed in Ref. [68]. Five different variants are presented in Ref. [68] and in our implementation we have chosen the two-pass-in-opposite-directions variant (as illustrated by Fig. 7 in Ref. [68]). It can be proven [68] that the removal operation runs in $O(\log_2 N)$ amortized time.

The update operation, namely, the modification (increase or decrease) of an existing value of the pairing heap, can be implemented as a removal following an insertion operation. Therefore, it also has amortised time complexity $O(\log_2 N)$. Observing that increasing a value may not violate the partial order property if the node of interest has no children, we are able to perform the update operation in constant time at the cost of checking whether the node whose value is to be decreased has a child or not.

3.2.2.4. Skip list

The skip list is a fully ordered, linked list based data-structure proposed by W. Pugh as a probabilistic and simpler alternative to balanced trees.[69] Figure 4 illustrates a simple skip list containing six elements. The bottom layer is an ordinary linked list, where every node points to its next one in the queue. Removing an element from a linked list requires the traversal of all the nodes preceding the element being sought for deletion. Similarly, inserting a new element requires the identification of its proper location. Therefore, the main idea behind the development of this data-structure was to find a way to traverse a linked list as quickly as possible using a divide-and-conquer approach instead of sequential access.

This functionality is achieved by allowing the nodes to have variable height thus forming interconnected linked lists, which compose the levels of the skip list. The “higher-level” linked lists are sparser, as they skip progressively more and more elements of the original list. Thus, these different levels provide a shorter path (i.e. faster access) to the nodes further down the list, thereby accelerating the search, remove and insert operations. These operations are performed

hierarchically, starting from the top linked list and continuing to the lower levels, enabling the skipping of a significant number of intermediate nodes. The above strategy (essentially a divide-and-conquer implementation) improves the amortized execution time of the search, insertion and deletion operations to $O(\log_2 N)$, where N is the number of elements in the skip list. The update operation, seen as a combination of a removal plus an insertion, also has an amortized execution time of $O(\log_2 N)$.

Each node stores the key value and one pointer per level, pointing to the next node in the linked list of that level. For our purposes, the key value is the time of occurrence of an elementary lattice process. As already mentioned, the skip list's nodes may have heights greater than one, whose unitary increments are determined by independent identically distributed Bernoulli random variables with success probability $p \in (0, 1)$. Therefore, the resulting heights follow the geometric distribution with parameter *p*. Following the definition, the expected maximum number of levels an individual node may have is given as:

$$H_N = \log_{\frac{1}{p}} N \tag{3.2}$$

Most often, a value of p=1/2 is used, which offers a good balance between speedup and memory requirements. Different applications may benefit from choosing a different value for *p*, for example 1/4 or 1/e, as discussed in Ref. [69], by trading the search cost with storage cost.

The search operation starts from the topmost level of the head element and follows the forward pointers as long as the element being searched for is not overshot, supposing that it exists in the skip list. The search continues to the immediate lower level until the bottom level is reached and the element is found. If the element does not exist in the list, the search operation finds its predecessor and therefore the appropriate position to insert it. Deletion also uses the search operation to find the element to be deleted.

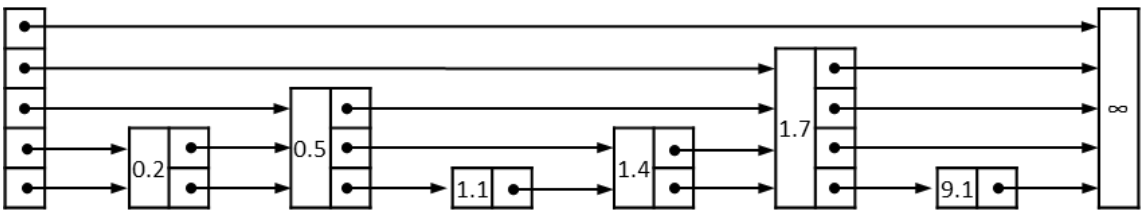


Figure 4: A simple skip list representation. The first and last nodes are termed head and tail respectively.

Both insertion and deletion operations require that the rightmost pointer of every level in the search path is temporarily stored in an array, which we refer to as *update_array* in subsequent sections. These pointers are used to correctly connect the new node with the existing ones upon insertion or the remaining nodes upon removal. A more thorough description of the skip list data-structure, its operations and pseudocode can be found in the original study.[69]

3.2.2.5. Developing skip list for KMC simulations

From the event scheduling perspective along the course of a KMC simulation, an efficient data-structure serving as the queueing system should incorporate effective implementations of the following operations:

a) Retrieval of the element with minimum key value. This value is the occurrence time of the next event, which is accompanied by a unique identifier / label that points to a lattice process.
b) Insertion of new elements. These new elements represent lattice processes that are created as a result of the previously occurred event.
c) Deletion of existing elements identified by their label (instead of their occurrence time as in the original skip list data-structure). The elements to be deleted represent the lattice processes that are no longer valid/realisable, for example, diffusion of a species that has desorbed in the previous KMC step.
d) Update of values that exist in the event queue. These elements represent events that are affected by a previously occurred reaction, as a result of changing the lateral interactions in the local neighbourhood of the just-occurred event. An example of such a case would be updating the inter-arrival time of a desorption event as neighbouring sites become more crowded and the spectators exert repulsive forces to the desorbing molecule.

Aiming at improving the timing of the above operations in the context of a KMC simulation, we further developed the skip list data-structure. The insertion operation remains the same as originally proposed.[69] However, the data-

structure was heavily adapted in order to improve the efficiency of the deletion operation. The rationale behind the development as well as the algorithm of the modified operations are presented below.

By definition, a skip list enables the execution of the “retrieve minimum value” operation (a) discussed above, since the entries are fully sorted and the element with the minimum key value is located at the very first node, immediately after the head element. We modify the node’s structure so that the unique element identifier, i.e. the label, is also stored within the node in addition to the key value. In the KMC context, this minor modification enables finding both the lattice process label and the time of occurrence in the most efficient way, i.e. in constant time, O(1).

As mentioned further above, deleting an element from a skip list, given its key value, requires searching for it, at a cost of $O(\log_2 N)$. However, if we wish to delete an element that is described by its label rather than its key value, as per operation (c) above, the situation is even worse, since skip list is fully sorted based on key values and not the labels. Searching for a specific label would require accessing all the nodes sequentially, thereby causing the deletion operation to scale as $O(N)$. The need to search elements in the process queue by their label comes from the fact that when a lattice process is executed in *Zacros* (or other KMC implementations), certain subroutines provide the labels of all the lattice processes which are eliminated (becoming obsolete) as a result of the “ongoing” process. Thus, the lattice processes affected by e.g. a diffusional hop, are given by their label, and using this label, one is able to find their occurrence time.

To enable deletion of an element given its label, we created an auxiliary array, called *mapping_array*, which provides access to the skip list’s elements by label in ascending order. Each entry of the *mapping_array* is a pointer to the corresponding element of the skip list as shown in Figure 5. It is now possible to retrieve in constant time the element with label *j* (where *j* corresponds to a lattice process) by evaluating *mapping_array[j]*, without going through the whole skip list. For the deletion operation to be fully functional, the skip list is turned from a singly linked-list based into a bidirectional linked-list based. Each node now retains information about its forward nodes but also its backward nodes. Reconnecting the remaining nodes after a deletion is possible using the

backwards node instead of the *update_array*. Algorithm 1 presents the core of the deletion operation; looping over the levels of a node and performing reconnections. Through these modifications, the deletion operation is performed in almost constant time, $O(1)$, since only a few node reconnections, with an upper bound of $2 \times H_N$, are needed.

Referring to the operation list at the beginning of this subsection, the insertion operation (b) is also adapted according to our modifications. Upon the creation of a new node with key value *t* and label *N+1*, the pointer *mapping_array[N+1]* is connected to the new node. Necessary operations are performed so that the backward pointers of the new node, as well as those of its following node, are connected (refer to Algorithm 2 and Figure 6). It is important to emphasize that all the pointers are pointing to the whole node (illustrated as grey rectangles in Figure 5) and not at an individual level of a certain node. We now proceed to discuss the insertion procedure in more detail with reference to the numbered lines of Algorithm 2 and Figure 6, which represents the operation

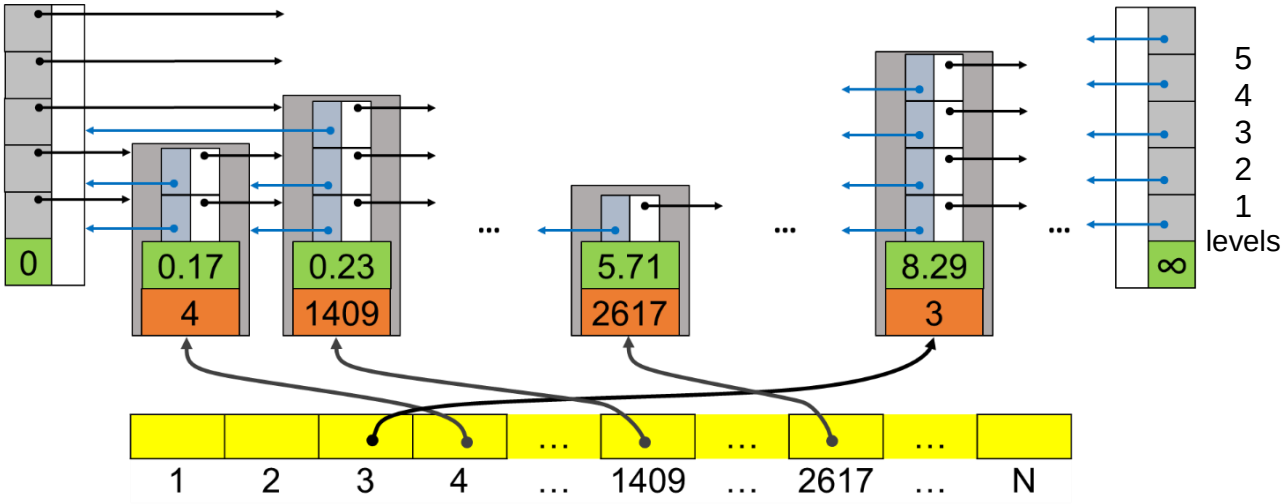


Figure 5: Schematic representation of the developed skip list-based data-structure. Nodes are represented by grey rectangles. The green cells contain the key value while the orange ones contain the corresponding label. The mapping_array is represented with yellow color and provides direct access to elements identified by their label. The first (head) and last (tail) node serve as "guard" nodes and they have only forward and backward pointers respectively. Levels are numbered from bottom to top.

graphically. Once the new node is created (Alg. 2, 2-4) and its position in the skip list is identified (Alg. 2, 5-6), the next step is to connect it with its predecessor and

successor nodes. At first, the new node is linked to its previous node via the *update_array* (Alg. 2, 7.1) which stores the rightmost pointer of every level in the search path. Likewise, the link to its next node is obtained from the *update_array* (Alg. 2, 7.2). Then, the previous node is redirected to point to the new node (Alg. 2, 7.3). Finally, the backward pointer of the next node is redirected to point to the new node as well (Alg. 2, 7.4).

Following the insertion and deletion operations, the update operation, which is a combination of a removal and an insertion, benefits from the improved deletion algorithm. In the following sections, we refer to the developed data-structure described above as *2-way skip list* while we refer to the original skip list-based queueing system by the name *1-way skip list*. These names originate from the direction of the pointers of each data-structure.

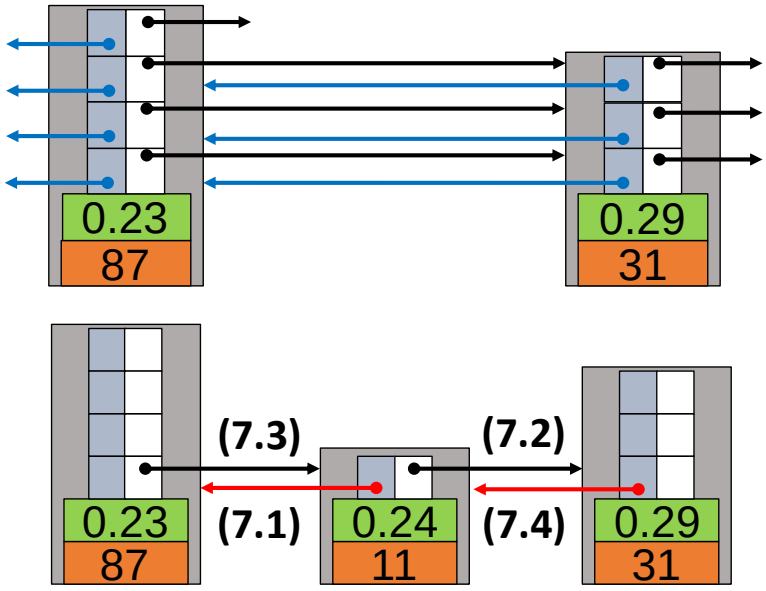


Figure 6: Reconnection of the nodes upon insertion of the element with key value of 0.24 and label 11. Top panel: a part of the queue with the initial connectivity. Bottom panel: connectivity of new pointers after insertion of the new element. The numbered arrows, (7.1)-(7.4), represent the links created by the respective code lines in Algorithm 1. The rest of the pointers are omitted for visual clarity. Green cells correspond to key values; orange cells to labels.

Delete(x)

```
1)   temp → mapping_array[x]
2)   for i:=1 up-to height(temp) do
     2.1) temp.backward[i].forward[i] → temp.forward[i]
     2.2) temp.forward[i].backward[i] → temp.backward[i]
```

Algorithm 1: Delete the node with label x.

Insert (list, new_key, new_label)

```
1)   current_list_level := height of the tallest node in
                        list
2)   height := random_height()
3)   new_node → create_node(height, new_key, new_label)
4)   mapping_array[new_label] → new_node
5)   temp → list.head
6)   for i:=current_list_level down-to 1 do
     6.1) while temp.forward[i].key < new_key do
          6.1.1)    temp → temp.forward[i]
     6.2) update_array[i] → temp
7)   for i:=1 up-to height do
     7.1) new_node.backward[i] → update_array[i]
     7.2) new_node.forward[i] → update_array[i].forward[i]
     7.3) new_node.backward[i].forward[i] → new_node
     7.4) new_node.forward[i].backward[i] → new_node
```

Algorithm 2: Insert a new elementary process to the list with *new_key* and *new_label* as the key and label values respectively.

3.2.3. Implementation

The data-structures described above were implemented in *Zacros* (a Kinetic Monte Carlo software package written in Fortran 2003 for simulating heterogeneous catalysts), as alternative queueing systems for the *first reaction propagation method*. For further details on the structure of the KMC software package, we refer the reader to Ref. [63, 70]. Further to the algorithms already presented, a few language-specific additions were incorporated to make the data-

structure operational and efficient; nevertheless, the generality of the above algorithms remains unaffected.

3.3. Benchmark models, results and discussion

In this section, we compare the performance of the queueing systems presented in section 2, which were implemented in KMC software *Zacros*. In order to evaluate the efficiency of these data-structures as queueing systems, benchmark KMC simulations were performed for three simple, yet representative chemical reaction systems: the first two in stationary conditions and the third one in non-stationary conditions. The simulations were performed on an Intel(R) Xeon(TM) E5-1620 processor, running at 3.60 GHz, with 8GB of RAM, under the Ubuntu Linux operating system, version 18.04 LTS. The *Zacros* source code was compiled for serial execution (the OpenMP directives, although present, were not processed) using the GNU Fortran (GFortran) compiler, version 8.3.0.

We would also like to investigate the effect of compiler-induced optimisations on the performance of our queueing systems, because the data-structures we implement differ in their building blocks and compiler optimisations may be able to transform certain operations into a more efficient executable code. In particular, our data-structures are either array based (unsorted list and binary heap) or linked list based (pairing heap, 1-way and 2-way skip list), and compiler optimisations may have different effect on each one of them. Consequently, in order to assess and compare the inherent performance of the various queueing systems, two executables of *Zacros* were compiled, one with compiler optimisations off (option -O0 added during compilation) and the other one with optimisations on (option -O3). Although it is less likely that the unoptimised executable will be useful for production runs, the insights given might be helpful in validation of results, as well as in further development of our software or the compilers. Thus, in the rest of the section, we present the results obtained from both the optimized and unoptimised executables of *Zacros*.

3.3.1. Stationary simulations

3.3.1.1. Ziff-Gulari-Barshad model system

The first model used is a continuous-time adaptation of the Ziff-Gulari-Barshad (ZGB) system,[64] as discussed by Stamatakis and Vlachos (Section III.A of Ref. [63]). The ZGB is a prototype surface-reaction model for the oxidation of carbon monoxide on a catalytic surface represented by a rectangular lattice. The elementary steps taken into account in the ZGB system are as follows:

$$CO(g) + * \rightarrow CO^*$$

$$O_2(g) + 2* \rightarrow O^* + O^*$$

$$CO^* + O^* \rightarrow CO_2(g) + 2*$$

where * denotes a vacant site on the catalyst surface, CO* and O* denote adsorbed species, and (g) stands for gas. The partial pressures of CO and O_2 in the gas phase are taken as $P_{CO} = 0.525$ and $P_{O2} = 0.475$ respectively. Since the original algorithm by Ziff et al.[64] operates in discrete time, the kinetic constants were carefully chosen to be appropriate for the continuous time simulation used herein. Summarizing the discussion of Stamatakis and Vlachos,[63] the kinetic constants of the CO adsorption and the O_2 dissociative adsorption are chosen to be proportional to P_{CO} and P_{O2} as $10 \cdot P_{CO}$ and $10/4 \cdot P_{O2}$ respectively. The constant factor of 10 is arbitrary, and allows for adjusting the frequency of the events per unit time (rescaling of time). Thus, for the aforementioned values, CO adsorption would happen on average $10 \cdot P_{CO} = 5.25$ times per KMC time-unit for each empty lattice site. On the other hand, the 1/4 factor in the O_2 dissociative adsorption comes from the 4-fold coordination of each site on the rectangular lattice used for the simulations. Thus, for the given values, O_2 adsorption would happen $10/4 \cdot P_{O2} \cdot n_{en} = 1.1875$ times per KMC time-unit for each empty lattice site surrounded by n_{en} empty neighbouring sites ($0 \leq n_{en} \leq 4$). The kinetic constant of the CO oxidation and desorption from the surface is taken much larger ($1/4 \cdot 10^5$) than the other kinetic constants, so that the instantaneous oxidation assumed by Ziff et al. [64] is reproduced.

3.3.1.2. Water-gas shift reaction model

The water-gas shift (WGS) reaction produces carbon dioxide and molecular hydrogen from carbon monoxide and water vapour:

$$CO + H_2O \rightarrow CO_2 + H_2$$

It has been the subject of intense theoretical research for the past 30+ years, which has focused on its accurate modelling and on the identification of the rate-limiting step(s) and the optimal reaction conditions [65, 71-75]. For our second benchmark, we use a simplified variant of the WGS model on Pt(111) presented in Ref. [65]. The elementary steps taken into account are as follows:

$$CO(g) + * \leftrightarrow CO^*$$

$$H_2(g) + 2* \leftrightarrow 2H^*$$

$$H_2O(g) + * \leftrightarrow H_2O^*$$

$$H_2O^* + * \leftrightarrow OH^* + H^*$$

$$OH^* + * \leftrightarrow O^* + H^*$$

$$CO^* + OH^* \leftrightarrow COOH^* + *$$

$$COOH^* + * \rightarrow CO_2(g) + H^* + *$$

$$CO^* + O^* \rightarrow CO_2(g) + 2*$$

where * denotes a vacant site on the catalyst surface, starred species such as CO*, H*, H_2O* etc. denote adsorbed species, and (g) stands for gas. The partial pressure of the gas species, namely CO, H_2O, H_2 and CO_2, are taken as $P_{CO} = 1.0 \cdot 10^{-8}$, $P_{H2O} = 0.950$ and $P_{H2} = P_{CO2} = 0$ respectively. The procedure for calculating the rate parameters is described in detail in the “Mapping DFT Energies to *Zacros* Input” tutorial webpage of Ref. [76] while the numerical values used in our simulations are provided in Appendix I, section 3.

3.3.1.3. Computational details

The simulations were performed on periodic lattices of increasing size up to a total of about 10^6 lattice sites: starting from 20×20 up to 1000×1000 for the ZGB system, where rectangular lattices were used and from 10×10 up to 707×707 for the WGS model, where hexagonal lattices were used to represent the Pt(111) surface. For each different lattice size, the system being simulated

was propagated in time until the surface coverages of the dominant species were fluctuating around constant values, i.e. until stationary behaviour was attained. As the dominant species, CO* and O* were chosen for the ZGB model, and CO*, H* and H_2O* for the WGS model, Then, 10^6 KMC steps were further simulated and the simulation clock time needed to execute these 10^6 steps was recorded for post processing. Initially, for every lattice size, five simulations were run, one for each of our five queueing systems. Then, we ran the previous set of five simulations three more times to ensure consistency in our benchmark results. In total, twenty simulations were run for a given lattice size. Worth mentioning, the various queueing systems produced numerically identical results, proving the correct implementation of the data-structures presented as queueing systems for KMC simulations. Lastly, both models presented above do not incorporate lateral interactions between adsorbates, therefore, during the simulations, there were no updates to the occurrence times of the process queue. Consequently, from the queueing system perspective, the update-an-element operation was never invoked. We will refer to this remark later on during the discussion of our results.

3.3.1.4. Results and discussion

The results from the benchmark simulations appear in Figure 7 where the simulation clock time, viz. the real-time taken to execute the 10^6 KMC steps while on stationary conditions, is plotted against the total lattice sites, N_L. The top panels, Figure 7 (a) and (b), correspond to the runtime scaling of the ZGB system whereas the bottom panels, Figure 7 (c) and (d) correspond to the runtime scaling of the WGS reaction system. Both cases discussed above, with compiler optimisations disabled and enabled are presented in the left (Figure 7 (a) and (c)) and right (Figure 7 (b) and (d)) panels respectively, where each curve represents the average simulation time from four individual runs.

At the first glance, the benchmark results for the ZGB (Figure 7 (a), (b)) and the WGS (Figure 7 (c), (d)) reaction systems appear almost identical. Indeed, the ordering of the queueing systems is largely the same: the computational expense of the unsorted list scales very quickly with the size of the lattice, making this the slowest queueing system followed by the 1-way skip list, the 2-way skip

list, the binary heap and the pairing heap. As an exception to these similarities, the binary heap outperforms the pairing heap for the ZBG simulations in the presence of optimisations, as seen in Figure 7 (b). In addition, the differences in the runtimes (and the dissimilar y-axis time ranges) are to be expected, as the runtime is system dependent. As an early conclusion, our results indicate that the performance of our queueing systems is generic regardless of the chemical system under study, as long as the queueing system related dominant operations are the same. We elaborate on the last point in greater depth below. Let us now discuss the performance of each queueing system and compare our results with theoretical expectations.

Referring to Figure 7(a) and (c), we verify the expected poor performance of the unsorted list queueing system due to its linear scaling with system size, $O(N_L)$. Indeed, this queueing system, however simple in implementation, is extremely inefficient in KMC simulations, whereby the find-minimum operation (which is the inefficient one in this queueing system) is performed at every KMC step. For comparison purposes, we note that for the WGS system (Figure 7c), the unsorted list took 9658.6 seconds to complete 10^6 KMC steps on a 707×707 lattice, whereas pairing heap (the fastest queueing system) completed the same simulation in 17.2 seconds and the 1-way skip list in 38.8 seconds. Therefore, the speedups are 562× for the pairing heap and 249× for the 1-way skip list. Crucially, the latter exhibits logarithmic scaling with respect to the system size (compared to the linear scaling of the unsorted list). The developed 2-way skip list has logarithmic scaling as well, but is always faster than its 1-way alternative, especially for larger lattices. This time difference comes from the fact that the deletion operation takes constant time in 2-way skip list, in contrast to $O(\log_2 N_L)$ time in 1-way skip list. Furthermore, the binary heap and pairing heap share the same excellent performance, with the latter being constantly faster than the former by 4% in the ZGB and by 20% - 30% in the WGS. The constant time insertion of pairing heap has a crucial role in the observed performance, since it enables the pairing heap to outperform the array-based binary heap queueing system.

Apart from the unsorted list that exhibits linear scaling, and to which the following discussion does not apply, the other queueing systems exhibit logarithmic scaling with respect to the simulated system size. However, they differ in their multiplicative constant factor. The theoretical time complexities of the performed operations, summarized in Table 1, may clarify the observed performance. In both our benchmark chemical reaction models, only insertions and removals, but no updates, are executed during the simulation. Comparing binary heap and 1-way skip list, we observe that, while both have $O(\log_2 N_L)$ time scaling on the relevant operations, their actual runtime differs substantially. In a binary heap of size N, at every insertion we perform at maximum $\log_2 N$ key value comparisons against the parent of the inserted key value. On the skip list,

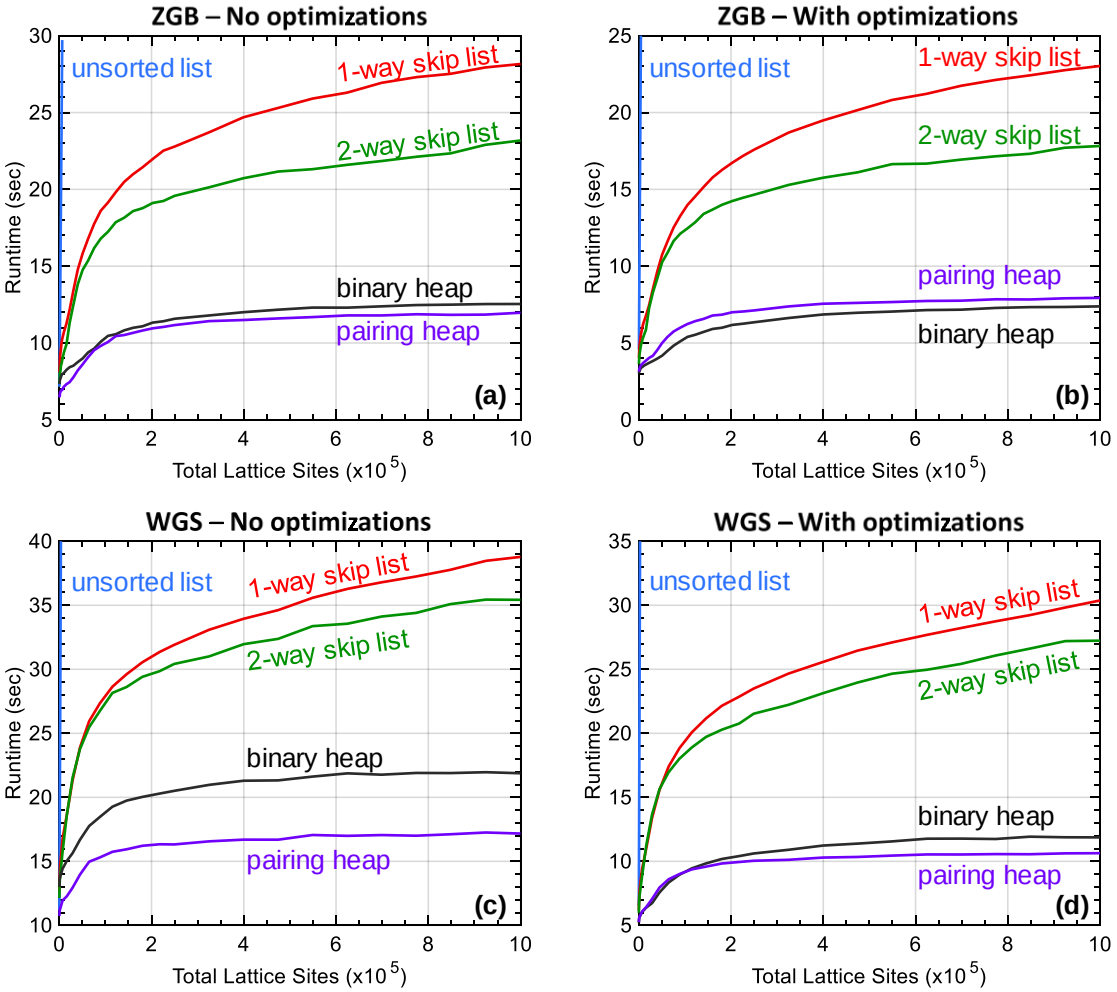


Figure 7: Runtime scaling of the various queueing systems for simulating 10^6 KMC steps while on stationary conditions in the ZGB (top panels) and the WGS (bottom panels) models, with compiler optimisations disabled (left panels) and enabled (right panels).

however, due to its probabilistic nature originating from the random assignment of levels to the nodes, the number of key value comparisons is at most *N*, the number of elements in the list. Nonetheless, this upper limit is not representative. What is more insightful, instead, is the average number of key value comparisons performed against existing values in the skip list, which is $3/2 \log_2 N + 7/2$, for a skip list with p=1/2, as reported by Pugh.[69] These extra key value comparisons are also performed during deletion on the 1-way skip list. Furthermore, the skip list is linked-list based, which means that its nodes are not likely to reside on a consecutive chunk of the computer's memory. On the contrary, binary heap, being array based, occupies memory that is always consecutive, and thus might benefit from lower-level acceleration schemes, such as memory caching. To summarize, the skip list's additional key value comparisons as well as its memory discontinuity incur computational overhead that accumulates over the large number of repetitions and largely contributes to the observed performance.

Considering now the results from the compiler optimized variant, shown in Figure 7(b) and (d) for the ZGB and WGS systems respectively, we observe that the runtime scaling seems similar, apart from two important features. First, the runtime, shown in the y-axis, has dropped significantly for all queueing systems. Second, the binary heap queueing system has outperformed pairing heap in the ZGB system. Let us discuss both in more detail.

Queueing system	Find minimum	Insertion	Removal	Update
Unsorted List	O(N)	O(1)	O(1)	O(1)
Binary Heap	O(1)	$O(\log_2 N)$	$O(\log_2 N)$	$O(\log_2 N)$
Pairing Heap	O(1)	O(1)	*$O(\log_2 N)$*	*$O(\log_2 N)$*
1-way Skip List	O(1)	*$O(\log_2 N)$*	*$O(\log_2 N)$*	*$O(\log_2 N)$*
2-way Skip List	O(1)	*$O(\log_2 N)$*	O(1)	*$O(\log_2 N)$*

Table 1: Expected computational time scaling with respect to the system size, N, of the various operations of all the queueing systems implemented. The expected computational times of pairing heap and both variants of skip list are amortized times (denoted by italics).

Enabling the compiler optimisations had the expected outcome: reduction of the runtime, which implies that the compiler utilized sophisticated techniques and algorithms to execute the same operations more efficiently. The unsorted list benefits from optimisations only when small and moderate size lattices are used. For lattices larger than 100×100 the runtime of the optimized version is almost the same as the unoptimised one. Moving on to the skip list queueing systems, we report that both are, at least, 1.3 times faster than before (i.e. without optimisations) for both the ZGB and WGS models. Lastly, heaps are the most favoured, since for the ZGB system, the speed up gain for pairing heap is 1.55×, whereas for the binary heap it is 1.75× (comparing panels (b) and (a) of Figure 7). For the WGS reaction system, the respective speed up gains for the pairing and binary heap are 1.65× and 1.85× (Figure 7 (d) vs (c)). Thus, in the presence of compiler optimisations, comparing the performance of 1-way skip list vs binary heap for the largest lattice size in the ZGB benchmark reveals an acceleration factor of 3.1× (Figure 7 (b)), while for the WGS benchmark, an acceleration factor of 2.8× is obtained for 1-way skip list vs pairing heap (Figure 7 (d)).

The most interesting result from the above benchmarks is the change in the ordering of the pairing heap and binary heap queueing systems, in the ZGB model, when compiler optimisations are enabled. Since no changes were made to our code, we attribute this result, i.e. that binary heap becomes the most efficient queueing system, exclusively to the optimisations. It is very likely that the compiler used in this study (GFortran v8.3.0) is more capable in delivering very efficient machine-level instructions when dealing with array-based data-structures, such as the binary heap. On the other hand, the pairing heap, being a data-structure that heavily relies on pointers, would not benefit from such array-level optimisations. The above explanation is also reinforced by the observed speed up gains as reported in the previous paragraph. In the case of the WGS, the initial separation of the binary and pairing heap curves is substantial, as seen in Figure 7 (c), so that when optimisations are applied, the pairing heap is still the most efficient queueing data-structure, as illustrated in Figure 7 (d).

To summarize, the ZGB and WGS reaction systems used for benchmarks require no updates of the occurrence times due to the absence of lateral interactions. Therefore, only the insertion, find-minimum and removal operations are executed during the simulation. When compiling with no optimisations, the

pairing heap queueing system was the most efficient one, followed by the binary heap. As for the optimized version, the binary heap, favoured by the compiler optimisations, becomes the most efficient queueing data-structure for the ZGB system, closely followed by pairing heap. In both versions, unoptimised and optimized, and in both benchmark systems, ZGB and WGS, the 2-way skip list is faster than its 1-way variant, showing that our development has had a positive impact on its performance. Nonetheless, the performance improvement was inadequate so that skip list could compete against the binary or the pairing heap.

Reiterating our early conclusions, and based on the above discussion, we note that the qualitative performance of our queueing data-structures is agnostic to the chemical system simulated, in the sense that it only depends on the execution frequency of the insertion, removal and find-minimum operations. In turn, a single execution of these three operations depends on the size of the data-structure holding the occurrence times of all realizable events. In its own turn, the total number of realizable events depends on (a) the total lattice sites and on (b) the number of elementary steps of the chemical model, where the latter is the only information that uniquely identifies the chemical system under study. Information directly related to a chemical system affects the runtime only indirectly via the dependence chain just described. Based on these arguments as well as the similarity of the runtime scaling of the ZGB and the WGS reaction models, it is reasonable to expect that the runtime scaling of the implemented queueing systems will be similar to the results presented above for all other chemical systems simulated, as long as these systems include no lateral interactions.

3.3.2. Non-stationary simulations

3.3.2.1. Temperature Programmed Desorption (TPD) model

We will proceed into discussing a set of benchmarks for a different chemical system, but let us first make a brief note on our motivation. The binary heap and pairing heap queueing systems store their elements partially sorted; on the contrary, both skip list variants are fully sorted. Also, the 2-way skip list has a

constant time removal operation. In order to make full use of these two properties, we sought a chemical system in which only event deletions happen, as we know that the 2-way skip list is very efficient for this: O(1) vs O($\log_2$N) for all other queueing systems.

To assess the previous rationale, KMC simulations were performed for a Temperature-Programmed Desorption (TPD) model of CO from a pure Cu (111) surface. The only elementary step considered for the TPD model is:

$$CO^* \rightarrow CO(g) + *$$

The partial pressure of the CO in the gas phase was set to zero in order to simulate ultra-high vacuum environment conditions. Since adsorption events are not taken into account, the reaction mechanism contains only the forward step of the above reaction. The rate constant of CO desorption is calculated from the Arrhenius equation parameterized by DFT calculations, which yield the adsorption energy and the vibrational frequencies for CO on the pure Cu (111) surface (refer to Section 2.3 by Darby et al.).[66] The temperature dependence of the pre-exponential factor of the rate constant is taken into account in the KMC simulation using the following fitted function of T:

$$A_{fwd}(T) = \exp\left[-\left(a_1 \log(T) + \frac{a_2}{T} + a_3 + a_4 T + a_5 T^2 + a_6 T^3 + a_7 T^4\right)\right] \qquad (3.3)$$

where T is the temperature in K, and α_1 - α_7 are the parameters listed in Table 2. The units of these parameters are such that the exponent of equation (3.3) is dimensionless and A_{fwd} is evaluated in units of s^{-1}. Lastly, the occurrence times of desorptions are calculated by solving a non-linear version of equation (3.1) with the Newton-Raphson method; a detailed discussion on how occurrence times are calculated when rate constants are time-dependent is given by Jansen in Ref. [44].

Parameter	Value
α_1	6.535
α_2	-2.762
α_3	-6.300×10^{1}
α_4	-7.076×10^{-2}
α_5	1.885×10^{-4}
α_6	-2.964×10^{-7}
α_7	1.957×10^{-10}

Table 2: Parameter values for fitting the equation (3.3)

To carry out the TPD simulations and represent the Cu(111) surface, we used hexagonal periodic lattices for which the unit cell contains two sites and each site has a coordination number (number of 1st nearest neighbors), of six. We studied lattices of increasing size, starting from 100×100 up to 1000×1000. At t = 0 s, the temperature is set to 100 K and the whole surface is covered by CO. During the simulation, the temperature ramp was set to 1 K s^{-1} for 100 s to a final temperature of 200 K and the coverage of CO* on the lattice surface was recorded at constant time intervals of 0.05 s. The clock time elapsed for the software to execute the necessary KMC steps was recorded and used as measure of comparison between the queueing systems tested. Similarly to the ZGB and WGS models, all the queueing systems produced numerically identical results, therefore, the difference in computational times is indeed a measure of their efficiency, since all the other operations in the simulation remain the same. At the post-processing stage, the TPD signal was calculated as the moving average of the instantaneous desorption rate, thereby enabling the determination of the temperature at which the CO desorption rate is maximized.

Our benchmark results from the TPD model are shown in Figure 8 and Figure 9 corresponding to the unoptimized and optimized versions respectively. Similar to the benchmarks under stationary conditions, we plot the simulation clock time against the total number of sites on the lattices used. Each curve in Figure 8 and Figure 9 represents the average simulation time from four individual runs. In addition, we normalize the simulation time, namely, we calculate the simulation time per 10^6 KMC steps and plot that quantity against the total lattice sites as shown Figure 10. The unsorted list is excluded from Figure 10 since its computational expense scales very rapidly and is out the range of that of the other queueing systems. The reason we normalize the running time of each queueing system in the TPD model is that the number of adsorbates on the surface is proportional to the KMC steps performed. Therefore, in the smaller lattices, the simulation executes less than 10^6 steps and using only the absolute simulation time (Figure 8 and Figure 9) would prevent us from being able to compare the performance between the stationary (ZGB, WGS) (Figure 7) and non-stationary (TPD) simulations.

A general observation drawn from Figure 8 and Figure 9 is that the scaling of all but the unsorted list queueing systems appears linear with respect to system

size, i.e. the number of sites on the lattices; we elaborate on the reason at the end of the current section. It is evident that, in both unoptimised and optimized variants, Figure 8(a) and Figure 9(a) respectively, the unsorted list scales rapidly and becomes extremely inefficient as compared to the other queueing systems. In the unoptimised variant, Figure 8, the two heap-based queueing systems exhibit the same performance. In addition, the two alternatives of the skip list, being faster than binary and pairing heap, require approximately the same time to complete a given simulation. The two-way skip list appear slightly faster than its one-way variant though and the difference in their runtimes increases for larger lattices.

Moving forward to the results in the presence of compiler optimisation (Figure 9), we observe that with compiler optimisations enabled and in accordance to our expectations, the performance of the queueing systems is improved. Furthermore, in line with previous observations on the ZGB and WGS systems, the compiler optimisations favour the binary heap over pairing heap. Indeed, both heap-based queueing systems exhibit the same performance before optimisations are applied (Figure 8 (b)), whereas after optimisations (Figure 9 (b)), there is a notable difference between their runtimes, with the binary heap being around 11% faster than the pairing heap. As for the skip list based queueing systems, we observe that their performance is similar and their ordering is

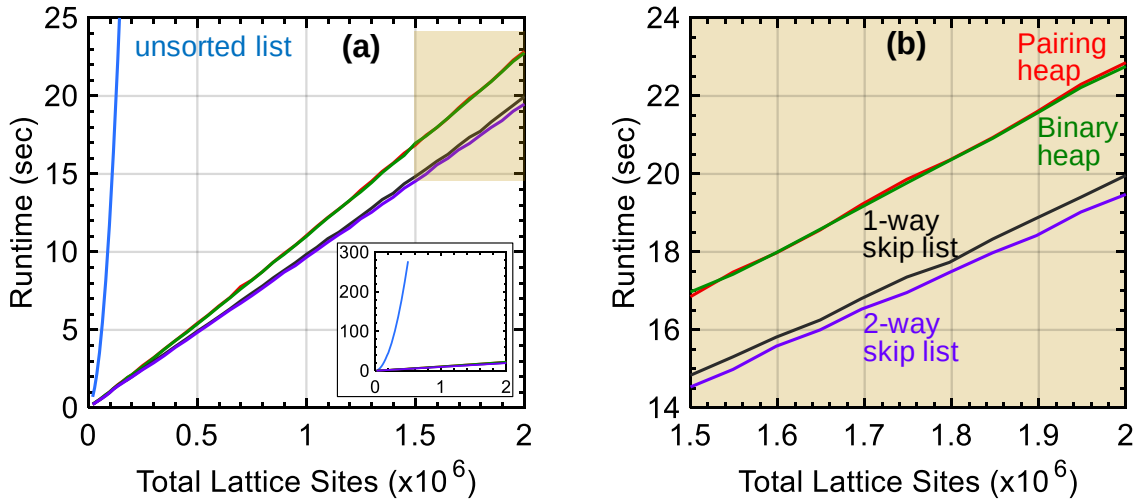


Figure 8: Benchmark results obtained from the unoptimised variant. The inset in **(a)** shows the $O(N^2)$ scaling of the unsorted list; its x- and y-axis represent number of sites and runtime respectively. **(b)** Magnification of the shaded area of (a). The curve labeling in (b) also applies to (a).

preserved, namely, the 2-way skip list is still slightly faster than the 1-way variant. However, for all practical purposes, their performance is identical and both benefit from a 1.5× speed up with respect to their unoptimised counterparts.

The normalized runtime with respect to the total lattice sites, illustrated in Figure 10, also verifies the above conclusions. The normalisation of the benchmark results clarifies the behaviour of our queueing systems especially in the region of smaller number of lattice sites. It is now clearer that the two-way skip list is consistently faster than the one-way skip list for the unoptimised variant (Figure 10(a)) whereas their difference in runtime becomes very small in the optimized one (Figure 10(b)). Equivalently, pairing heap and binary heap perform the same in the absence of compiler optimisations (Figure 10(a)) whereas they are well separated in the presence of optimisations (Figure 10(b)). In addition, the greater normalized runtime corresponding to small lattices indicates that the bottleneck i.e. the slowest operation of the *Zacros* software is not related to the queueing system. To better understand this, consider a TPD simulation on a fully covered 100×100 hexagonal lattice (20,000 sites). The maximum number of KMC steps that can be simulated is 20,000, equal to the maximum number of adsorbates on the lattice. The absolute simulation time of the above case using the binary heap (or any of the skip list-based) queueing system is 0.130 seconds while the corresponding normalized simulation time is 6.5 seconds. However, since the absolute simulation time obtained is very small, it is not representative

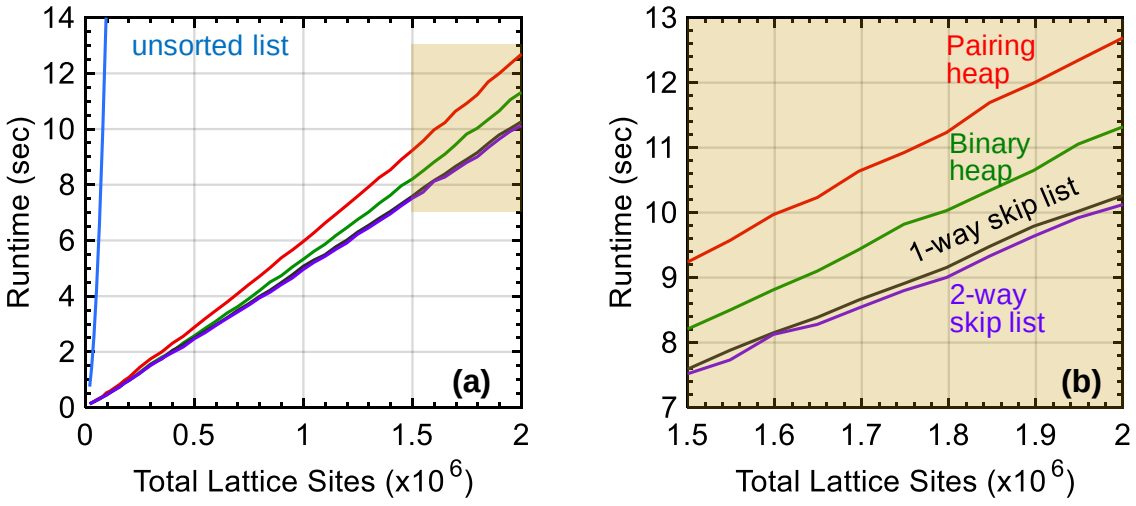


Figure 9: Benchmark results obtained from the compiler-optimized variant of Zacros. **(b)** Magnification of the shaded area of (a). The curve labeling in (b) also applies to (a).

of the queueing system because other internal procedures require more time to run and therefore, their contribution on the absolute simulation time dominates. On the other hand, for large lattices, the time required for all other procedures is becoming negligible as compared to the time of the queueing system related operations.

For the large lattices, the skip list-based queueing systems outperform the heap-based ones, a completely different outcome as compared to the ZGB and WGS results presented in section 3.3.1.1 and in particular in Figure 7. The TPD reaction model benefits from the fact that the skip list stores all inter-arrival times fully sorted, the minimum one is located in the beginning of the queue and no other elementary events are added during the simulation. Therefore, the entries in the queue of realizable events are retrieved one after the other, with a small number of pointer reconnections necessary to ensure the queue's proper connectivity. On the contrary, the binary heap stores the inter-arrival times partially sorted. After every event execution, namely finding the minimum and removing it from the queue since the corresponding event has occurred, the algorithm restores the partial order on the heap by performing element swaps (refer to section 1 of Appendix I for a schematic representation of the process). In the case where the queue contains a very large number of elements, like the TPD model on large lattices, these frequent swaps incur an overhead on the simulation that accumulates and becomes measurable as shown by the "Binary heap" curves in Figure 10. The same principle applies to the pairing heap as well:

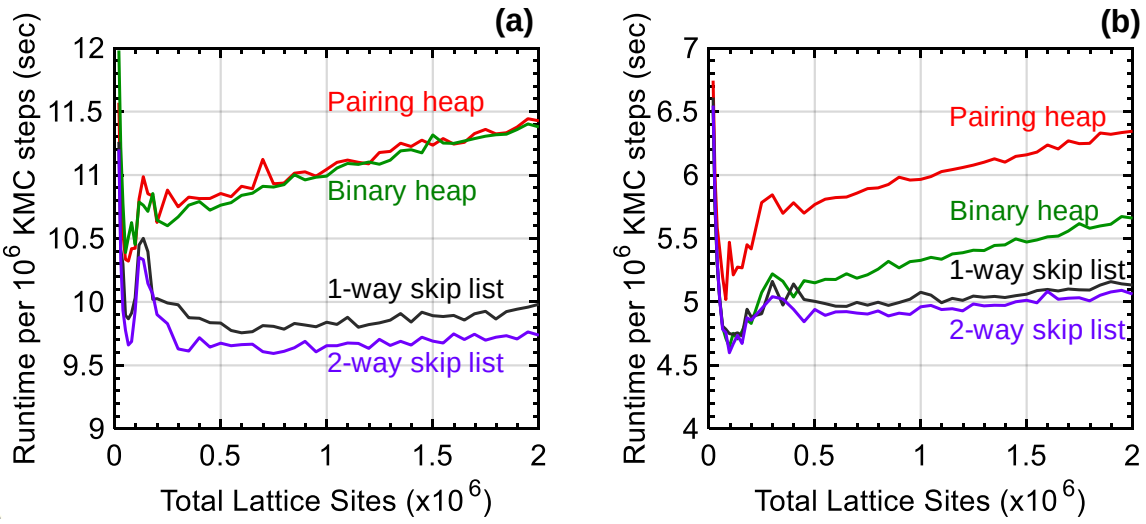


Figure 10: Normalised runtime corresponding to the unoptimized **(a)** and the optimized **(b)** variant of Zacros.

following the execution (removal) of an event, the pairing heap has to be rebalanced via operations that are responsible for the observed behavior shown by the curves labelled “Pairing heap” in Figure 10. It is also reasonable to assume that the rebalancing procedure on the pairing heap is not optimized at the same level as the swap procedure on the binary heap. Therefore, the former dominates the runtime and results in the pairing heap having the worst performance as compared to the two skip list-based queueing systems and the binary heap (Figure 9 and Figure 10(b)).

Let us now discuss the observed scaling based on the theoretical time complexity of each queueing system. From the event scheduling perspective, only desorptions occur during the simulation and no other elementary events are added in the queue since adsorption is not considered. Therefore, once the simulation is initiated and every event is inserted in the queue, only the find-minimum and remove-minimum operations are executed at every KMC step. In a lattice with N sites there will be a total of N adsorbates to desorb, hence, N KMC steps are executed. In the unsorted vector the pair of operations “find-minimum”-“remove-minimum” takes O(N) time and is executed N times. Consequently, we expect that the unsorted vector scales quadratically, $O(N^2)$. Indeed, our benchmark results verify the expected scaling (inset in Figure 8(a)). As for the binary and pairing heap queueing systems, the pair of operations mentioned above takes $O(\log_2 N)$ computational time. The simulation terminates after the execution of N desorption events, equivalently after N executions of “find-minimum”-“remove-minimum”, and hence the expected time taken is $O(N \log_2 N)$. This scaling behavior mostly resembles the linear scaling (refer to Appendix I, section 4, for a comparison) which is the one we obtained from our benchmarks, in particular, Figure 8(a) and Figure 9(a). Lastly, the “find-minimum”- “remove-minimum” pair of operations takes constant time, O(1) in both skip list variants. That pair is executed N time, therefore, the expected scaling is linear O(N) which is what we observe in both Figure 8(a) and Figure 9(a).

Upon normalisation of the runtime, presented in Figure 10, the scaling behavior changes. Binary and pairing heap are expected to scale as $O(\log_2 N)$, whereas in Figure 10 their scaling is mostly linear. It is very likely that we have to go to even larger lattices in order to see the logarithmic scaling, just like in either of the skip list curves in Figure 7(b) where a linearly increasing part precedes the

slowly increasing part that compose the entire logarithmic function. As for the skip list-based queueing systems, the expected scaling is O(1), namely constant, since retrieval of the minimum element and its deletion does not depend explicitly on the size of the list. The skip list curves of Figure 10 exhibit a slight increase, however. This is due to the removal operation and more specifically due to the reconnections of the nodes that are bounded by H_N in the one-way skip list and by $2\times H_N$ in the two-way skip list, H_N being the expected height of the skip list. The latter depends on the size of the skip list though (equation (2)) which makes our observation consistent with the theoretical analysis.

The advantage of running TPD simulations in very large lattices is portrayed in Figure 11 where the normalized TPD signal is plotted against the surface temperature. Since the signal is calculated as the derivative of the number of CO molecules desorbed from the surface with respect to time, the greater the number of molecules that desorb per unit time the better the obtained resolution is. Fine resolution is even more important in the presence of multiple desorption peaks, in which case the correct identification of the temperature at which the desorption rate is maximum, depends strongly on the quality of the simulated spectrum.

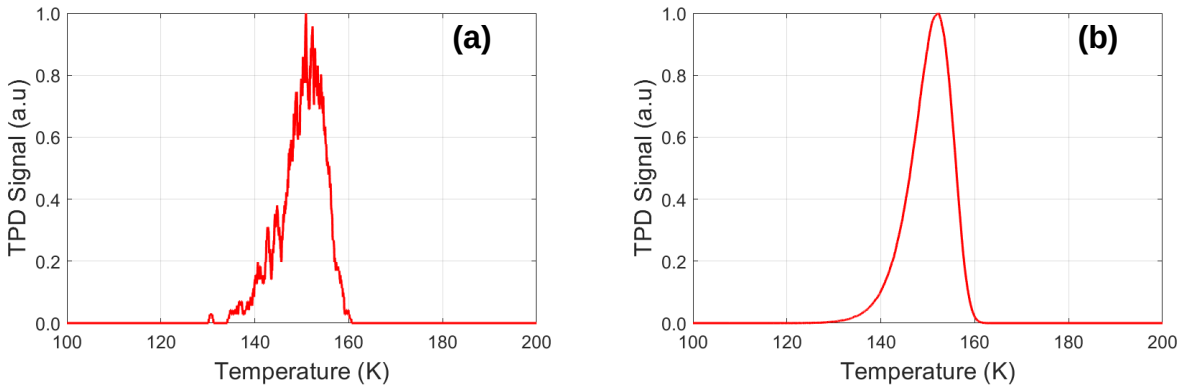


Figure 11: TPD signal from **a)** 20×20 and **b)** 600×600 hexagonal lattice

3.4. Conclusions

In this study, we compared various data-structures as queueing systems for the *first reaction* method of the KMC algorithm and we further developed a skip list based queueing system. In order to compare their performance in common ground, we implemented these five approaches in the *Zacros* software package and performed benchmark simulations using a CO oxidation model adapted from the ZGB study [64], a simplified WGS reaction model on Pt(111) based on Ref. [65] and a TPD model of CO on Cu(111) [66]. We also studied the effect of compiler optimisations on the computational time required to complete the benchmarks.

The unsorted list was found to be extremely slow, as expected, and thus unusable for production simulations. The binary heap and the pairing heap have the best performance in the ZGB and WGS models, which involve insertion and deletion of elements in the queue. Our data indicates that for such simulations, the heap-based methods could be viewed as equivalent with each other, as their relative efficiency depends on the system and the compiler optimisation level. Regarding the skip list-based methods, even though the 2-way skip list consistently appeared to be slightly more efficient in our ZGB and WGS tests, the computational times are quite comparable with each other, so the skip list-based approaches could also be viewed as equivalent. We could potentially be more assertive in stating that the skip list-based queueing systems are less efficient than the heap-based ones, something that makes intuitive sense, since the skip list maintains a fully ordered list of elements, whereas the heap data-structures can only ensure partial ordering. In addition, the heap data-structures, especially the binary heap, appear to be more amenable to compiler optimisations. Still though, skip lists could outperform heap-based queueing systems, e.g. for special cases of systems in which all events are scheduled in the beginning and are executed in that predetermined order. Our TPD model falls precisely into this category and therefore, the skip list queuing systems demonstrated superior performance for these simulations.

The 2-way skip list developed herein is used as a queueing system during a KMC simulation; its usability, however, extends far beyond the applications

considered in this work. Any application that involves maintaining a large number of entities in a sorted manner with frequent access to arbitrary elements of this set may take advantage of our proposed design. In particular, choosing to adopt the 2-way skip list would leverage the conditional O(1) element retrieval, the O(logN) generic search and the O(1) element removal operations.

Any *first-reaction* method KMC implementation is, in one way or another, based on the following core operations: event identification and scheduling of realizable events, and execution of the most imminent process. Since KMC is becoming increasingly popular as a tool for understanding and predicting catalytic performance, optimal implementations for each one of the core KMC operations is of paramount importance. In our work, we focused on the efficient event scheduling, thereby evaluating existing data-structures and further developing a novel one (the 2-way skip list) for our purposes. The current work addresses the lack of benchmark studies of not-so-common data-structures for KMC implementations in the scientific literature. Moreover, our benchmarks could serve as a guide for further development of KMC approaches and related software aiming at high efficiency. While these approaches were benchmarked in heterogeneous catalysis problems, they could be applied equally well to other KMC frameworks, simulating for instance diffusion in porous media,[77] or intracellular reactions occurring in biological systems [45, 78].

4. Evaluating and optimising the performance of distributed KMC simulations

Part of the work presented on this chapter has already been published. In accordance with the policy of the copyright holder, the relevant material (adapted text and figures) that appear in this chapter is *Reproduced with permissions from Ravipati et al., Comput. Phys. Commun. 270, 108148 (2022) Copyright 2022, Elsevier.*

4.1. Introduction

KMC simulations have had a significant contribution towards understanding and predicting the dynamic properties of materials [77, 79-81]. Among the different fields, heterogeneous catalysis has widely adopted on-lattice KMC simulations where the catalytic surface is represented by a suitable lattice as discussed in more detail in Chapter 3. The increasing popularity of on-lattice KMC is also evident by the increase in software codes that implement such methods. Some examples of popular KMC software is Zacros [50, 51], SPPARKS [52, 53], KMCLib [54], and kmos [55], to name a few. Depending on the complexity of the chemical system under study in terms of number of reactions and adsorbate-adsorbate lateral interactions, either short- or long-range, the KMC simulation can be very computationally intensive. The latter complexity, in combination with the serial nature of KMC [82] limits the KMC simulation to small lattices, on the order of a few tens of nanometers [17, 83].

Improving the computational efficiency of KMC simulations is an active area of research and tends to focus on tackling specific needs, for example addressing the timescale disparity [84-86], or on generic algorithms and implementations that improve certain procedures [32, 49, 87-89] without affecting the accuracy. One such example has already been presented in Chapter 3, where different data-structures have been implemented to undertake the scheduling and execution of elementary events during the KMC simulations. Despite these

developments, KMC simulations on larger lattices still face issues in terms of tractability due to their computationally intensive nature.

KMC simulations on small but representative lattices provide invaluable insights and even explain experimentally observed phenomena [90]. However, when the surface phenomena involve formation of patterns or spiral waves [91], the size of the computational lattice that is required to capture such phenomena is dictated by the length of the patterns or the wavelength of the spirals. For example, in the spirals reported by Nettesheim and coworkers [91], the wavelengths are on the order of a few μm. Due to the larger scale in which those patterns evolve, KMC simulations on the conventional nanometer-scale lattices cannot reproduce the experimental observations. The limiting factors to attempting such large-scale simulations are related to the computational resources, namely memory and processing power required to obtain meaningful results. Since the serially executed KMC implementations are unable to simulate systems in which patterns form on the μm scale, other means have to be sought to make such simulations possible.

To enable large-scale KMC simulations, one has to resort to distributing the workload to multiple processing units. Various approaches and methodologies have been developed to enable the execution of Kinetic Monte Carlo simulations in parallel [30, 31, 92-94]. Conceptually, these methods involve domain decomposition and algorithmic protocols to execute the elementary events in such a way that there are no conflicts on the boundaries of the subdomains. The latter is achieved via either synchronisation or reverting back and resimulating the history that is invalidated because of the boundary conflicts. In the latter approach, each processing unit generates a KMC trajectory for its subdomain and once all those trajectories are validated as consistent up to a specific point in time, they are all registered to the global history that corresponds to the entire computational lattice.

In terms of software implementations for catalysis, SPPARKS [52, 53] includes the semi-rigorous synchronous sublattice algorithm [92]. In addition, SPOCK [95] includes an exact parallel KMC implementation based on the Time Warp paradigm [31]. The downside of the implementation in SPOCK is that it is not generic and the user needs to provide system-specific code when building custom models. Also, both implementations mentioned above, lack validation

procedures that would verify the correctness of the implementation. To address the lack of a generic, parallel KMC implementation that can be validated, Ravipati and coworkers [33] have coupled the optimistic Time-Warp algorithm with the graph-theoretical KMC framework of the Zacros software package.

The implementation and all technical details on the Time-Warp implementation are covered fully in the relevant publication [33]. This chapter focuses on the performance evaluation and performance optimisation of the implementation, which is the main contribution of the author's thesis on the latter work. Therefore, after a brief introduction on the main working principles of the Time-Warp algorithm, the results are presented, followed by the discussion.

4.2. Methodology

Based on the Time-Warp paradigm, the main idea behind distributed KMC simulation is to decompose the entire lattice domain into smaller, non-overlapping, subdomains, each one of which is assigned to a different processing unit (PU) or element (PE). A PU or PE is conventionally a CPU core. Each unit executes, independently from other units, the usual KMC method as applied to its subdomain and communicates the necessary events with its neighbours. Therefore, all events that happen away from the boundaries are executed asynchronously by each processing unit, whereas the events at the boundaries are communicated and handled appropriately. Due to the asynchronous nature of the execution of events and the random time advancement in KMC simulations, each processing unit has its own simulation time, and thus causality violations may occur, as will be discussed in more detail shortly. For the KMC trajectory to be exact, these violations need to be resolved in an algorithmically robust manner. The Time-Warp algorithm provides the required "machinery" so that exact KMC simulations can be performed in a distributed manner.

In this algorithm, each PU propagates the system in time broadly independently from other units. However, communication is required between PUs when an event happens on shared domain boundaries. For instance if PU1 executes an event at a boundary shared with PU2, the latter neighbouring unit needs to have knowledge about this event, and thus, an appropriate message is

sent from PU1 to PU2. If the receiving process, PU2, is behind in time as compared to the process who sent the message, PU1, handling the message is straightforward. The message is put into a queue and appropriate action is taken when PU2 reaches that particular KMC time. What is not so straightforward, however, is when the sender, PU1 is behind in KMC time as compared to the receiver, PU2. Suppose that the message about an event that had just occurred on the timeline of PU1 is communicated to PU2. That message has a timestamp of t_1. However, PU2 is ahead in KMC time and therefore the message is sent to its past. More importantly, the history of PU2, from the time t_1 onwards is wrong because the communicated event is not taken into account. The only way to make the history consistent is to revert back and resimulate it. To do so, all processing units need to have snapshots available along their history so that they can revert to them when needed. Therefore, as the PUs propagate the system on their subdomain, they take complete snapshots of their entire simulation state and store them in the memory. Snapshots are necessary to resolve causality violations and are saved at regular intervals during the course of the KMC simulation.

The situation described above is even more complicated if PU2 had sent messages to other units after KMC time t_1 before the causality violation occurred. Since the history of PU2 between t_1 and t_2 is now being resimulated, any actions performed by other PUs as a result of such messages need to be corrected. Then, PU2 needs to send to those units what is called an anti-message, that encodes an “undo” action to a previously sent message. The PUs that receive these anti-messages may have to revert back as well and correct their history (this would happen if the timestamp of an anti-message is smaller than the current KMC time of the receiving PU). A schematic representation of causality violations involving three processing units is illustrated on Figure 12. The causality violations and the rollback procedure are likely to produce a cascade of violations that could affect the entire domain. However, such cases are system-dependent and are rare in practice. In any case, the Time-Warp algorithm provides all the procedures necessary to resolve such conflicts and ensure the consistency of the simulated history.

To enable rolling back in KMC history, all PUs need to save snapshots in memory. Since memory is limited, the simulation needs a procedure to free up

memory and delete the snapshots that are no longer needed. The latter involves global communications. At regular intervals, all PUs communicate with each other to determine which one has the smallest KMC time, t_s. Assuming that there are no pending messages, the history of all PUs is consistent up until t_s, and therefore, any snapshots taken before t_s may be safely deleted. The time t_s until which the KMC history is correct and consistent is called Global Virtual Time (GVT), and it is a useful metric to quantify the progress of the KMC simulation and decide when the simulation can be terminated safely. More information and technical details on the implementation can be found in the original work [33].

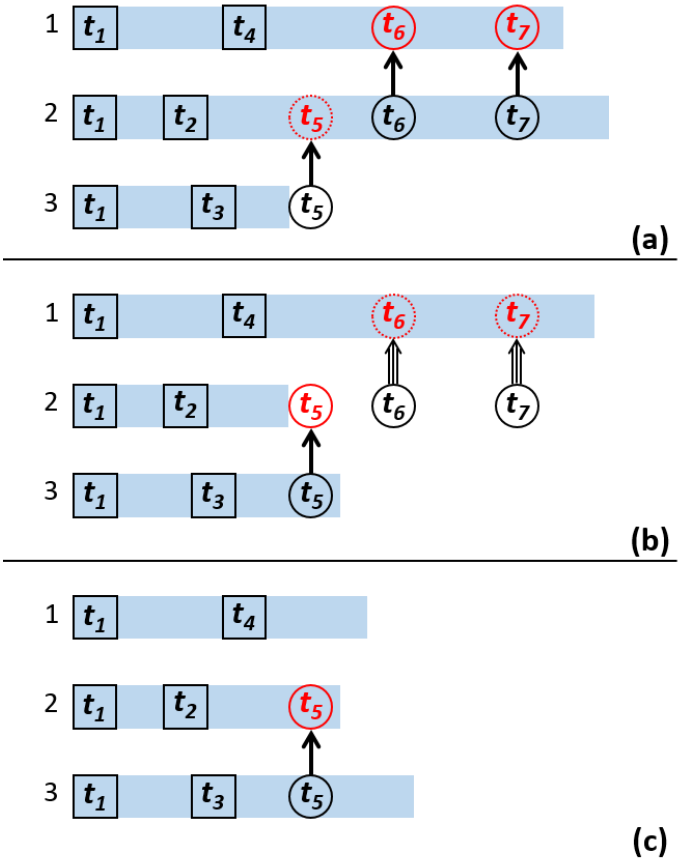


Figure 12: Schematic procedure of causality violations and of the way they are resolved. The blue rectanges represent KMC timelines; black squares represent snapshots saved; circles with inward and outward arrows represent received and sent messages, respectively; triple arrows represent anti-messages. **(a)** PU2 receives a message with timestamp t_5, which is at its past. **(b)** PU2 reverts back to time t_2 using a saved snapshot and resimulates the history until t_5. PU2 sends anti-messages corresponding to the previously sent messages. **(c)** PU1 receives the anti-messages and reverts back to t_4 using a saved snapshot.

The last important detail that needs to be covered is what happens if the available memory is filled-up before a global communication takes place to remove the no-longer-needed snapshots. In the latter scenario, internal procedures of the Time-Warp algorithm are invoked to: **(a)** sparsify the queue in which snapshots are saved, namely delete every other snapshot to free memory, and **(b)** double the interval over which snapshots are taken so that the queue does not fill up again after the GVT is calculated. If needed, the above procedures are invoked again to ensure that the frequency of saving snapshots is appropriate, given the available memory.

The Time-Warp algorithm, as briefly introduced above, includes a few tunable parameters that eventually determine its performance. These are **(a)** how often the PUs are saving snapshots and **(b)** how often the global communication takes place to determine the GVT, the smallest KMC time among all PUs. The former is called Snapshot Saving Interval (SSI) and is measured in KMC steps. Practically, it denotes the number of KMC steps after which each subdomain saves a snapshot into the snapshot queue. Saving snapshots too often, for example, every 10 KMC steps, results in filling the memory too quickly, in which case sparsifications occur and the initial SSI is doubled, here to a value of 20. However, frequent snapshot saving means that there is always a snapshot in the recent past and, in case of causality violations, the PU does not spend much time resimulating a substantial part of its KMC history. From the hardware point of view, frequent reads and writes in the memory incur overheads that decrease the performance of the algorithm. Therefore, as a generic guideline, it is not advised to save snapshots too often. On the other hand, if the value of the SSI is large, namely, the snapshots are saved much less frequently, for example, every 1000 KMC steps, the memory requirements are decreased significantly. However, more time is spent on resimulations when causality violations occur because the most recent suitable snapshot is likely to be further back in the past. Referring to Figure 12(b), when PU2 needs to roll-back in time because of the message with time-stamp t_7, it uses the snapshot taken at time t_4. If snapshots were saved less frequently, the most recent snapshot for all PUs could be at e.g. time t_1. If reverting back to t_1 instead of at t_4, PU2 would need to spent more time to resimulate the history from t_1 to t_7 and then take appropriate action for the

received message. Likewise for PU1, upon receiving the anti-message with time-stamp t_8, as shown in Figure 12(c).

The second parameter, the interval after which a global communication takes places and the GVT is calculated is termed GVTi and is measured in real-time seconds, for example 30 seconds of clock-time. As already mentioned, global communication is crucial to free up memory by deleting the snapshots that are no longer needed. Performing communication too often, e.g. every 1 second, might incur overheads that would slow down the simulation. Conversely, infrequent global communications, e.g. every 120 seconds, necessitate much more memory available to save the snapshots of each PU.

4.3. Chemical Reaction systems

To better understand the scaling of the newly implemented Time-Warp algorithm in Zacros, we perform a scaling analysis of the algorithm on two simple, yet representative systems and present our results in the next section.

- **System 1** includes CO adsorption, CO* desorption and CO* diffusion, without lateral interactions.
- **System 2** includes CO adsorption and CO* desorption with 1st nearest-neighbour lateral interactions among the adsorbed CO*.

The reactions in both systems are considered reversible and are summarised below:

$$CO_{(g)} + * \leftrightarrow CO^*$$

$$CO^* + * \leftrightarrow * + CO^*$$

where (g) stands for gas, * denotes a vacant site, and a superscript * denotes an adsorbed species on the surface. The rate constants of adsorption and desorption were both taken as 1.0 s^{-1} whereas the diffusion rate constant was taken as k_{diff} = 10.0 eV. Exclusively for system 2, lateral interaction between adsorbed CO* molecules is taken as ε_{CO^*} = 0.1. Simulations were run at 500 K and at 1 bar pressure.

The above systems where chosen such that the coupling between the neighbouring domains is due to different factors. In system 1, the coupling comes

from the diffusion of the adsorbed CO* that results in CO* crossing the boundaries and diffusing to a nearby domain. In system 2, the coupling comes from the pairwise interaction between the adsorbed CO* molecules. If there was no diffusion in system 1 or no interactions in system 2, the simulation would be "embarrassingly parallel", namely, it would be ideally parallelisable because the subdomains are not coupled.

In addition to the two simple toy models presented above, we make use of a third, more complicated system with rich dynamic behaviour and pattern formation.

- **System 3**, is an on-lattice adaptation of the so-called Brusselator system [96], which exhibits autocatalytic behaviour and includes the following 8 reactions:

Elementary Event	k_{fwd}(s^{-1})	k_{fwd}/k_{rev}
$X + * \leftrightarrow X^*$	0.70	0.91
$2X^* + Y^* \rightarrow 3X^*$	3.80	-
$X^* \leftrightarrow Y^*$	9.00	15.00
$X^* + * \leftrightarrow * + X^*$	4.00×10^2	1.00
$Y^* + * \leftrightarrow * + Y^*$	4.00	1.00
$X^* + Y^* \leftrightarrow Y^* + X^*$	4.00×10^2	1.00

Table 3: Reactions of the on-lattice Brusselator system

No interactions were included among the adsorbates on the lattice and the simulations were run at a temperature of 500 K and at a pressure of 1 bar. To observe the patterns formed by system 3, we had to use a much larger lattice than system 1 and system 2. More information is provided in the relevant subsection below.

4.4. Scaling Benchmarks

To assess the performance of the Time-Warp implementation, we perform benchmarks in terms of weak- and strong-scaling. To ensure consistency in the results, we initially simulated our systems (1 and 2 only) until stationary state is reached and used this as the initial state for our benchmarks. Since much larger lattices were used in the benchmarks, we tiled the initial stationary state to create an input state suitable for the lattices used in the benchmarks. Specifically for system 3, we run a long simulation until the spirals were well developed and then we used a representative snapshot as the initial state for the benchmarks. Starting from stationary state is necessary because, for these benchmarks, we would like the average number of KMC steps executed per one KMC time unit to remain constant.

As already discussed, the user-defined parameters are the SSI and the GVTi. In addition, the user specifies the memory that is available to the software exclusively for saving snapshots (S_{mem}). We summarise the values used in Table 4. These settings were chosen after a small number of trials and a detailed study on them is provided in the next subsection.

System	Scaling	MPI configuration	S_{mem} (GB)	SSI (# events)	GVTi (s)
1	Weak	1×1 2×2 - 30×30	4.0	10^9 (i) 100	5
	Strong	2×2 - 20×20		100	
2	Weak	1×1 2×2 - 30×30	3.5	10^9 (i) 50	3
	Strong	2×2 - 20×20		50	

Table 4: Settings used in the performance benchmarks. In the strong-scaling, the single-processor runs were serial, thus the 1×1 MPI configuration is missing. (i) the snapshot saving is practically switched off in the serial runs.

To quantify the scaling performance of the simulations, we define the scaled time, t^* as the KMC time divided by the clock time:

$$t^* = \frac{t_{KMC}}{t_{clock}} \tag{4.1}$$

We also define $t^*(n_{sites} : n_{PU})$ as the scaled time of a distributed run of a lattice with total sites n_{sites} that is distributed over n_{PU} processing units. For serial runs, the n_{PU} is omitted.

4.4.1. Serial runs

The first part includes serial runs executed in progressively larger lattices, measuring their performance and comparing the latter against the expected scaling law. Ideally, the scaled time, t*, would be inversely proportional to the number of lattice sites. The efficiency is defined as:

$$\eta = \frac{t^*(n_{sites})}{t^*(n_{sites}^{min})} \tag{4.2}$$

where n_{sites}^{min} is the number of sites of the smallest lattice used in the serial runs. The latter is a 100×100 lattice with 10,000 sites. The results are shown in Figure 13(d-e), with open, red circles, which represent the efficiency as defined by the relation (4.2). The red lines on the same subplots represent the ideal scaling of the efficiency as:

$$\eta = \frac{n_{sites}^{min}}{n_{sites}} \tag{4.3}$$

For system 1, Figure 13(d), the serial runs scale as expected up to lattices of size 500×500 and the efficiency deteriorates for larger lattices. For system 2, Figure 13(e), the actual performance deviates from the ideal one starting from the lattice of size of 200×200 sites. The difference in the behaviour observed between system 1 and system 2 is attributed to the additional workload that has to be executed in system 2 to compute the rate constants of events because of the pairwise lateral interactions among the adsorbed CO* species. The latter operation includes a pattern detection procedure and updating the relevant occurrence times; both operations incur overheads and slow down the KMC run as a result.

4.4.2. Weak scaling

The idea underlying the weak scaling analysis is to increase the size of the domain proportionally to the number of processing units so that the workload per PU remains constant. For the weak scaling benchmarks, the efficiency is defined as:

$$\eta_{WS} = \frac{t^*(n_{sites} : n_{PU})}{t^*(n_{sites}^{min})} \tag{4.4}$$

First, we compare the single-unit parallel run with the serial run for the same lattice size (leftmost point on Figure 13(d-e)). In the parallel run, although the snapshot taking is switched off, as per Table 4, and causality violations cannot happen, the MPI-related subroutines of the Time-Warp algorithm are still called, as also done during truly distributed runs (run on several PUs). These overheads cause a slowdown in the simulations by 30-40%.

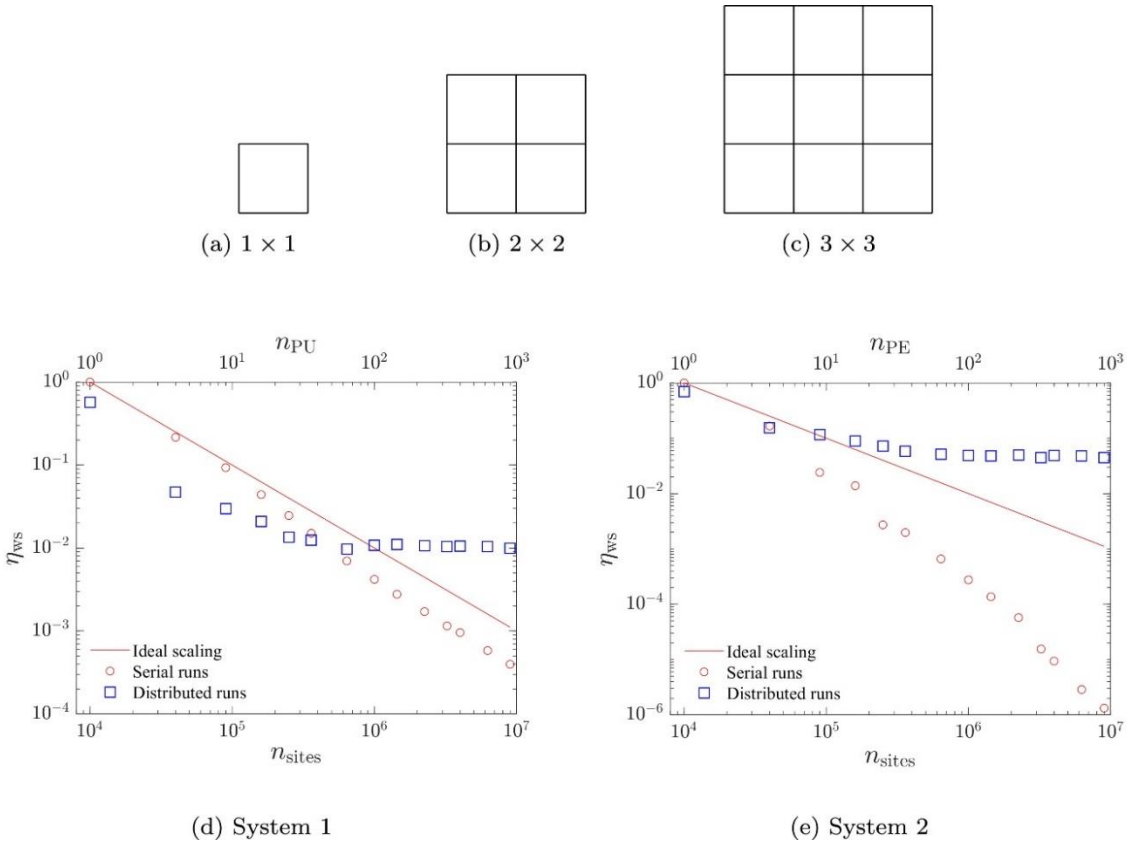


Figure 13: (a-c) Schematic representation of how the lattices change in the weak scaling benchmarks: the PUs increase along with the size of the lattice so that the workload per PU is constant. Our base lattice, the 1×1 in (a) is of size 100×100. (d-e) Serial and weak scaling results for systems 1 and 2 respectively.

A truly distributed run is performed using 4 PUs on a 200×200 lattice for both systems. The drop in efficiency with respect to the serial run is attributed to the emergence of communication and the procedures invoked to resolve causality violations. As the lattices become larger and larger, in particular, for lattice sizes of 800×800 and beyond, the distributed runs outperform the serial ones. In addition, the efficiency of the distributed runs remains constant whereas the serial runs are becoming more inefficient. For the largest lattice we used, of size 3000×3000, containing 9 million sites, the distributed run for system 1 is ~25 times faster than the corresponding serial run.

In system 2, Figure 13(e), the difference between the serial and distributed runs is more pronounced. For the largest lattice used, as reported above, the distributed run is more than four orders of magnitude more efficient than the serial run. The great difference in the performance is attributed to the distribution of the additional workload: the pattern detection and occurrence time updates due to lateral interactions are happening “locally” for each subdomain as part of the execution of elementary events and are parallelised efficiently by the Time-Warp algorithm.

4.4.3. Strong scaling

In the strong scaling analysis, the size of the domain remains constant while the computational resources increase. As a result, the work per processing unit becomes smaller and smaller. A schematic representation of the procedure is shown in Figure 14(a-c). For the strong scaling benchmarks, the efficiency is defined with respect to the serial run as:

$$\eta_{SS} = \frac{t^*(n_{sites} : n_{PU})}{t^*(n_{sites})} \tag{4.4}$$

Our strong scaling benchmark results are illustrated in Figure 14(d-e) for three different sizes as the base lattice (Figure 14(a)). A general observation is that, for a truly distributed run, i.e. with more than just one PU, the efficiency increases as more PUs are used. The point at which the distributed runs outperform the serial ones is system- and lattice size-dependent. In system 1, when 100 processing units are used, the distributed runs perform better than the

serial runs for all the lattices used. In system 2, all distributed runs, with 9 PUs or more, exhibit better performance than the serial ones.

It is also interesting to observe the plateau in the efficiency when a large number of PUs is used. As already mentioned, the more PUs are used, the smaller each subdomain becomes. As a result, the parallelisation overheads become dominant when the workload is not substantial enough to be executed by a PU. In addition, the ratio of the boundary sites over the internal sites increases which, in turn, increases the communication among the neighbouring PUs and the probability of causality violations. The efficiency plateau is not observed for the other two lattices size used, namely the 900×900 and 1440×1440. It is expected, however, that, for a high enough number of PUs, the strong scaling efficiency, η_{SS}, will drop after going through a maximum.

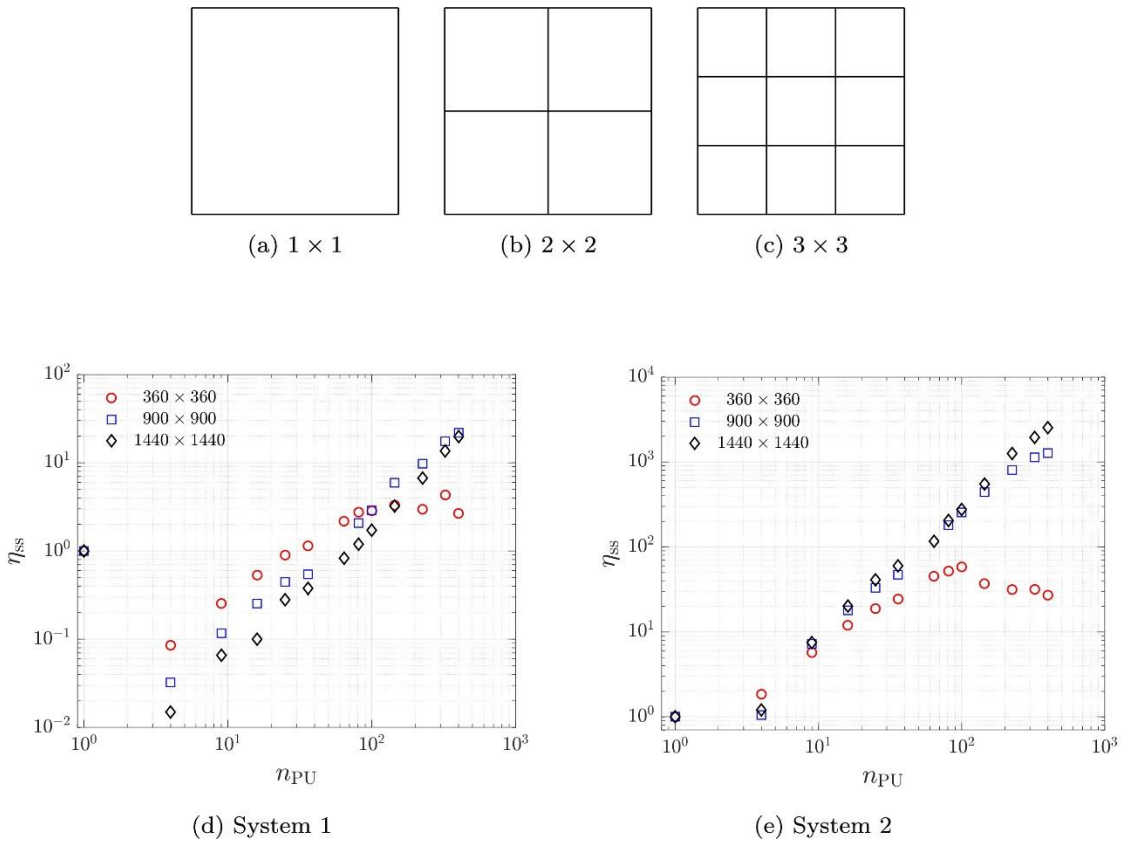


Figure 14: (a-c) Schematic representation of how the subdomains change in the strong scaling analysis: the workload per PU is progressively smaller. (d-e) Strong scaling results for systems 1 and 2 respectively.

4.5. Performance Investigation & Optimisation

The benchmarks of the previous section were performed for fixed values of the SSI and GVTi, for each system. These values were chosen after a few trial runs and are reported in Table 4. However, it should be clear from the discussion in the introduction, section 4.1, and the methodology, section 4.2, that the optimal values of SSI and GVTi are interlinked and depend on the chemical reaction system. In this section, we thus perform detailed investigations to understand the coupling between SSI and GVTi and their effect on the performance of a KMC simulation and present our results. We use all three systems presented in section 4.3, and investigate a wide range of values for both SSI and GVTi. For the present analysis, our metric of performance is the inverse of the scaled time defined by equation (4.1): we define the *simulation cost rate*, ρ, as the real time (clock-time) needed to propagate the system by one KMC time unit.

$$\rho = \frac{t_{clock}}{t_{KMC}} \tag{4.5}$$

Given the above definition, the lower ρ is, the better.

Our results, for system 1, on a domain of 200×200, distributed over just 4 PUs (so that each PU gets a subdomain of 100×100), are shown in Figure 15. In Figure 15(a), we plot the simulation cost rate, ρ, against the GVTi, and the different values of SSI appear as different curves, while in Figure 15(b), we plot ρ against SSI and the different values of GVTi appear as different curves. In practice, since we calculate ρ when two parameters change, we get a 3D plot, and the panels in Figure 15 are the 2D projections. The most obvious conclusion drawn from Figure 15(a), is that the value of GVTi does not affect the performance, given that there is enough memory available to avoid doubling the user-provided SSI. On the other hand, as suggested by Figure 15(b), the value of SSI is crucial since a poor choice can make the simulation even three times more costly as compared to the optimal performance. As expected, too small and too large values of SSI deteriorate the performance. The optimal values for SSI that deliver the best performance range from 50 to 100 as seen by the broad minimum of Figure 15(b) and by the lower-most curves of Figure 15(a).

All the simulations performed were given access to the same amount of memory. The simulation in which SSI was set to 5 had the higher memory

utilisation, since a snapshot was saved every 5 KMC steps. To avoid running out of memory, thereby having to double the initial SSI, the global communication must be done more frequently. This is the main reason behind the small number of points in Figure 15(a), top-most curve, that corresponds to SSI=5. When the parameters were chosen as SSI=10, GVTi=10, the memory was not enough to hold all snapshots and therefore, sparsification was occurring and the SSI was doubled to a value of 20. To ensure that our results are consistent and representative, all those cases (where sparsification occurred) are omitted from the plots.

In Figure 16, we present our results, for system 1, on a domain of 1200×1200, distributed over just 12×12=144 PUs (so that each PU gets a subdomain of 100×100). The 200×200 domain, used to produce the previous results, is distributed over 4 PUs, namely 4 CPU cores, and as a result, a single a computational node of the UCL-provided cluster *Thomas* was enough to execute each simulation. The 1200×1200 domain, however, is distributed across 6 computational nodes, each one containing 24 CPU cores, therefore, our results might contain the effect of cross-node communication, if there is any. The results of Figure 16 are very similar to those presented in Figure 15. For a given SSI, the GVTi does not seem to affect substantially the performance of the simulation. The best range of values for the SSI for optimal performance is again over the range of 50 to 100. The difference is that the simulation cost rate is higher in the case of the large lattice, 1200×1200, as compared to the small one, 200×200. This is likely to be due to the increased number of causality violations because there are

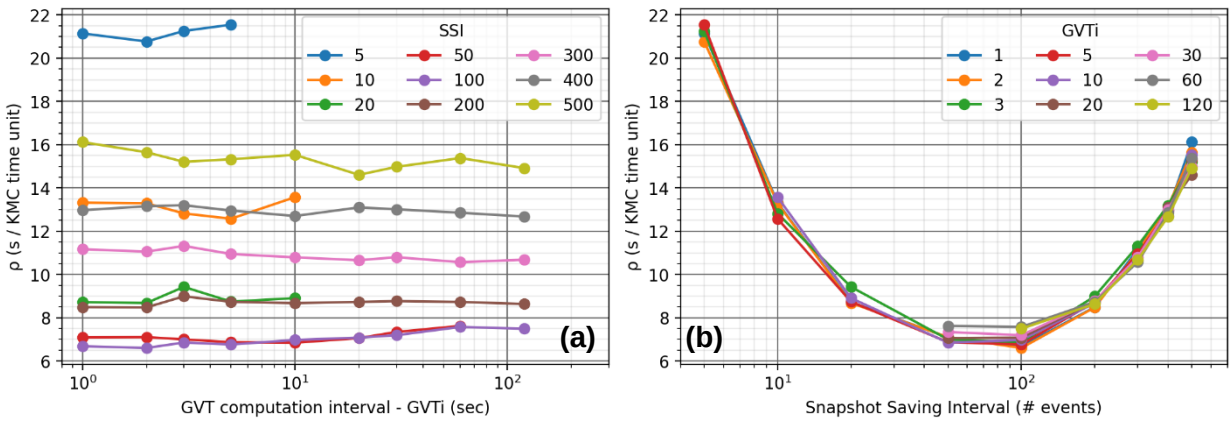


Figure 15: Simulation cost rate, ρ, against the GVTi (a) and against SSI (b) of system 1, for a 200×200 domain distributed over 4 PUs.

more subdomains and because of the cross-node communication overheads. The latter is not easy to quantify, it would require detailed profiling of the code, which is out of the scope of this investigation.

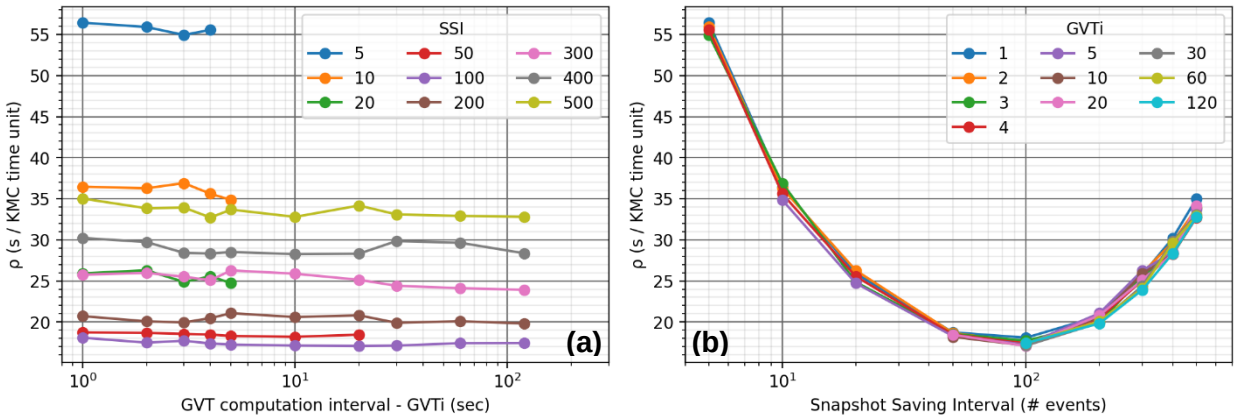


Figure 16: Simulation cost rate, ρ, against the GVTi (a) and against SSI (b) of system 1, for a 1200×1200 domain distributed over 144 PUs.

Moving on to system 2, the system that includes pair-wise lateral interactions among the adsorbed CO*, we perform simulations for multiple combinations of SSI and GVTi for two different lattice sizes, namely, 200×200 and 1200×1200. The results that correspond to the domain of size 200×200, distributed over 4 processing units, are shown in Figure 17. Similarly, the results that correspond to the largest lattice of size 1200×1200, distributed over 144 processing units, are shown in Figure 18.

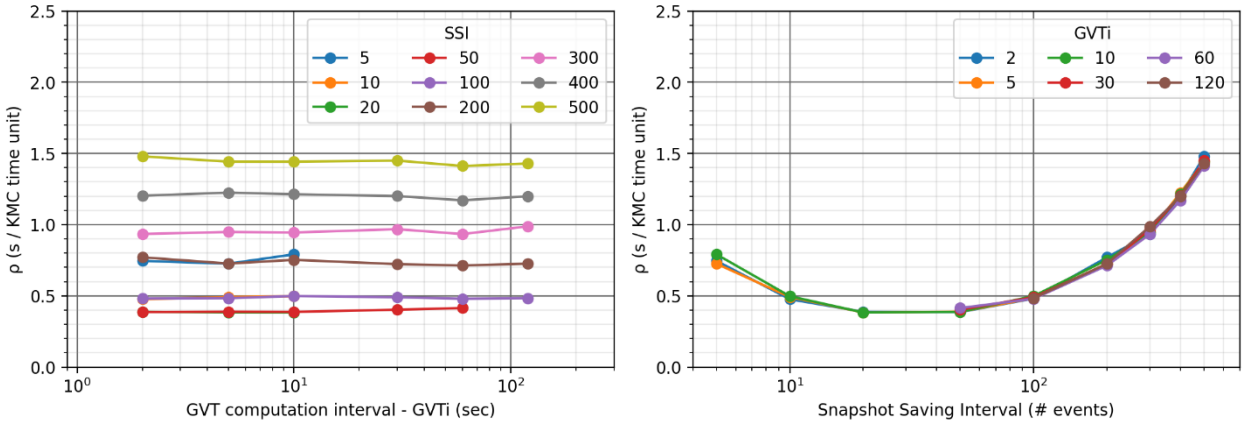


Figure 17: Simulation cost rate, ρ, against the GVTi (a) and against SSI (b) of system 2, for a 200×200 domain distributed over 4 PUs.

The performance of the Time-Warp algorithm for system 2 is similar to that already presented for system 1, namely there is little variability with respect to GVTi, especially in Figure 17(a), whereas the SSI is the parameter that determines the performance. As observed in system 1 and discussed above, the simulation cost rate, ρ, is higher for the larger lattice, 1200×1200, Figure 18, as compared to the smaller one, of size 200×200, Figure 17, for a given set of (SSI, GVTi) values. Despite depicting similar trends, Figure 17 and Figure 18 have different scales on the y-axis. A notable difference from the previous results, Figure 15(b) and Figure 16(b), is that the optimum values for SSI cover a wider range, from 20 up to 100, with the marginally better efficiency delivered by SSI=50. Also, comparing Figure 16(b) and Figure 18(b), we conclude that simulating system 2 is more efficient than simulating system 1. This is in line with the weak scaling results shown in Figure 13(d-e), where the efficiency drop due to the parallelisation is much less in system 2 than in system 1. In addition, SSI values on the higher end of the range used result in worse performance that SSI values on the lower end of the range. This is expected and is related to the structure of system 2, which includes pairwise lateral interactions among the adsorbed CO* species. The presence of interactions increases the workload because in every step, the KMC algorithm, regardless of whether it is distributed or not, has to check the neighbours of the species participating in the just-executed reaction, and update accordingly the kinetic rates in which these species participate. When snapshots are saved less frequently, i.e. SSI is larger, then, the snapshot to be reinstated in case of a causality violation is likely to be

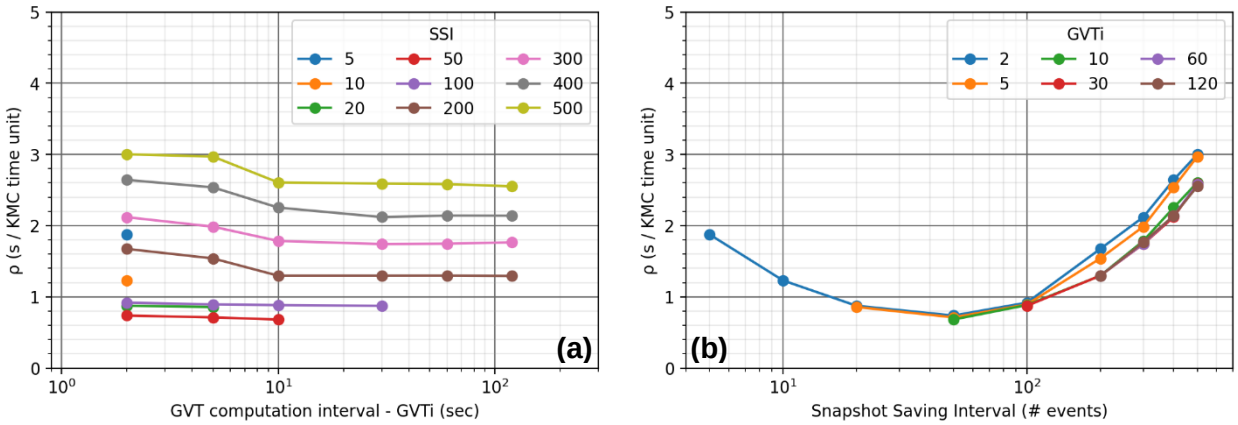


Figure 18: Simulation cost rate, ρ, against the GVTi (a) and against SSI (b) of system 2, for a 1200×1200 domain distributed over 144 PUs.

further in the past. Therefore, more time is spent in resimulating part of the KMC history. Since every KMC step is more computationally expensive in the presence of interactions, the additional simulation-related workload becomes more than the overhead of the frequent memory accesses when SSI is smaller.

The last system for which we performed this performance analysis is system 3. We used a domain of 4000×4000 and start with an initial state in which the spirals of the Brusselator system are well developed (Figure 19, left panel). Due to the much larger size of our lattice, we distribute the simulation over 25×25=625 processing units, so that each one of them is assigned a subdomain of 160×160 sites. Even with this more complex system, our results, shown in Figure 20, are similar, in terms of overall trends, to what has already been presented for the two simpler model systems. Due to the greatest complexity of the Brusselator system, even the most efficient set of parameters, i.e. SSI=150, the simulation requires 4500 sec (75 mins) to propagate the system for just one KMC time unit. In contrast to Figure 18(b) where the most inefficient runs are those with SSI values in the higher end of the range used, in Figure 20(b), the worst performance is observed towards the smallest SSI values. The latter is due to the much larger memory overheads incurred while saving and deleting snapshots, mainly because the snapshots themselves are large since the subdomain is of size 160×160.

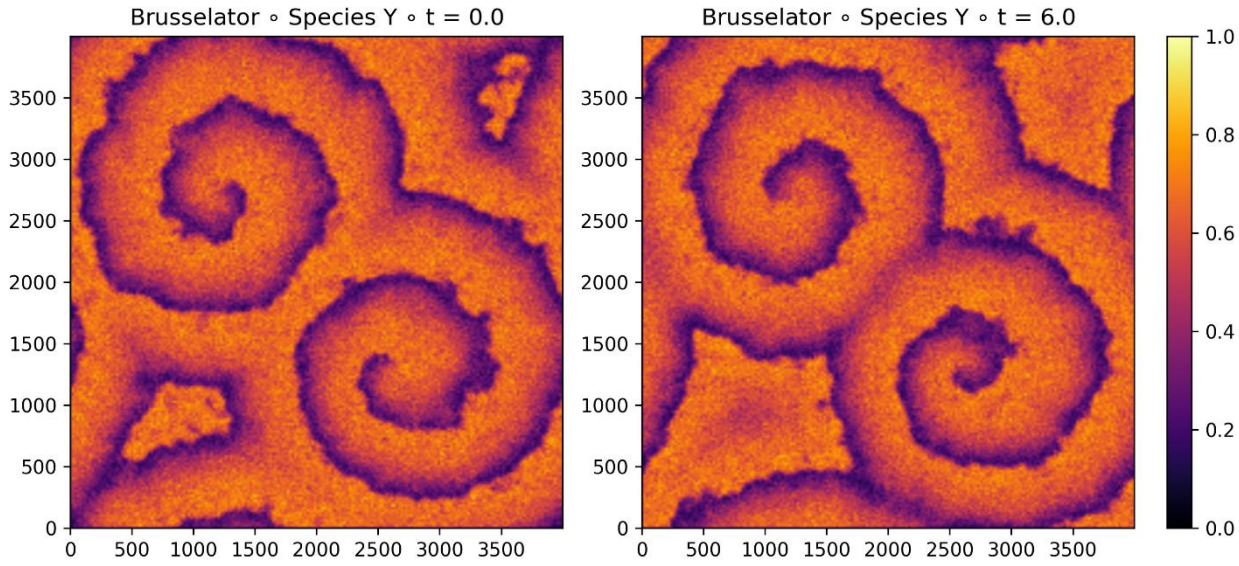


Figure 19: Surface density of species Y illustrating the evolution of the spirals at t_{KMC}=0 (left) and t_{KMC}=6 (right).

In Figure 19, we present two snapshots of the state of the lattice for system 3. Figure 19(left) illustrates the initial state, at t_{KMC}=0. By the length of the arms of the two spirals evolved, it is obvious that we could not have obtained these results by just using “conventional size” lattices. In addition, these results could not be produced without the distributed Time-Warp algorithm that decomposes the initial domain into smaller subdomains, here of size 160×160. If we were to use the serial KMC algorithm on the 4000×4000 lattice, the simulation would have taken an extremely large amount of memory and time to produce the results shown in Figure 19.

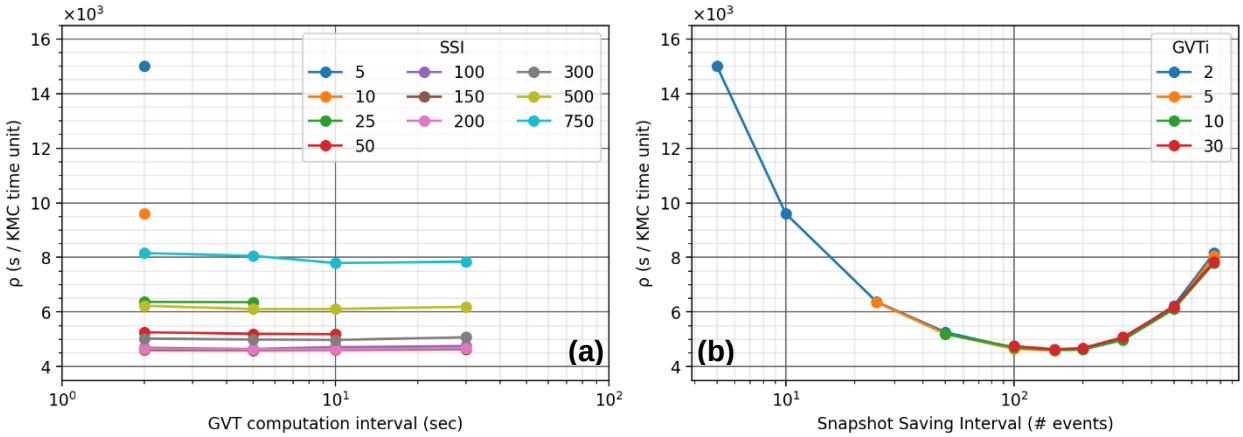


Figure 20: Simulation cost rate, ρ, against the GVTi (a) and against SSI (b) of system 3, for a 4000×4000 domain distributed over 625 PUs

The detailed investigations just presented aimed at better understanding the effect of the user-defined parameters, SSI and GVTi, on the performance of the Time-Warp algorithm. We performed simulations covering a wide range of values for both parameters. The SSI determines how often snapshots are saved in the memory whereas the GVTi determines how often global communication occurs. Given a fixed amount of memory, the two parameters cannot take arbitrary values: small values of SSI require equally small values of GVTi so that the obsolete snapshots are deleted. Otherwise, the simulation will run out of memory and it will adapt the SSI by doubling its value. The most important finding of the above investigation is that, as long as there is enough computer memory available, the value of the GVTi does not affect the performance of the KMC simulation. On the other hand, the performance is affected, sometimes heavily

as in Figure 20(b), by the value of SSI. Our results also suggest that the optimum SSI value, or range of values, is system-dependent:

- system 1: $SSI_{opt} = 100$
- system 2: $SSI_{opt} \in [20, 50]$
- system 3: $SSI_{opt} \in [100, 200]$

In addition, slight variations are observed when the domain size changes.

Based on the above findings, to achieve the maximum performance for an arbitrary system, the user might be required to perform a small number of exploratory runs and “construct the performance landscape” with respect to the SSI, as shown in Figure 15(b) - Figure 20(b). For the GVTi, a reasonable value is in the range of 20 - 60 seconds, while making sure that the memory is enough to accommodate the snapshots saved, especially towards the largest GVTi values. The user may, then, proceed with the production runs, after identifying the set of values for SSI, GVTi that deliver the optimum performance, namely achieving the minimum wall time for advancing their system for one KMC time unit.

4.6. Conclusions

In this study, we evaluated the scaling of the Time-Warp algorithm [31] as implemented on the Graph-Theoretical Kinetic Monte Carlo framework [27] of the software package *Zacros* [51]. The Time-Warp algorithm is a sophisticated methodology that enables the exact, distributed execution of KMC simulations. The main idea is to decompose the domain (lattice) into smaller subdomains, and assign each one of them to a different processing element. Events are executed “locally” for the internal sites of each subdomain, and events that involve boundary sites are communicated to the neighbouring subdomains. Since each subdomain has its own timeline, causality violations may occur when, for example, one subdomain that is “lagging behind” in KMC time, tries to send a species via diffusion to a nearby domain that is further ahead in time. The KMC history of the domain that receives the particle is no longer valid and needs to be

corrected. This is achieved by saving snapshots along the KMC simulation and restoring the most suitable one when such conflicts arise.

Using two simple systems, we investigated the scalability of the Time-Warp implementation in Zacros. For the weak scaling benchmarks, we observe that the distributed simulations are slower than the corresponding serial ones for lattices up to 600×600 for a system without lateral interactions. For larger lattices, the distributed implementation becomes faster, and, most importantly, there is no further decline in efficiency. For a system with lateral interactions, any truly distributed simulation, i.e. excluding the 1×1 MPI configuration, is progressively faster than the serial one, reaching efficiency factors up to 34×10^3 for lattices of size 3000×3000. For the strong scaling, we observe that, for a high enough number of processing units, both of our benchmark systems are more efficient than the corresponding serially executed ones.

In the second part of this study, we investigated thoroughly the dependence of the performance on two user-defined parameters: the SSI, how often snapshots are saved and the GVTi, how often all processing units communicate. For this study we used three systems, system 1 and system 2 that were used for the scaling benchmarks, and system 3 as the more complex system studied. We concluded that the GVTi does not affect the performance, whereas the optimal value for SSI depends on the system being simulated.

5. Tackling the timescale disparity on well-mixed chemical reaction systems

5.1. Introduction

Given a set of interacting species or molecules and the elementary reactions occurring between them, one is able to derive differential equations that fully describe the system using the law of mass action. The solution of the latter equations, obtained analytically or numerically, provides a complete description of the evolution of the system in time, given the initial conditions. Ordinary differential equations (ODEs) have been extensively used to study models of spatially homogeneous systems of particular interest in biology with the most notable example being the enzymatic reactions proposed by Michaelis and Menten [97]. However, an inherent weakness of this approach is its inability to take into account stochastic effects. In systems with low populations of molecules, stochastic noise is important, and in fact, determines the time evolution of the system by giving rise to macroscopic effects [98-100]. In addition, bistability effects, such as those observed in the Schlögl model [101], cannot be captured by treating the system deterministically, namely, by solving the differential equations that govern the system [102].

Kinetic Monte Carlo, and more specifically the Stochastic Simulation Algorithm (SSA), proposed by Gillespie [43], is a powerful computational tool that enables the study of interacting molecules and reaction intermediates in a well-mixed vessel [47] or over a surface (on-lattice KMC) [17] while incorporating the inherent randomness of such systems. The trajectory produced using the SSA for a specific system is consistent with and, in fact, is a sample path (realisation) of the underlying Chemical Master Equation (CME) [103]. KMC is an exact method in the sense that it does not introduce any approximation and the time evolution of the system studied depends solely on stochastic effects and on the propensity, i.e. the probability per unit time for a reaction to occur [43]. In turn, the latter depends on the current state of the system, namely, the species populations, and on the rate constants of the reactions simulated. When

compared to the traditional reaction rate equations based approach, KMC is so powerful because it simulates explicitly and sequentially, i.e. one at a time, every single reaction that occurs in the system.

However, the benefits of KMC come with significant increase in computational cost. In systems with several species and multiple reactions, the accessible timescales might be too short to allow meaningful conclusions to be drawn. This issue is especially acute in systems in which some species exist in high populations. In addition, reactions may occur on different timescales, i.e. some reactions might be very frequent, while others very rare. Since the reactions are executed one at a time, the KMC simulation would be dominated by the fast reactions, whereas the slow ones, which might be of interest and contribute to the evolution of the system, are not executed and not sampled adequately. The scenario just described gives rise to an issue often termed “timescale disparity” or “stiffness” in a KMC simulation.

Multiple approximate acceleration methods have been developed throughout the years to tackle the timescale disparity in well-mixed chemical systems. A prominent such method is the τ-leap developed by Gillespie [104]. The main idea of this method is to “leap over” many fast reactions in a single step and advance the KMC time accordingly instead of executing all these events one at a time. The main assumption of the method is that the propensities of the fast reactions do not change significantly during the time leap. However, this assumption breaks down when species appear in small quantities, typically less than 10 molecules, which makes the method unusable for systems with low population species. Nevertheless, various refinements have been developed for the tau-leap method aiming to improve efficiency [105, 106], address stability issues [107] and prevent negative populations [108-110].

Another approach to tackle timescale disparity is to develop hybrid methods [111, 112] that combine the reaction rate equations with a discrete stochastic treatment of the system. The first step is to partition the reaction network into fast and slow reactions. Then, to reduce computational cost, the fast reactions are approximated in a deterministic way or via Langevin equations and the slow reactions are simulated as stochastic events using an implementation of the SSA [111]. This approach achieves a bound on the computational cost by sacrificing accuracy since all fast fluctuations are eliminated, which might be

acceptable depending on the nature and dynamics of the system studied [111]. Building upon the previous work, Cao and coworkers developed a systematic approximate theory [113] and then presented the Multiscale Stochastic Simulation Algorithm (MSSA) [114] which assumes partial equilibrium for the fast reactions and works efficiently for low population species. Deterministically, partial equilibrium takes the form of algebraic constraints that the quasi-equilibrated species concentrations must always satisfy. Stochastically, partial equilibrium means that the probability distribution(s) over the corresponding states must be at quasi-steady state. However, the distribution over the partially equilibrated states is, in general, not known, and not easy to obtain [114].

Other methods [115-119] use perturbation analysis along with the quasi-steady-state approximation to reduce the CME, eliminate fast reactions from the reaction network and use the SSA to obtain realisations consistent with the reduced CME. This approach is shown to perform well in systems presented in the aforementioned studies. However, according to perturbation analysis, the reduced solution converges to the solution of the full model as the perturbation parameter, usually denoted by ϵ, approaches zero, or when it becomes "sufficiently small". In general, ϵ is used to quantify the ratio of the slow versus the fast kinetics, and $\epsilon \to 0$ implies that the fast kinetics are much faster than the slow ones. In reality, ϵ is obtained by scaling appropriately the parameters of the model, and as a result, the order of magnitude of ϵ that ensures convergence depends on other parameters of the model.

In practice, to tackle timescale disparity and reduce the computational cost of KMC simulations, one may manually reduce the rate constants of fast reactions in such a way that the dynamics of the system at the slow timescale remain unaffected. To achieve the latter, one usually performs a "convergence study" of a quantity of interest against the rate constant(s) that cause the chemical reaction system to become stiff. Initially, the rate constants of the fast reactions are aggressively reduced, aiming to distort the dynamics of the system. Then, the same rate constants are gradually increased (not to exceed the original values, though) until key indicators, such as the production rate of a species, have converged to a specific value. When convergence is observed, it is implied that further increase of the rate constants would have no effect on the dynamics of

the system. Finally, the reaction rate constants for which the key indicators appear converged are used for production runs that are much cheaper computationally than using the original rate constants.

Similar to the approximate accelerated methods summarized above, the manual reduction of reaction rate constants does not provide any metric on the approximation error introduced by the reduction procedure. Although the procedure of reducing the rate constants of fast reversible reactions has physical meaning when the quasi-equilibration condition holds, there are no guarantees that the latter action has not incurred a change in the dynamics of the system under study. Aiming to address the lack of an accuracy index on the available methods, we have developed an on-the-fly downscaling scheme with an error metric that quantifies the effect of the rate constants reduction on the accuracy. The method we propose generates multiple trajectories with different rate constants and quantifies the computational cost and error. Based on the latter, the rate constants of the fast reactions are reduced in an optimal way, by taking into account the potential speedup gain while minimizing the error introduced.

The rest of the Chapter is organized as follows. First, we introduce briefly the background of the methods we use and make the connection to our work. The implementation of our on-the-fly rates scaling method along with the necessary theoretical background is discussed in detail in Section 5.2. The results obtained by applying the proposed methodology on a well-mixed system are presented in Section 5.3, along with the discussion of the performance and applicability of our approach. Finally, Section 5.4 summarises the motivation and main points of this work.

5.2. Methodology

In this section, we first provide the necessary background on the relevant kinetic Monte Carlo methods and we discuss other relevant methods such as the Common Reaction Number and Common Reaction Path. Subsequently, we proceed by presenting in detail the developed methodology for scaling on-the-fly the rate constants of interest.

5.2.1. Background

The stochastic simulation algorithm (SSA), originally derived by Gillespie, is an exact method of simulating numerically the stochastic time evolution of spatially homogeneous mixtures of interacting species, often termed as “well-mixed systems”. Since the introduction of two variants of this method by Gillespie, additional approaches and further improvements have been proposed. In the following subsections, we present briefly the relevant KMC methods and highlight features thereof that are further used in our work. In addition, we introduce two widely used methods in parameter sensitivity analysis, namely the Common Reaction Number and Common Reaction Path and make the connection to our work later on.

5.2.1.1. First Reaction Method (FRM)

The First Reaction Method is one of the two procedures proposed by Gillespie [43]. It is based on the intuitive idea that, given a set of feasible reactions along with their inter-arrival firing times, the reaction to occur next is the one with the smallest waiting time. Therefore, for every possible reaction i, a tentative reaction time τ_i is generated via:

$$\tau_i = -\frac{ln(r_i)}{a_i} \tag{5.1}$$

where r_i is a uniformly distributed random number in the unit interval and a_i is the propensity function of reaction i. The reaction μ that occurs next and its inter-arrival time τ_μ are both determined by finding the minimum among all the reaction waiting times τ_i:

$$\tau_\mu = \min(t_i) \tag{5.2}$$

Then, the KMC clock is advanced by τ_μ, and the reaction μ is executed by updating the molecular populations accordingly. New tentative reaction times are generated using equation (5.1) and the procedure described above is repeated.

For every iteration, the First Reaction Method requires N new random numbers, where N is the number of reactions in the reaction network. Regenerating all tentative reaction times might not be necessary, however. If the

execution of reaction μ does not change the propensity function a_j of reaction j through the modification of the populations of any of the involved species, then the waiting time τ_j is still valid and could be used in the determination of the next reaction to occur. Although useful, the computational "trick" just described might not be applicable to complex and strongly coupled reaction networks, in the sense that, for such networks, the execution of any reaction event would affect the majority of the other reactions, and no computational savings would be obtained by the above "trick". For such cases, one would still be forced to generate a significant number of uniformly distributed random deviates, which could lead to computational inefficiencies.

5.2.1.2. Next Reaction Method (NRM)

Identifying the drawbacks of the First Reaction Method as using many random numbers per KMC step and having a computational cost that scales linearly with respect to the number of reactions, Gibson and Bruck developed the Next Reaction Method where they introduced a number of new features [45]. First, the method operates with absolute times, denoted by t_i, instead of relative waiting times, τ_i. Second, the algorithm uses exactly one random number per reaction execution, excluding the initialisation step during which the random numbers used are as many as the reactions of the system modelled. Third, on the implementation level, the method makes use of a dependency graph and indexed priority queues to speed up the reaction scheduling operations.

The NRM builds upon the ideas of the FRM with some modifications. At the beginning, all reactions are assigned an occurrence time t_i using the expression (5.1) above. The minimum among all the occurrence times, t_μ, is obtained and the KMC clock is set to t_μ, $t_{KMC} = t_\mu$. Reaction μ is executed, the molecular populations are updated and the propensities α_i of the affected reactions are updated as well. Then, for the reaction just occurred ($i = \mu$), a new random number, r, is generated and the new occurrence time is obtained as:

$$t_i^{new} = t_{KMC} - \frac{ln(r)}{a_i^{new}} \tag{5.3}$$

For all other affected reactions ($i \neq \mu$), the occurrence times are *adjusted* without the generation of new random numbers according to the equation:

$$t_i^{new} = t_{KMC} + \frac{a_i^{old}}{a_i^{new}}\left(t_i^{old} - t_{KMC}\right) \tag{5.4}$$

where a_i^{old} and a_i^{new} are the propensity functions of reaction *i* before and after the execution of reaction μ respectively, and t_i^{old} is the occurrence time of reaction *i* before the execution of reaction μ. In practice, the occurrence time is decreased or increased, with respect to the current KMC time, t_{KMC}, so that it reflects the increase or decrease of the propensity function. No new random numbers are used at this stage.

During a KMC run, it may happen that a propensity function a_i becomes zero. For as long as the propensity is zero, the corresponding occurrence time, t_i, should be set to ∞, i.e. the reaction will not occur at any finite time. In such cases, the equation (5.4) presented above requires modification [45]. The correct time adjustment transformation when a propensity goes to zero and then ceases to be zero is:

$$t_i^{new} = t_2 + \frac{a_i^{old}}{a_i^{new}}\left(t_i^{old} - t_1\right) \tag{5.5}$$

where t_1 is the KMC time when a_i first became zero, t_2 is the KMC time when a_i ceased to be zero, a_i^{old} is the last pre-zero propensity and a_i^{new} is the first post-zero propensity.

Especially in biological systems, where species participate in low populations, propensities might become zero and then reach non-zero levels again quite frequently. Implementing the NRM in a way to handle zero propensities would require significant bookkeeping of pre- and post-zero quantities thereby complicating the implementation significantly.

5.2.1.3. Modified Next Reaction Method (Mod-NRM)

By changing the representation of reaction times and viewing them as firing times of independent, unit rate Poisson processes, Anderson [46] developed a modified version of the Next Reaction Method of Gibson and Bruck [45]. The advantages of the new method that are relevant to our work are (**a**) the

randomness in the model is uncoupled from the state of the system and (**b**) the approach is general and inherently able to handle zero propensities without any change to the core algorithm. The latter naturally leads to a simpler implementation even for complex systems with multiple species and reactions. For these reasons, we used the Modified Next Reaction Method in our work and we outline its algorithm below. For mathematical derivations and physical meanings of the quantities defined, the reader is referred to the original work [46] and more specifically to Section III thereof.

At the beginning of the KMC run, set the KMC time, t_{KMC}, to zero. For every reaction i, calculate the propensity functions a_i, initialize the "internal times" T_i to zero and set

$$P_i = -ln(r_i) \tag{5.6}$$

where r_i is a uniformly distributed random number in the range (0,1). Then, calculate the firing times of every reaction i as:

$$\tau_i = \frac{P_i - T_i}{a_i} \tag{5.7}$$

Find the minimum among all τ_i, $\tau_\mu = min(\tau_i)$, advance the KMC time, t_{KMC}, by τ_μ and update the molecular populations according to reaction μ. For every reaction i, update the "internal times" T_i as:

$$T_i \rightarrow T_i + a_i \, \tau_\mu \tag{5.8}$$

For the reaction that just occurred ($i = \mu$), generate a new random number, r, and update P_μ as follows:

$$P_\mu \rightarrow P_\mu - ln(r) \tag{5.9}$$

Lastly, recalculate all the propensities a_i, or just the affected ones for increased efficiency. New waiting times are generated using equation (5.7) and the steps following equation (5.7) are repeated until a step-based or time-based termination criterion is met.

We note again that the Modified Next Reaction Method consumes exactly one random number per reaction execution via equation (5.9), excluding the initialisation step. As compared to the NRM, the Mod-NRM operates with time advances, τ_μ, instead of absolute times and, more importantly, zero propensities do not require a different transformation. Elaborating on the latter point, we note that when a propensity a_i becomes zero, the corresponding inter-arrival firing time, as per equation (5.7), becomes infinity. However, the internal time, T_i,

remains unaffected since it is “updated” to the same value as per (5.8). P_i is also unaffected since the reaction i will never occur because its propensity is zero. When a_i becomes non-zero, a finite value is generated for τ_i via (5.7), using values for P_i and T_i that are not infinity.

5.2.1.4. Random Time Change (RTC) method

Working towards efficient methods for computing parameter sensitivities in biochemical networks and changing the representation of reaction times, Rathinam and co-workers developed the Random Time Change (RTC) method [120]. Their method is mostly similar to the Modified Next Reaction Method of Anderson [46] apart from a conceptually significant difference. In the Mod-NRM [46] and, in fact, in any other KMC method described so far [43, 45], all the random numbers consumed during the initialisation step, via the relations (5.1) or (5.6), and during the simulation via (5.1), (5.3) or (5.9), are drawn from a single random number stream. In the RTC method, N independent, parallel random number streams are used, where N is the number of reactions in the system studied, corresponding to one random number stream per reaction channel. Therefore, when using equations (5.6) and (5.9), a different random number stream is used for each value of the index i, i.e. for each different reaction channel.

5.2.1.5. Common Random Number (CRN) method

In the context of KMC simulations, one wishes to collect uncorrelated samples of the state of the simulated process or system, so a high-quality random number generator is used. Not unexpectedly, reusing the same random numbers has generally been avoided in production runs. In sensitivity analysis studies, however, using on purpose the same random numbers across different runs, hence the term “common”, has been an easy and well-known method to reduce the variance by introducing dependence, viz. non-zero covariance, between the results being compared [121]. By using the same random numbers among different runs, one is able to rule out stochasticity-induced changes to the

system's trajectories. By combining the latter approach with appropriate mathematical methods such as finite differences, one may quantify how the system's trajectory changes if one parameter is perturbed from c_0 to c_0+h. In other words, the CRN method makes it possible to obtain quantitative metrics of the sensitivity of a stochastically simulated system on certain parameters by comparing appropriately the reference and the perturbed results.

For sensitivity analysis purposes, when combined with any KMC method, the CRN method is used as follows [120]: a random seed, *rs*, is used to initialize the random number generator and for a given parameter c_0, a solution trajectory of the system studied is obtained. Next, a second trajectory is generated by performing another KMC run in which (a) the random number generator is initialized using the same random seed, *rs*, that was used for the first run, and (b) the rate parameter is perturbed by *h* and its value is changed from c_0 to c_0+h. Since the same random numbers are used in the two runs, thereby eliminating the stochastic component that would differentiate the results, any differences between the two trajectories corresponding to the parameters c_0 and c_0+h respectively, are attributed exclusively to the perturbation *h*, added to the original rate parameter c_0.

Even though the idea of the CRNs is not a novel concept, the combination of a KMC method, and specifically the RTC, along with the CRN enabled the development of yet another method described below.

5.2.1.6. Common Reaction Path (CRP) method

The Random Time Change (RTC) method described in section 5.2.1.4 differs from all other KMC methods because it uses as many random number streams as the reactions in the systems for the generation of tentative firing times. This unique feature of the RTC algorithm in conjunction with the CRN method has been instrumental in the development of the Common Reaction Path (CRP) method by the same authors [120] as an efficient approach for computing parameter sensitivities. Without getting into the details about its performance as compared to other methods, we outline the idea of the CRP method and make the connection with our work later on.

We have already mentioned that the RTC uses one random number stream per reaction channel and that the CRN method reuses the same sequence of random numbers for multiple runs by initialising the random number generator with the same seed. Combining these two features results in the CRP, an algorithm in which the *internal* jump times of each reaction channel remain unaffected among the perturbed and unperturbed simulations. Note that this invariance of the internal jump times requires both that each channel has its own random number stream and that the same random numbers are used. On the contrary, if using the CRN method along with any KMC method other than the RTC, such invariance would not be achieved, i.e., the internal jump times of the perturbed simulation would be different from those of the unperturbed one. This happens because, even though the random numbers are drawn from a single stream, there is no guarantee that they are used in the same order in the perturbed versus in the unperturbed run [120].

5.2.2. On-the-fly rate scaling algorithm

In this section, we describe our algorithm in connection with the methods introduced above. At first, we provide a conceptual description of our algorithm focusing on its features without any technical details. Next, we proceed by presenting a more detailed outline of the algorithm where we introduce its main components. Then, we discuss in detail all parts of our algorithm and last, we list explicitly the user-tunable parameters and provide further details on their importance on obtaining an accelerated simulation with minimal error.

5.2.2.1. Conceptual description

The purpose of our algorithm is to tackle the timescale disparity in well-mixed systems by reducing, on the fly, namely along the course of the KMC simulation, the execution frequency of the very fast (extremely frequent) reactions so that the KMC simulation reaches far greater timescales. We combine the Modified Next Reaction Method [46], multiple random number streams and the ideas of the Common Random Number method as used in sensitivity analysis

studies [120, 121] and develop a methodology that is capable of providing metrics on the error introduced in the system because of the reduction of the reaction rate constants.

The main idea of our algorithm is to use the Mod-NRM, CRNs and distinct random number streams, one for each slow reaction and another one for all the fast reactions of the mechanism (reaction network), to generate and compare over a short and representative KMC time interval two trajectory chunks obtained by using the original and reduced rate constants respectively. When an appropriate comparison, based on all the inter-arrival times of the slow reactions, identifies no differences between the two trajectories, it is implied that the reduction in the rate constants did not incur a perceivable effect on the dynamics of the system. Therefore, it is valid to continue propagating the system in time using the reduced rate constants instead of the original ones. Expanding the idea further, multiple trajectory chunks are generated over a specific KMC time interval, with each one of them corresponding to rates having been downscaled, i.e. reduced, by a progressively larger downscale factor. Among the multiple downscale factors used, the best one is chosen by taking into account the gain in computational efficiency versus the reduction in accuracy, as described below.

It is easy to understand that the more the rate constants are reduced or equivalently, downscaled, the faster the KMC simulation advances in time, but the more inaccurate it becomes due to the introduction of approximation error. In our algorithm, we quantify both computational cost and error, combine them in an objective function and solve an optimization problem for identifying the optimum downscale factor that maximizes the speed gain and minimizes the error.

5.2.2.2. Outline

Continuing from the brief conceptual description, we now proceed by providing a detailed outline of the procedures involved during a KMC simulation that incorporates our on-the-fly rate constant scaling algorithm. More technical and implementation details are provided in subsequent sections.

Once the simulation is up and running with the original reaction rate constants, the algorithm determines a KMC time interval, (t_s, t_f), $t_f > t_s$, over which

it is invoked, in order to perform and evaluate the feasibility and correctness of a downscaling (the latter refers to a reduction in the rate constants of the fast reactions). Over that KMC time interval, (t_s, t_f), and while using the CRN method and multiple random number streams, the algorithm generates multiple trajectories that correspond to rate constants reduced by an exponentially increasing downscaling factor, such as 5, 25, 125, 625, etc. Since the downscaling factor is applied only to the rate constants of the very frequent reactions that consume most of the computational time, the execution frequency thereof decreases proportionally to the downscale factor. As a result, the generation of each addition trajectory chunk is progressively less costly. From a quantitative perspective, we estimate the computational cost of each individual trajectory chunk using the number of KMC steps executed over the aforementioned time interval (t_s, t_f). In fact, the very definition of the computational cost is often the number of steps executed or operations performed.

It is intuitive to realise that the higher the reduction in the rate constants of the fast reactions, the faster the simulation progresses and the computational cost decreases. However, the more aggressive the downscaling is, the more inaccurate the simulation becomes. By downscaling, one gains in terms of speed and loses in terms of accuracy. Quantifying accuracy is non-trivial due to the inherently stochastic nature of KMC simulations. Thus, we need a reliable procedure to discern changes in the system's trajectory arising from (a) the reduction of the rate constants of the fast reactions versus (b) the inherent stochasticity of KMC. Aiming to eliminate stochasticity and focus on changes incurred solely because of the modification of the rate constants, we use the standard technique of the sensitivity analysis studies: Common Random Numbers. In addition, we make use of multiple random number streams, along the principles of the RTC method presented above. More specifically, we generate the firing inter-arrival times for all the fast reaction channels by drawing random numbers from a single random number stream, whereas we use separate random number streams, one per reaction channel, for the slow reactions. Finally, we quantify the accuracy of the downscaled trajectories as compared to the reference one, in terms of the inter-arrival time differences of the slow reactions in a way that is described in detail further below.

By collecting information on the scaling of the computational cost, *C*, and error, *E*, with respect to the downscale factor, *df*, we compose the following objective function, *F(df)*:

$$F(df) = a \times C(df) + b \times E(df) \tag{5.10}$$

with *a, b*, being the weights on the computational cost, *C,* and error, *E*, respectively. Given the objective function, the goal is to identify the optimal downscale factor, df_{opt}, which maximises the speed-up gain with the least error. Once that factor has been determined, the rate constants are modified accordingly, and the simulation continues propagating the system in time with the reduced rate constants. The above procedure is invoked regularly at specific KMC time intervals throughout the KMC run and aims at identifying whether further downscaling would be possible without increasing the error to unacceptable levels.

Our method, as outlined above, applies state-of-the-art concepts from sensitivity analysis [120, 121] to make long KMC simulations tractable. The ideas we use have not appeared before in the context of accelerating KMC simulations of spatially homogeneous chemical reaction systems. Our algorithm reduces on the fly the rate constants of fast reactions in such a way that the introduced error is imperceptible and indistinguishable from the KMC error itself. Most importantly, the rate reduction procedure of our algorithm is data-driven and optimises for maximum speed-up and minimum error by solving an appropriate optimisation problem.

5.2.2.3. Partitioning the reaction network into fast and slow reactions

A key component of our methodology is the use of a different random number stream for each one of the slow reaction channels in the system simulated. Therefore, the correct identification of the reactions that are expected to be slow is a crucial first step towards setting up our algorithm, so as to initialise the correct number of independent random number streams.

To classify the reaction channels into fast and slow, the user needs to first provide information on the ordering of the execution frequency of reactions, obtained from a short trial run. In practice, the user provides a ranking of the

reaction channels from the fastest to the slowest and pinpoints those reactions whose rates are allowed to undergo downscaling. The “downscalable” reactions are restricted to forward-reverse pairs only in order to maintain detailed balance and quasi-equilibrium and before invoking a downscaling we check whether those reactions are indeed quasi-equilibrated (QE) using the criterion:

$$\frac{|N_f - N_r|}{N_f + N_r} \leq \delta \tag{5.11}$$

where N_f and N_r is the number of executions of the forward and reverse reaction respectively and δ is a threshold value chosen as 0.05, namely the forward and reverse reaction executions should differ by a maximum of 5% to be considered quasi-equilibrated.

The above criterion along with the user provided information on the expected ordering of the reaction frequencies are taken into account while attempting a downscaling. If the above criterion fails, namely, the value of the ratio in equation (5.11) is greater than 5% or the ordering of execution frequencies changes, then the attempt is terminated and the KMC simulation continues with the most recent rate constants.

The classification of processes into fast and slow can, in principle, be done automatically. In the presence of non-reversible reactions however and especially in oscillatory systems [100, 122], the latter procedure is not trivial and requires deeper analysis of the reaction network. Since our focus is to develop an on-the-fly rate scaling algorithm that optimises cost and error, we opted for requesting user input on the reaction network and postponed the automation of the partitioning procedure for a future improved version of our algorithm.

5.2.2.4. Generating multiplicates with CRNs and different rate constants

Our methodology is based on the idea of generating multiple trajectories using progressively reduced rate constants and comparing all those against a reference trajectory to evaluate the error introduced and the gain in computational cost. To meaningfully compare the various trajectories, we use Common Random Numbers in a way described below.

Following the initialisation step, the KMC simulation starts as usual and after the execution of N KMC steps, at KMC time t_s, our algorithm is invoked to check whether a reduction on the rate constants of the fast processes is feasible. At time t_s, a snapshot is saved that contains information on the current state of the system such as the species populations, the rate constants of all reactions and the states of the random number generators that correspond to the slow reactions.

Following the snapshot saving, the algorithm propagates the system until KMC time t_f with the same rate constants used up until t_s, that is, the rate constants are not modified. The time t_f is chosen such that (a) propagating the system further for $w = t_f - t_s$ KMC units is still tractable in a reasonable amount of time and (b) all reactions, especially the slow ones, are sufficiently sampled, although the latter is not strictly necessary for reasons that will become clear further below. In addition to the usual KMC procedures performed, such as population sampling over time and incrementing of reaction counters, over the interval (t_s, t_f) we record the occurrence times $t_{i,j}$ of every single reaction firing i belonging to any slow reaction channel j. As a reminder, we reiterate that a different random number stream is used for each of the different slow reaction channels j. The generation of the trajectory with the unmodified rate constants is very important because it serves as our reference in subsequent comparisons. For this reason, we refer to this trajectory as the “unscaled” or “reference” trajectory.

Once the unscaled trajectory is generated, we perform the checks described previously in subsection 5.2.2.3. At first, we check whether the fast reversible reactions, whose rates we are allowed to reduce, are quasi-equilibrated using the criterion (5.11). Second, we order the reaction frequencies from the fastest to the slowest and we compare their ordering against the user-provided, i.e. the expected ordering. More specifically, we check if the anticipated fast reactions are indeed fast and the anticipated slow reaction are indeed slow in terms of their execution frequencies. The latter is a necessary check because in oscillatory systems the execution frequencies might not have the same ordering for any arbitrarily chosen KMC time interval. Third, we quantify the timescale separation, TSS, namely, the difference on the execution frequencies among the fast and slow reactions, in terms of orders of magnitude. The user

provides TSS_{min}, the minimum timescale separation, which implies that if the fast reactions are not fast enough, no downscaling is attempted. The purpose of the timescale separation criterion is to prevent making the initially fast reaction too slow, or comparably slow to the initially slow ones. If any of the above checks fails, the unscaled trajectory is accepted as the "official" trajectory over the interval (t_s, t_f) and the simulation continues from KMC time t_f. In the case where the above checks yield a positive outcome, namely, the fast processes are quasi-equilibrated in the interval (t_s, t_f), their ordering is as expected, and the fast reactions are fast enough, the algorithm proceeds in generating the downscaled trajectories.

Having generated the unscaled trajectory, the algorithm restores the state of the system back at time t_s using the saved snapshot. The latter action restores the random number generators of the slow reactions to the state they had at t_s. This is done intentionally, so that the same random numbers are used for the generation of the occurrence times for the slow reactions in the interval (t_s, t_f), in line with the CRN methodology [120]. On the other hand, the occurrence times of the fast processes are not of interest and are thus not saved at all, neither is their generator's state saved or restored. Then, the rate constants of the fast processes are reduced by a user-defined "base downscale factor", df_b, a downscaled trajectory is generated over the KMC time interval (t_s, t_f), and the occurrence times $t_{i,j}$ of all slow reactions are recorded. Using the state snapshot at t_s as the starting point, a total of k_{max} downscaled trajectories are generated, each one with an exponentially increasing downscale factor of df_b^{k-1}, where k is the index of the downscaled trajectory being generated. It is important to note here that during the generation of the first downscaled trajectory, corresponding to $k=1$, the rate constants are actually not reduced since the downscale factor is one. Despite the additional computational cost, this trajectory, in combination with the unscaled trajectory, whose generation was described above, are both crucial in determining a baseline for the error calculations as we will see shortly. To conclude with an example, if the user provides $df_b = 5$, then the first downscaled trajectory is generated by dividing the original rate constants by $df = 1$, the second by $df_b^{2-1} = 5^1 = 5$, the third by $df_b^{3-1} = 5^2 = 25$ and so on.

At the end of the above procedure, we have obtained one unscaled and k_{max} downscaled trajectories, over the same KMC time interval (t_s, t_f) while reusing exactly the same random numbers for the occurrence of the slow reactions. In addition, we have obtained the absolute occurrence times of all the slow reaction firings along with the total KMC steps executed for every trajectory generated. The latter quantities are used to quantify the error and computational cost respectively, as we shall see in the next section.

5.2.2.5. Cost and error quantification & optimization

In this section, we describe the way we quantify the computational cost and the error for each slow reaction introduced because of the reduction of the rate constants of the fast processes. To this end, we use the number of KMC steps executed during the generation of the unscaled and downscaled trajectories and the occurrence times of all slow reaction firings that were generated from the procedures described in section 5.2.2.4.

A direct measure of the computational cost for the generation of each one of the alternative trajectories is the number of KMC steps executed. To derive a relative dimensionless cost for each trajectory we divide the total number of KMC steps executed therein by the number of KMC steps of the unscaled trajectory, so that the latter has a dimensionless cost of one. Then, we identify the maximum downscale factor that does not distort the dynamics of the system with the following reasoning and procedure. Since we are downscaling only the fast reactions, we expect the computational cost (equivalently: the number of KMC steps executed) to drop according to an inversely proportional relation with respect the downscale factor: *cost ~ 1/df*. Hence, the normalised cost plotted on log-log axes with respect to the downscale factor would appear as a line with gradient of -1. The relationship just noted should hold true as long as the reactions whose rate constants are reduced are much faster than all other reactions and dominate the computational simulation. Violations of this condition may occur, and potentially indicate that the initially fast reactions are no longer the most time-consuming ones. To detect such violations and identify the downscale factor after which the cost does not decrease as expected, we use

two different methods, thereby ensuring the robustness of the procedure. First, we perform successive linear fittings on the *log(cost)* versus *log(df)* data using a "sliding window" approach with a set of just four data points each time. When the gradient of the linear fit is larger than -0.9, the second to last data point of the set is marked as the maximum valid downscale factor, df_{max_1}. Second, we calculate the second derivative of our data points using central finite differences. For a linear relation the second derivative is zero; thus, when the second derivative exceeds the threshold of 0.05, we have identified the maximum valid downscale factor, df_{max_2}. The minimum among (df_{max_1}, df_{max_2}) is taken as the final df_{max} and we discard the data points for larger downscale factors as invalid. Next, to obtain an analytical expression for the cost, i.e. the first component, *C(df)*, of equation (5.10), we fit a line to the above data points that are deemed valid.

We have so far discussed how we quantify the cost of the simulation, and we now move on to discuss the error. The main motivation in using the Modified Next Reaction Method of Anderson [46] is that it uncouples the randomness in the model from the state of the system. In addition, using different random number streams for each slow reaction channel makes the firing times of each channel independent from the firing times of other channels. Restoring the states of the random number generators when generating a new downscaled trajectory ensures that the internal firing times are preserved across the different runs which is equivalent to keeping the reaction path of the slow processes the same, in line with the CRP method [120].

Thus, to quantify the error incurred by downscaling, we use the occurrence times $t_{i,j}$ to calculate the inter-arrival times $\tau_{i,j}$ for every single reaction firing *i* from the slow reaction channel *j* for the unscaled and all the downscaled trajectories. Then, for each downscaled trajectory *k*, and each slow reaction channel *j*, we calculate the inter-arrival time (IAT) difference vector, $\boldsymbol{d}$, with respect to the unscaled trajectory as:

$$\boldsymbol{d}_j^k = \boldsymbol{\tau}_j^u - \boldsymbol{\tau}_j^k \tag{5.12}$$

where $\boldsymbol{\tau}_j^u$ is the vector holding the inter-arrival times of the unscaled trajectory, hence the superscript *u*. In general, $\boldsymbol{\tau}_j^u$ and $\boldsymbol{\tau}_j^k$ have the same number of elements, i_{max}, which is equal to the number of reactions fired. If it happens that their sizes differ, namely, there was a different number of reactions fired in the

unscaled, *u*, and in the downscaled trajectory with index *k*, the common reaction firings, i.e. the minimum number thereof is used to calculate $\boldsymbol{d}_j^k$. Observing a different number of firings for a slow reaction channel *j* is more likely to occur at large downscale factors. Lastly, for each *j* we define the IAT error norm as the Euclidean norm of the IAT difference vector $\boldsymbol{d}_j^k$:

$$\boldsymbol{e}_j^k = \sqrt{\sum_{i=1}^{i_{max}} \left|\boldsymbol{d}_{i,j}^k\right|^2} \tag{5.13}$$

The above metric quantifies the difference between two trajectories in terms of differences in their inter-arrival times. We calculate the error norm for every downscaled trajectory *k* thus obtaining the error scaling for a particular slow reaction channel *j* up to df_{max}. To obtain an analytical expression for the scaling of the overall error, we fit the scaling equation $y = px^q$ to all error norm data points, with *p* and *q* being parameters determined by the fitting, and *x, y* being the downscale factor and IAT error norm respectively. Fitting the equation $y = px^q$ is equivalent to fitting a first-order polynomial to the logarithms of the error norm and downscale factor, just like in the case of the computational cost. Following the above procedure, we obtain the second component, *E(df)*, of equation (5.10), and we are now ready to form the objective function to optimise.

Using the user-provided weights on cost and error, along with the analytical expressions for the scaling of cost and error, *C(df)* and *E(df)* respectively, fitted to the data obtained from the KMC simulation, we form the objective function as per equation (5.10). Then, we find its minimum, thereby identifying the optimal downscale factor, df_{opt}, to use for the current downscaling attempt. Using df_{opt}, we generate a trajectory using rate constants reduced by df_{opt}, until a KMC time of t_f, and save the trajectory as the “official one”, along with the samples taken before t_s.

At KMC time t_f, the algorithm exits the downscale mode and continues propagating the system with the most recent, downscaled, rate constants. Following the execution of *N* new KMC steps, the algorithm is invoked again to check whether further downscaling is feasible. This is done over a different KMC time interval $(t_s, t_f)_m$, where the subscript *m* counts the number of the downscaling attempts. The previous description corresponds to the first attempt, and the

subscript *m* was omitted for brevity since *m=1*. Finally, the cost and error are calculated in terms of the absolute downscaling factor, that is, from the second downscaling attempt onwards, *m>1*, we take into account any previous successful downscaling attempts. This is to prevent the loss of information regarding the overall speed gain and loss in accuracy.

5.2.2.6. Additional considerations in decision making

Following the formation of the objective function, the identification of the optimum downscale factor is easy. Apart from the weighting factors on cost and error, there are additional criteria put in place while evaluating whether the optimum downscale factor leads to an acceptable increase in the error.

We mentioned above that the k^{th} downscaled trajectory has the kinetic rate constants of the fast reaction reduced by df_b^{k-1}. Practically, the first downscaled trajectory uses the same kinetic rate constants as the unscaled trajectory. In addition, the same random numbers are used for the generation of the slow reaction firings. What is different is that the fast reaction channels do not have their random number generator restored, and thus, the slow reaction firings in the unscaled trajectory do not occur in the same time instances as those of the "downscaled" one with *df = 1*. For reasons explained in detail in the SI, the slow reaction firing times have a distribution, with a certain mean and variance, and the latter depends on the timescale separation, i.e. how much faster are the fast processes with respect to the slow ones. To obtain a reference point for the IAT error norm, the generation of a trajectory with *df = 1* is not only necessary but crucial as well, and its importance outweighs the additional computational expense for its generation.

By taking into account the reference IAT error norm and the IAT error norm of df_{opt}, we impose the following additional criteria to prevent the overall error from increasing excessively:

(a) absolute threshold: the IAT error norm that corresponds to df_{opt} should not exceed e_{max}, taken as 0.05 in our benchmarks, and

(b) relative threshold: the IAT error norm corresponds to df_{opt} should not increase by more than two orders of magnitude with respect to the IAT error norm of the downscaled trajectory generated using *df=1*.

A last criterion taken into account considers cases where the IAT error norm appears saturated, i.e. it does not increase further with increasing downscale factor over the interval up to df_{max}, as illustrated in Figure 21, curve (c). This phenomenon is an indication that the KMC time interval over which we attempt to do a downscaling is inappropriate for downscaling. To detect such cases, we impose a lower bound of 0.2 to the coefficient *q* of the equation $y = px^q$ that is fitted to the IAT error norm data points for each slow reaction channel. We also note that we take into account data points up to df_{max}, the vertical dashed line of Figure 21. As a result, the error curve of reaction channel (a) on Figure 21 does not suffer from error saturation, it is the scaling of reaction channel (c) that will cause the downscaling attempt to be aborted.

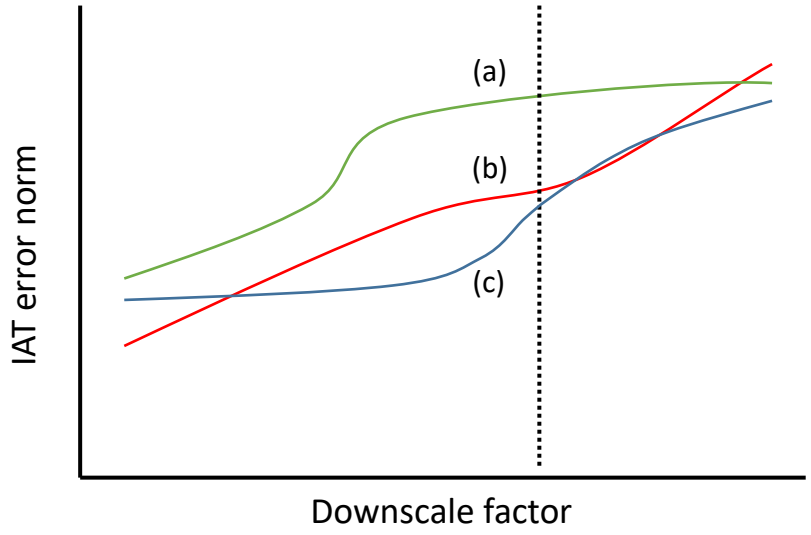


Figure 21: Qualitative error scaling with respect to the downscale factor for three fictitious slow reaction channels (a)-(c). Both axes are on logarithmic scale. The IAT error norm of reaction channel (c) appears saturated over the small downscale factors, whereas for reaction channel (a), the error norm is saturated over the large downscale factors, which are discarded anyway. Reaction channel (b) exhibits a more conventional scaling, for both small and large downscale factors. The vertical dashed line represents the maximum allowed downscale factor and data-points to its right are discarded.

5.2.2.7. User-tunable parameters

In this subsection, we summarize all the user tunable parameters involved in our algorithm and discuss their effect on the performance thereof.

- N: the number of KMC steps after which our downscaling algorithm is invoked. The larger N is, the less often our algorithm is called.
- $w = t_f - t_s$: the width, in KMC time units, of the window over which the algorithm performs the downscaling evaluations. The larger w is, the more confidence we have on a downscaling if accepted, but at an increased computational cost. A good value for w is, generally, system dependent.
- df_b: the base downscaling factor used in the generation of multiple downscaled trajectories. Small values such as $df_b = 2$ would increase the total cost the downscaling attempts, whereas large values such as $df_b = 20$ could result in sub-optimal choices for df_{opt} during the optimisation procedure.
- n: the total number of downscaled trajectories generated inside the downscaling window including the one with $df = 1$. The generation of each subsequent trajectory is becoming cheaper, so large values on n do not affect performance. At the same time, generating too many downscaled trajectories may be redundant, because the trajectories in which the system is distorted are discarded.
- TSS_{min}: the minimum timescale separation, i.e. the difference of the execution frequencies among the fast and slow reaction channels, in terms of orders of magnitude, that is required in order to proceed with the generation of the multiple downscaled trajectories. Larger values favour accuracy whereas smaller values allow for more downscaling and greater speedup.
- a, b: weight coefficients for the computational cost and error respectively. Higher values of b favour the accuracy and therefore the df_{opt} is shifting to smaller values.

- e_{max}: maximum absolute error threshold for any chosen downscale factor df_{opt}. Smaller values may cause attempts to be rejected due to stricter accuracy requirements.
- e_{inc}: maximum allowed absolute increase in orders of magnitude of the overall error norm. Smaller values limit the number of successful attempts as well as the overall reduction in the rate constants for the fast processes.

5.2.2.8. Summary of algorithm and flowchart

Summarising all the steps described in detail in the previous subsections, we present the flowchart of our algorithm (Figure 22) and highlight its important components. The main KMC loop that propagates the system forward in time is the same as in "traditional" KMC algorithms. The counter i_{DS} keeps track of the KMC steps executed for the purposes of invoking our algorithm when $i_{DS} = N$, at KMC time t_s. The simulation enters the downscale mode, saves the state of the system, and generates the unscaled trajectory (subsection 5.2.2.4). If there is sufficient separation between the fast and slow reaction in terms of execution frequency, then the downscaled trajectories are generated. Using the KMC steps executed and the firing times of all slow reaction events, we calculate the cost and IAT error and fit the appropriate equations to both. If the error checks on the scaling of the IAT error norm pass successfully, we form the objective function and find the optimum downscale factor. If its error is below the threshold, the corresponding trajectory is generated and officially accepted/registered, the rate constants are reduced, and the algorithm exits the downscale mode. Then the system is propagated in time using the reduced rate constants. In the case where a check fails, the downscaling attempt is aborted, the unscaled trajectory is registered as the official trajectory over the KMC time interval (t_s, t_f) and the KMC simulation continues as usual. The counter i_{DS} is reset and keeps track of the KMC steps to invoke the algorithm again when $i_{DS} = N$.

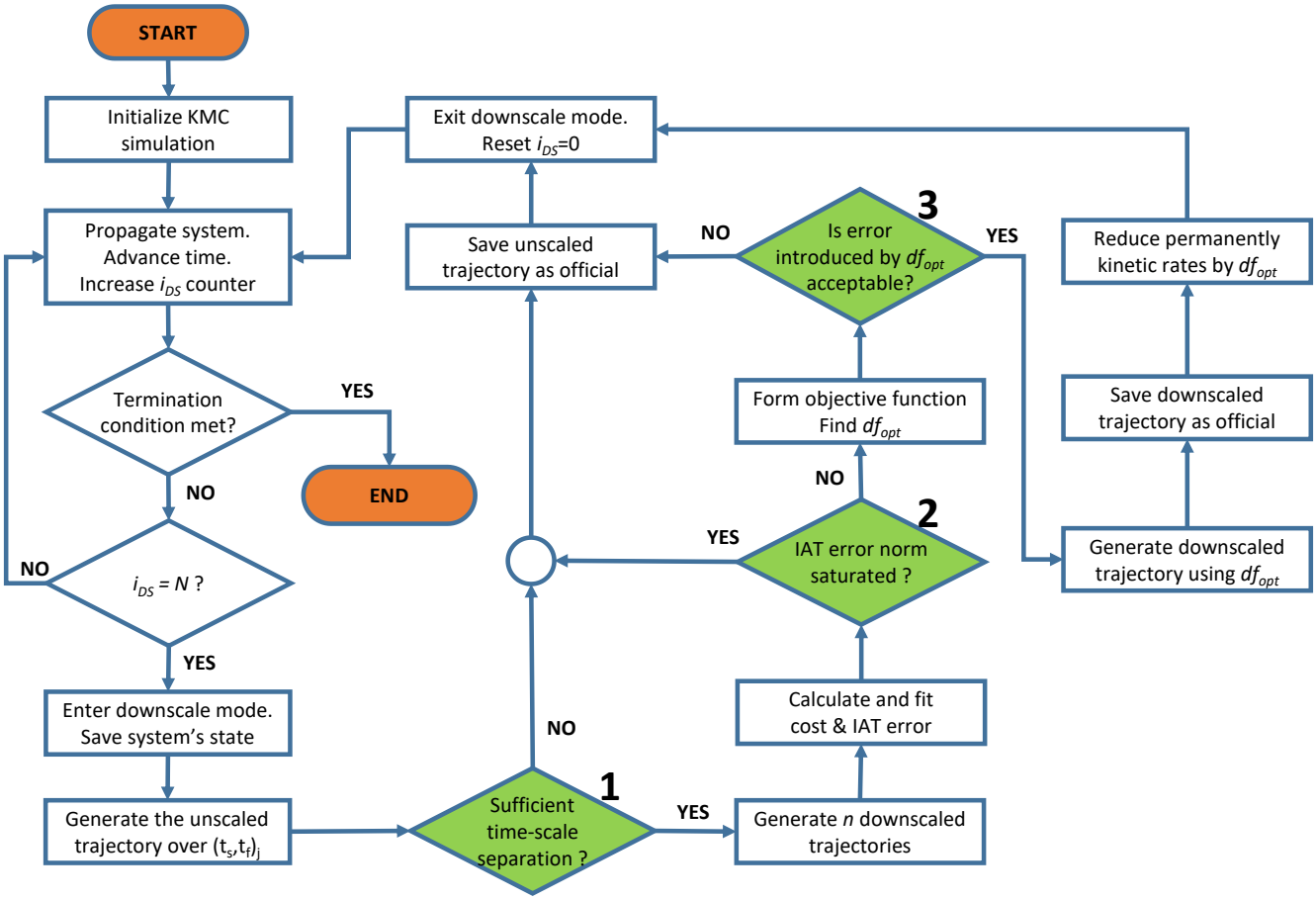


Figure 22: Flowchart of the proposed algorithm. The green shaded and numbered shapes represent the decision steps of the algorithm.

5.3. Computational Model, Results and Discussion

To validate our methodology and study its performance and efficiency, we apply the algorithm described above to an oscillatory reactive system inspired from biology. In the following, we first present our benchmark system along with its main features and identify the source of its temporal stiffness. We then present in detail the application of the downscaling algorithm on this system and show in practical terms the various algorithmic steps and decision criteria. In addition, we present and discuss cases where the downscaling is rejected, such as when the error is saturated. In the last subsection, we compare the results of the original, unscaled simulation with the downscaled one and discuss the validity of the methodology by presenting evidence that our algorithm did not distort the key features of the dynamics of the system.

5.3.1. Reaction model

Our motivation and aim was to develop a generic on-the-fly rate scaling methodology rather than a system-specific one. For this reason, the benchmark system was chosen carefully in order to try to push the limits of the methodology developed, towards ensuring the robust operation and effectiveness of the latter. Thus, we would like a system (a) with a good level of complexity, (b) with rich dynamic features such as oscillations, which must not be disturbed by the downscaling procedure, (c) with stiffness that arises from pairs of reversible reactions, and (d) that is of practical interest to multidisciplinary fields. Regarding point (d), we also note that in biological systems, where the number of interacting molecules is typically low, stochastic effects may be significant for the evolution of the system [123, 124]. Taking into consideration the above reasons, we have chosen, as out benchmark system, a biological oscillator that mimics the cell cycle. Our benchmark system is a slightly modified version of the model introduced by Stamatakis and Mantzaris, as summarised in Table III in [122] and includes the following nine reactions:

1	$O \xrightarrow{k_0} O + X$
2	$OX_2 \xrightarrow{k_1} OX_2 + X$
3	$2X \xrightarrow{\chi} X_2$
4	$X_2 \xrightarrow{\beta\chi} 2X$
5	$O + X_2 \xrightarrow{\varphi} OX_2$
6	$OX_2 \xrightarrow{a\varphi} O + X_2$
7	$Y_i + X_2 \xrightarrow{k_2} Y + X_2$
8	$X + Y \xrightarrow{\lambda_1} Y$
9	$Y \xrightarrow{\lambda_2} Y_i$

Table 5: Reactions included in our benchmark model.

The reaction network of Table 5 involves the following six species: O, X, X_2, OX_2, Y_i and Y. Due to the system's conservation laws O + OX_2 = O_{total} and Y + Y_i = Y_{total}, stemming from reactions 5,6 and 7,9 respectively, only four of the

participating species are independent, viz. O, X, X_2 and Y. The populations of OX_2 and Y_i are calculated using the conservation laws just presented. For detailed information on the physics of our benchmark model and the correspondence of species presented to actual biomolecules, we refer the interested readers to the original work [122]. Very briefly, we note that the oscillations on this system are caused by the autocatalytic action of species X, whose dimer, X_2, exerts positive feedback, i.e. further enhances its own production, while Y, an active state of Y_i, exerts negative feedback by degrading X via reaction 8 of Table 5.

To provide a quick overview of the behaviour of our benchmark system along with its main features, we use the parameters listed in Table 6, with ζ=10, and propagate the system up to a final time of 500 minutes (KMC time units). The obtained trajectory illustrating how the concentrations of the various species evolve in time is presented in Figure 23(a). Evidently, the main feature of the system is its oscillatory behaviour with a period of around T=200 minutes. For most of the time, the concentrations of species X and X_2 are very close to or exactly zero. As a result, the propensities of reactions 3-6 are very small or zero. The KMC time advancement with respect to KMC time (Figure 23(b)) reveals that

Parameter	Value	Unit
O_{total}	10	copy number
Y_{total}	1040	copy number
k_0	0.15	min^{-1}
k_1	50	min^{-1}
k_2	1.88×10^{-3}	$nM^{-1} \cdot min^{-1}$
φ	$9.77 \cdot \zeta$	$nM^{-1} \cdot min^{-1}$
χ	$3.91 \cdot \zeta$	$nM^{-1} \cdot min^{-1}$
α	159.37	nM
β	5.31	nM
λ_1	9.38×10^{-3}	$nM^{-1} \cdot min^{-1}$
λ_2	0.01	min^{-1}

Table 6: Parameter values for the benchmark model. The parameter ζ controls the timescale separation or the stiffness of the system.

the system has fast- and slow-propagating regions, whose time advancements exhibit periodicity as well. Figure 23(c) also shows that the computational cost, quantified by the number of KMC steps executed, is different during the course of the KMC simulation. More specifically, as illustrated in Figure 23(c), 8×10^6 KMC steps are executed to propagate the system until KMC time 40, whereas only a tiny fraction thereof (13×10^3) are needed to further propagate the system until KMC time 200. Combining the visual information of the panels (a), (b) and (c) of Figure 23, we conclude that the slow-propagating regions occur when the concentrations of species X and X_2 become non-zero and the concentration of Y increases rapidly. Lastly, the overall execution frequency of all reactions in our network is illustrated in Figure 23(d). It is obvious that reactions 3-4, and 5-6 are quasi-equilibrated as the forward and reverse steps of reversible reaction events. It is also clear that reactions 3-6 dominate the computational simulation, since they are executed at least two orders of magnitude more frequently than the next most frequently executed reactions, i.e. reactions 2 and 8.

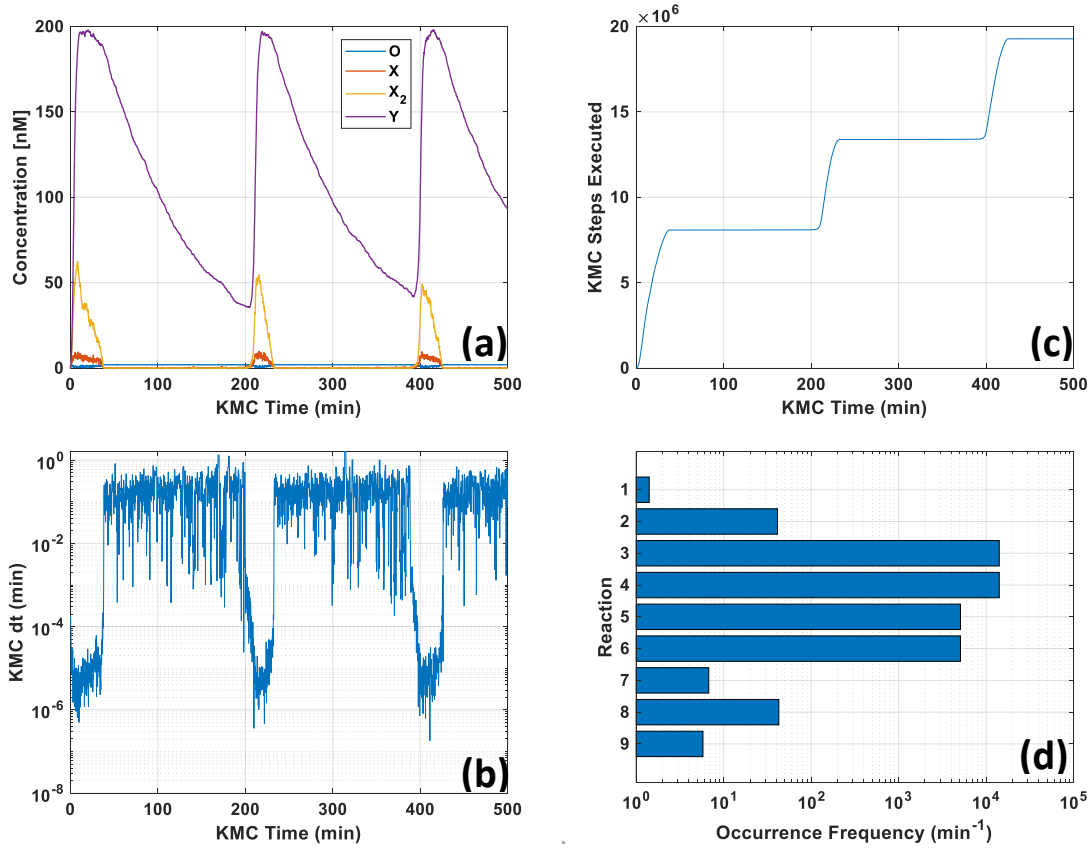


Figure 23: (a) Time evolution of the benchmark system. (b) KMC time advancement (dt) with respect to KMC time. (c) KMC steps executed with respect to KMC time. (d) Overall execution frequencies of all the reactions on the network.

With a closer look at the reaction network and Figure 23, we may conclude that once reactions 3-6 are quasi-equilibrated, they no longer contribute any transient features to the dynamics of the system. Yet, reactions 3-6 consume most of the computational time, more specifically 99.5% of the total executed KMC steps, thereby making the system “stiff”. Based on the above, along with the criteria introduced in the beginning of this section, the chosen benchmark system is ideal for our downscaling methodology and could help us draw conclusions on the generality of the latter. The chosen benchmark system is relevant to the mathematical biology field. Nevertheless, any field studying interacting species and the evolution of reaction networks in time could benefit from the proposed methodology.

5.3.2. Application of the downscaling algorithm

In this section, we present the application of the developed methodology to the conceptual “toy-model” of the cell cycle presented above. We present all the steps of the algorithm along with our results. We also discuss and present results on cases where the downscaling attempt is aborted because an error-related check fails, as per the numbered decision-blocks of the flowchart in Figure 22.

As already discussed in section 5.2.2.3, for the algorithm to scale down the rate constants of the fast reactions, we provide information about the expected execution frequency of the reactions in our network and indicate the reactions whose rates may undergo reduction. For the cell cycle model, the fast reactions are 3-6, the remaining reactions are expected to be (much) slower, and the parameters allowed to be downscaled are χ and φ only. A total number of $N = 10^5$ KMC steps are executed before our algorithm is invoked. We have chosen our base downscale factor as $df_b = 5$, hence the rate parameters χ and φ are reduced in powers of df_b such as 5, 25, 125, etc. A total of $n = 9$ downscaled trajectories are generated in each attempt. As a reminder, we note that the first trajectory has $df = df_b^0 = 1$, therefore the maximum reduction of the rate constants is done by $df = df_b^8 = 390625$. The minimum required Time-Scale-Separation, TSS_{min}, in the execution frequencies between the fast and the slow reactions is

set to two orders of magnitude. Thus, the algorithm will only attempt a downscaling if the fast reactions, especially 5-6, are at least $10^2 = 100$ times faster than the fastest slow reactions, reactions 2 and 8 as shown in Figure 23(d). The weights on the cost and error are set as $\alpha = 1$ and $\beta = 2$ respectively since we aim for better accuracy. Finally, we allow a maximum IAT error of 0.05 and a maximum increase in the error by two orders of magnitude. The latter is feasible since we obtain a reference error point by generating the trajectory with $df = 1$. The values of all user-defined parameters used in our simulations are summarised in Table 7. Lastly, for the parameter ζ that appears in the rate parameters in Table 6 we have chosen the value of ζ=100 unless stated otherwise.

Parameter	Value	Units
N	10^5	KMC steps
$w_{min} = t_f - t_s$	10	KMC time units
$w_{max} = t_f - t_s$	15	KMC time units
df_b	5	-
n	9	-
TSS_{min}	2	Orders of magnitude
α	1	-
β	2	-
e_{max}	0.05	-
e_{incr}	2	Orders of magnitude

Table 7: User-defined parameters of the developed algorithm as discussed in section 5.2.2 and summarised in 5.2.2.8.

In the beginning, the KMC simulation is initialised, the time is set to zero and the species populations are set to zero apart from OX_2 and Y_i that are initialised using the first two values of Table 6 respectively. The KMC simulation runs as usual for $N = 10^5$ and reaches a KMC time of $t_{KMC} = 1.503$ minutes. Then the downscaling algorithm is invoked. The state of the system is saved and the width of the current downscaling window is chosen as w = 10 minutes, which is the minimum width imposed by us. Over the next KMC time the interval (1.503,

10.503) the algorithm generates the unscaled trajectory of the system and calculates the reaction execution frequencies (as illustrated in Figure 24). The Time-Scale-Separation, calculated as the logarithm of the difference of the execution frequencies of reactions 5 and 2, is found to be 3.093, which is sufficient based on the user-defined TSS_{min}, therefore, the algorithm proceeds in restoring the state of the system back to t=1.503 minutes and generating the n = 9 downscaled trajectories. Once the downscaled trajectories are obtained, the algorithm moves on to the evaluation and decision-making stage.

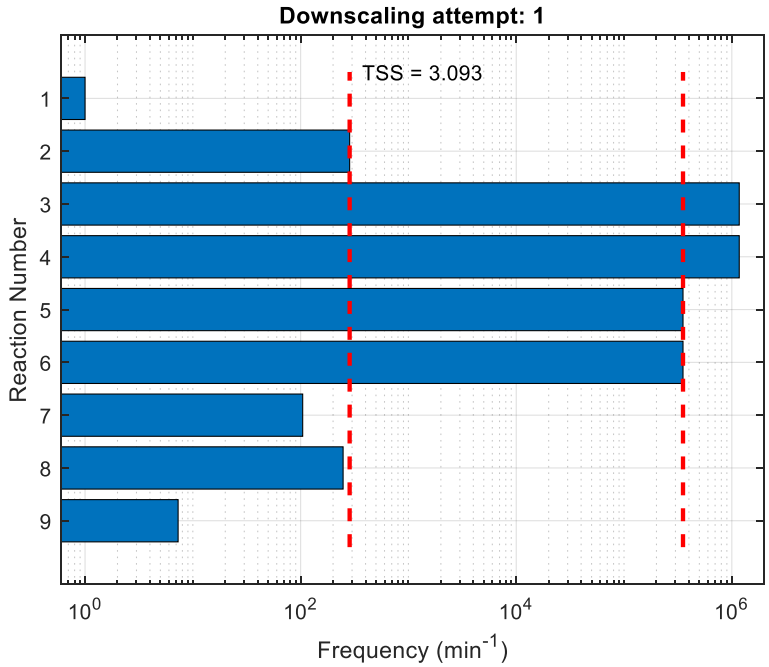


Figure 24: Execution frequencies of all reaction channels of the unscaled trajectory over the first downscaling interval (1.503, 10.503)

Using the collected data, such as KMC steps executed and the occurrence times of all slow reaction firings, the cost and IAT error norm are calculated as discussed in section 5.2.2.5. Both quantities are plotted against the downscale factor in Figure 25. The additional error check concludes that the IAT error norm for each of the slow reaction channels scales as expected, i.e. none of those error norms is saturated. Before forming the objective function, we fit the cost and IAT error data points to obtain the analytical expressions for the cost, *C(df)*, and error, *E(df)*, respectively, as shown in Figure 25 by the red and blue bold dashed lines. More specifically, for the fit on the IAT error, we use all the error data-points from all slow reactions and perform a single fitting to obtain *E(df)*. Then, the objective

function is formed using equation (10) and the weights as reported in Table 7. The minimum of the objective function (bold dark orange curve in Figure 25) suggests that the downscale factor $df = 81.3$ strikes the ideal balance between cost and accuracy. The latter downscale factor does not violate any of the additional error checks, namely, the absolute error incurred is 0.013, below the $e_{max} = 0.05$, and the overall error increase remains below the two orders of magnitude since the average error increases by a factor of 10. For comparison purposes, we use additional pairs of weight factors for the cost and error, to illustrate how the weights affect the objective function and thus the optimum downscale factor chosen. The additional objective function curves are illustrated with bold continuous curves in Figure 25 and the corresponding weights are as listed in the legend.

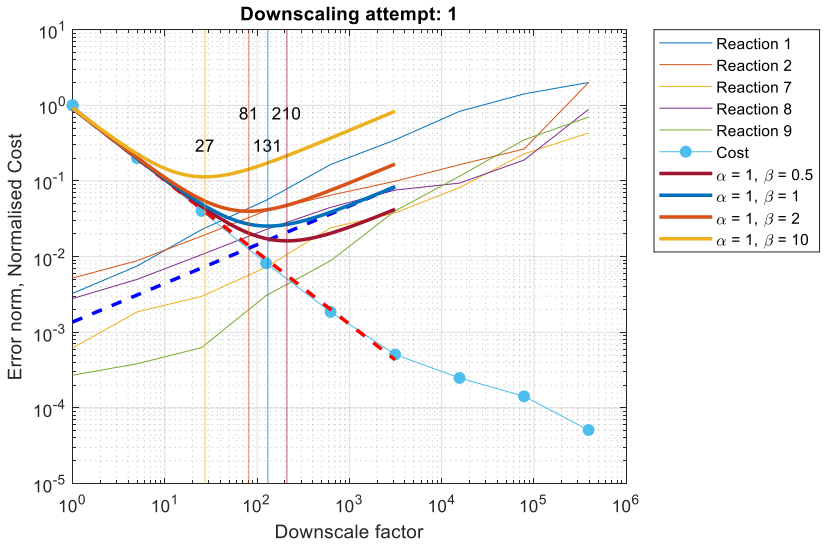


Figure 25: Cost and IAT error norm with respect to the downscale factor. The red dashed line is the fit of the cost. The blue dashed line is the aggregated fit on the IAT error. The coloured bold lines are the objective functions for different weights on cost and error as listed in the legend. The coloured vertical lines and the overlaid numbers correspond to the minimum of the different objective functions, matched by color.

The decision-making procedure identifies the optimum downscale factor as $df_{opt} = 81.3$. Then, for the final time, the state of the system is restored back to $t_{KMC} = 1.503$ minutes, to generate the final downscaled trajectory of the system using df_{opt}. The latter trajectory is registered as “official”, i.e. the species concentration samples are saved along the samples collected outside any

downscaling attempt, the rates χ and φ are reduced permanently by a factor of df_{opt}, and finally, the algorithm exits the downscale mode. In addition, the counter, i_{DS}, that triggers the downscaling attempts is reset. From that point onwards, the system is propagated as usual with the reduced rate constants. Once another $N = 10^5$ KMC steps are executed, the algorithm is invoked again to check for downscaling. This happens at KMC time $t_{KMC} = 228.7$. The second downscaling window $(t_s,t_f)_2$ is determined as $(228.7, 238.7)_2$ as illustrated in Figure 26. In the second attempt, the timescale separation between the fast and slow reactions is less than the user-defined minimum and thus, the attempt is aborted. We would like to note that an aborted attempt, due to insufficient timescale separation (decision block #1 in flowchart of Figure 22), has negligible overhead because the already generated unscaled trajectory is used to fill the KMC time interval $(t_s,t_f)_j$ for any j for which the attempt is aborted.

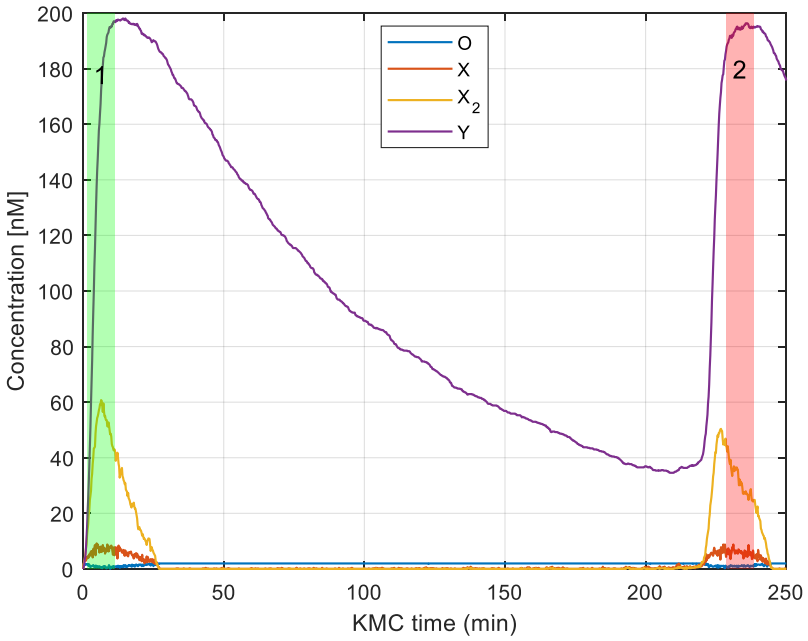


Figure 26: Trajectory of the system obtained with downscaling algorithm active. The green vertical band represents the first downscaling attempt, which was successful whereas the red vertical band the second attempt, which was aborted.

In the run just discussed, the tunable parameters were chosen such that the first downscaling attempt reduces the execution frequency of the fast processes up to the point that no more downscaling is permissible. To illustrate how the algorithm takes into account previous successful attempts, we performed

a different test run, choosing a different value for the parameter ζ so that the timescale separation is larger and therefore, higher reduction factors of the corresponding rate constants are possible. Thus, the simulation results presented below are obtained using ζ = 1000, w_{min} = 5, and β = 10. These parameters were chosen such that the computational cost of the first downscaling attempt is reduced and that the first successful downscaling would allow the second one to be accepted as well, i.e. by favouring accuracy, the first downscaling is less aggressive. The first attempt is successful, identifying df_{opt} = 72.1, and the relevant error and cost curves along with various objectives functions are plotted in Figure 27(a). The second attempt is also successful. The methodology deems that a further reduction by a factor of 1.3 is acceptable so that the overall downscale factor becomes 94, as shown in Figure 27(b). The time-ranges over which the two successful attempts took place are illustrated in Figure 28.

Based on the results of Figure 27, it is worth discussing a few important points. First, in all attempts, the calculation of the cost takes into account any previous accepted attempts. This is a crucial to propagate information about the actual cost of the downscaled trajectories generated in subsequent downscaling attempts. Second, for every downscaling attempt, the downscaled trajectories are obtained by reducing the *current* rate constants by powers of the user-provided df_b. Since the *current* rate constants are used, the reduction procedure is done in a relative manner during a downscaling attempt. For example, during the second attempt, the rate constants that have already been reduced by a factor of 72.1 (the df_{opt} of the first attempt), are further reduced by 5, 25, 125, etc. However, the decision procedures are performed using the absolute downscale factor as illustrated in Figure 27(b).

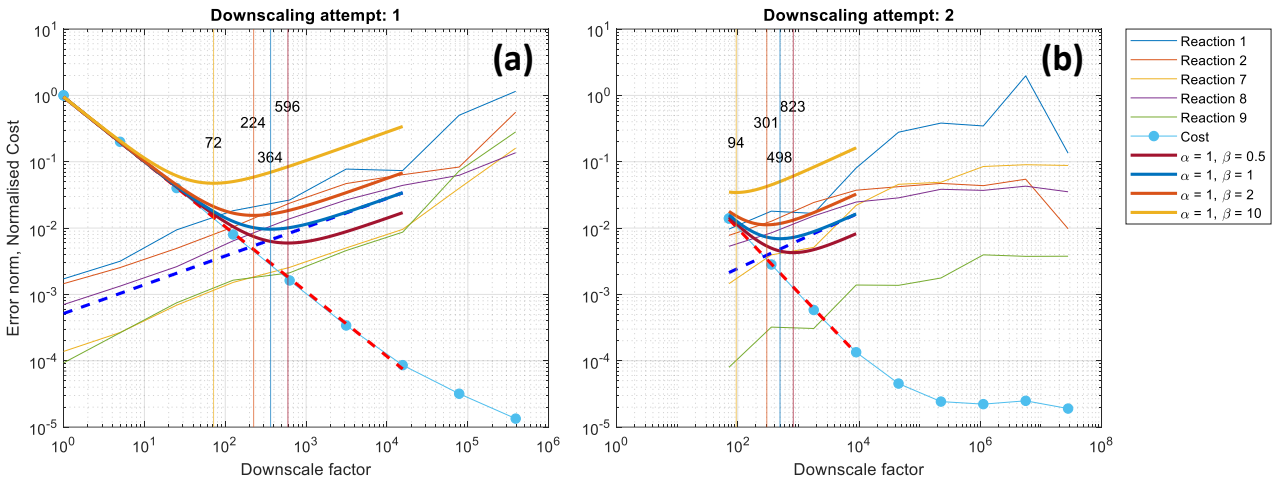


Figure 27: Scaling of the error norm and normalised cost with the corresponding objective functions for the two accepted downscaling attempts. On panel (b), the curves start at 72.1 which was the chosen df_{opt} during the 1st attempt.

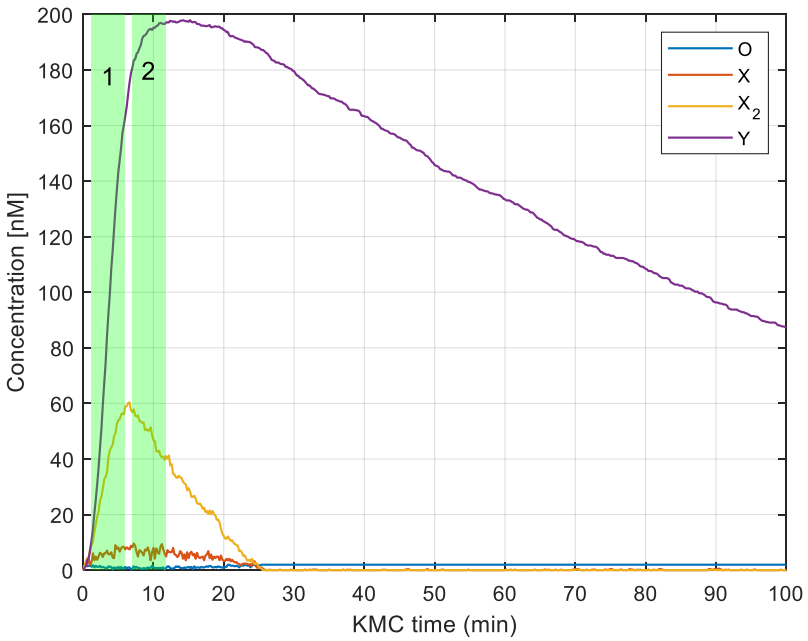


Figure 28: Location, in terms of KMC time, of the two accepted downscaling attempts denoted by the two vertical green bands.

5.3.3. Results and validation

We use the parameters of Table 6 with ζ=10 and run our benchmark model twice until a final time of 5×10^6 KMC minutes. For the first run, the downscaling methodology is disabled and therefore the simulation is executed from start to

finish using the initial rate constants. For the second run, the downscaling algorithm is enabled. The user-defined parameters values are as listed in Table 7 except for the weight on the error, β, which was chosen as $\beta = 1$ thus favouring the cost reduction. In the latter run, and following the procedure described above, the optimum downscale factor was identified as $df_{opt} = 125$ right from the first downscaling attempt with no further reductions for the rest of the run. The run using the original rate constants reached the final KMC time in 66 hours of wall time. On the other hand, the run in which the rates were reduced was completed in just 90 minutes. Consequently, the developed methodology accelerated the second run by 44 times, thereby achieving a significant reduction in the total computational cost.

At this point, we would like to discuss the achieved acceleration as just presented. The acceleration factor we obtained by comparing the runs above is underestimating the full capabilities of the developed methodology. The reason is that we have chosen the parameter of our benchmark chemical system, more specifically, $\zeta = 10$, so that we are able to run it using the initial rates within a reasonable amount of time (66 hours). Choosing a much larger value for ζ would have resulted in greater overall speedup, however running the original system would have been intractable without spending computational resources for an unreasonably high amount of time. As an example, we *estimate* that the original simulation with $\zeta = 100$ would have a runtime between 27 and 28 days, while the one with the downscaling enabled takes only 3.2 hours (parameters as Table 7, $w_{min} = 5$, $df_b = 12$, $\beta = 1$, $df_{opt} = 288$). Using the former estimate of 27-28 days, along with the actual runtime of 3.2 hours we obtained for the reduced system, the speed-up gain ranges from 200× to 207×. In principle, the speedup may be much higher than that in cases where the timescale separation is much larger than two to three orders of magnitude.

The main reason we have opted for a tractable original system is to enable the validation of the downscaling procedure. The stochastic nature of the KMC method makes it impossible to directly compare the equivalence of the original and the downscaled versions of the system in terms of concentrations with respect to time. Due to the on-the-fly reduction of certain rate constants, the two trajectory sets, original and downscaled, will not be identical. However, the aim of the downscaling procedure is to speed-up the computational simulation while

preserving the important features of the system under study and not the exact path in the concentrations space. In our benchmark model, the oscillatory behavior is the main feature, and here we examine whether the reduction in rates incurred a change on the dominant oscillation frequency. To investigate the oscillatory patterns and extract the dominant frequency, we calculate the autocorrelation function of the concentration of species Y and from the latter we calculate the power spectral density via a cosine transformation, as explained in detail in Section IV of Stamatakis and Mantzaris [122]. In Figure 29, we plot the spectral density of the trajectory of species Y for both the original and downscaled system. It is impressive to see that the dominant frequency of $f = 5\times10^{-3}$ min, which corresponds to a period of $T = 200$ min is obtained almost intact from the reduced system, thereby validating the developed method. More specifically, the on-the-fly reduction did not alter the dynamics of our benchmark system and, by choosing system-informed parameters for the algorithm, we are able to extract the system's main features in a fraction of the time as compared to the original system.

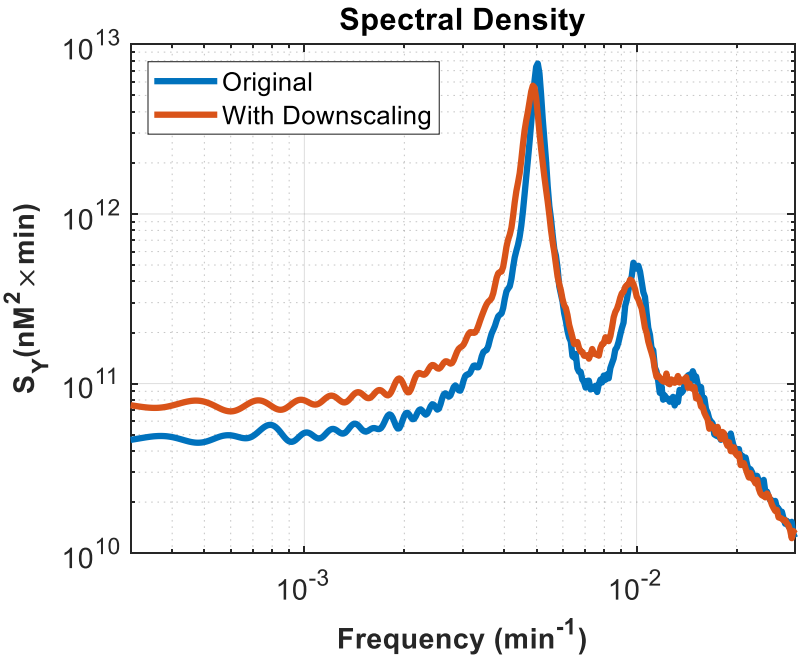


Figure 29: Spectral density of species Y for the original and downscaled system, both run for 5×10^6 KMC minutes.

5.4. Summary & conclusions

In this study, we developed a methodology to reduce on the fly the kinetic rate constants of very frequent processes in well-mixed chemical systems. At specific intervals during the course of the simulation, our algorithm generates multiple trajectory sets that utilise rate constants that are reduced by different factors, and collects data regarding the computational cost and the error introduced. Then, an objective function is formed and an optimisation problem is solved, to identify the downscale or reduction factor that achieves the best balance between accelerating the simulation with the least error. Our algorithm includes some well-defined, user-provided parameters to tune its performance, depending on the user's preference on favouring speed or accuracy. To assess the performance of our algorithm, we have chosen as our benchmark a chemical system from biology [122], which exhibits oscillatory behaviour.

The application of our algorithm on the benchmark system demonstrated reductions of the simulation runtime from 66 hours to 90 minutes, providing an acceleration factor of 44×. The reported acceleration factor probably underestimates the capabilities of the methodology since we have chosen the parameters of the benchmark system such that we may obtain the original run with a reasonable amount of time. We also estimate that acceleration factors of 200×-207× are easily obtained. In principle, much larger acceleration factors may be obtained if the timescale separation between the fast and slow reactions is greater.

This study comes to fill the gap in methods that tackle the timescale disparity on well-mixed systems especially regarding the quantification of error introduced because of the reduction of the rate constants of the fast processes. By having a quantifiable error, we can obtain trajectories that converge to the "true" solution, and therefore have confidence that the final results are reliable approximations of this "true" solution.

6. Concluding remarks and future work

The current thesis explored various approaches in accelerating Kinetic Monte Carlo simulations of reactive systems. The major inherent drawback of KMC is its serial nature, namely the scheduling and execution of computational steps one after the other. For this reason, industrially relevant length-scales are virtually impossible to simulate, at least for complex systems, with the current methodologies. In addition, certain chemical reaction systems are even more challenging because of the vastly different timescales on which the reactions take place. In practice, such systems are out of reach with current methods.

Chapter 3 investigated the performance of different scheduling data structures. Queuing systems available in the literature were implemented in the KMC software *Zacros*, and another one was developed to address specific needs. It was found that the queuing system that delivers the best performance depends on the chemical system studied. Based on our results, the compiler-induced optimisations provide an advantage to the array-based queueing systems as compared to the node-based ones.

Chapter 4 investigated the scaling performance of the Time-Warp algorithm, as implemented in *Zacros*. Time-Warp enables distributed, on-lattice, KMC simulations of reactive systems and it is the first-of-its-kind approach implemented in a general-purpose code. In addition, performance optimisation benchmarks were conducted to tune the user-defined parameters of Time-Warp so that the best possible performance is delivered. For sufficiently large lattices, distributed simulations offer a significant speedup as compared to the serial runs and the speedup itself depends on the system studied. The performance investigation studies revealed that the frequency of global communications among the different processors does not affect the performance, provided that there is enough memory to save snapshots. Conversely, the snapshot saving interval may decrease the performance dramatically if it is not chosen appropriately.

Lastly, Chapter 5 focused on tackling the timescale disparity on well-mixed systems by developing an on-the-fly methodology for downscaling rate constants. The algorithm developed reduces the rate constants of fast reaction channels in

an optimal, data-driven way, by balancing the introduction of error and the reduction of the computational cost. Most importantly, it provides metrics on the error introduced, and along with some user tunable parameters, one may favour accuracy or computational efficiency according to their needs.

The conducted research is by no means exhaustive in what can be done to accelerate KMC simulations. It provided, however, a few starting points for further exciting research. Extending the downscaling algorithm to on-lattice systems will have a significant impact on simulating industrially relevant systems accurately. Further studies and developments on the Time-Warp algorithm will soon make it possible to break the billion-site KMC simulation barrier, which is necessary towards capturing the formation of patterns on catalytic surfaces. Apart from direct extensions of the current research, one could investigate other avenues as well in terms of software and hardware paradigms. The use of Graphics Processing Units (GPUs) is becoming the norm for accelerating computationally intensive calculations. It may be worth investigating the applicability of such hardware acceleration for on-lattice KMC simulations. In addition, Field-Programmable Gate Arrays (FPGAs), i.e. integrated circuits that can be configured to perform specific functions, may be worth investigating for executing the computationally intensive procedures of a KMC simulation. Lastly, the development of mathematical methods related to the perturbation and convergence analysis for reactive systems would provide the necessary theoretical support on top of which KMC simulations can be developed. This is especially needed on approximate KMC algorithms with error bounds in order to tackle stiff systems and reach relevant timescales.

The efforts for reducing the computational cost of stiff systems, well-mixed or on-lattice, may be aided by methods developed in other fields. One such method is the Graph Transformation (GT) method as applied to finite-state discrete-time markovian systems [125-130]. For such systems, one needs to know all the accessible states and the transition rules among them. Then, the system is represented as a graph, with the nodes representing the states and the edges connecting the nodes representing the transition probabilities. Stiff systems arise in the presence of metastable states which are visited extremely frequently as compared to the rarely visited states. This “flickering” in metastable states may be eliminated via the GT method. The main idea of the GT method is

to remove nodes from a graph gradually in such a way that the average properties of interest are left unchanged. This is done by renormalizing, i.e. adjusting the transition or branching probabilities to preserve the system's average properties, such as the mean first-passage time (MFPT). The complexity of the GT method depends on the average number of connections of nodes, depends on the heterogeneity of the distribution of the number of connections, and especially the renormalization procedure is shown to scale as $O(|S|^3)$ [125, 130] where $|S|$ the number of nodes in the graph. In addition, the linear scaling of the memory requirements with respect to the graph size [125] might be substantial for extensive systems. There are however certain techniques to reduce memory usage [128]. In the context of KMC, stiff systems arise in the presence of reactions being executed extremely frequently as compared to other rarely executed reaction channels. Accelerating such a KMC simulation would entail reducing the execution frequency of the fast events without affecting the dynamics of the system. Therefore, the ideas and applications of the GT method would be worth investigating given that the GT method has been shown to perform well in various applications [130].

Another promising method to tackle stiffness in KMC simulations is the kinetic Path Sampling (kPS) [130, 131], which is statistically equivalent to the usual KMC algorithm [131]. The main ideas around the kPS are (a) a state reduction procedure is applied to a set of metastable states or a trapping basin, and (b) sampling a single exit state and the exit time that the system escapes from the subnetwork of metastable states. The kPS does not require an a priori knowledge of the trapping states since the mapping of these states is done iteratively and they are eliminated through path factorization[131]. The kPS method may be used to accelerate a KMC simulation when the latter becomes extremely inefficient due to fast processes. When implemented and performed in a correct manner, a stochastic simulation using kPS and KMC interchangeably, and as needed, is always more efficient than the KMC simulation alone [131]. An exciting direction of research could be the investigation of the generality of the kPS method, and especially to chemical reactions systems relevant to catalysis.

Building upon the work presented in chapter 5, attempts to accelerate on-lattice KMC simulations without introducing significant error could benefit from existing studies such as that of Chatterjee & Voter [84] in which states are lumped

together as belonging to a wider energetic super basin. If states within a super basin are visited very often, the corresponding barriers are raised i.e. their rate constants are decreased so the execution of the slower events is favoured. Following similar approach, Dybeck and co-workers [86] developed an acceleration scheme for KMC simulations that operates on the reaction channels and not by monitoring individual lattice configurations as in Ref. [84]. The main idea of the work done by Dybeck and co-workers is to reduce the rate constants of reactions that are QE and that have been executed a certain amount of times. Once a non-QE event occurs, all rate constants are unscaled to their original values. Along the same lines, Danielson and co-workers developed the Staggered Quasi-Equilibrium Rank-based Throttling for Steady-state algorithm, abbreviated SQERTSS [85]. This algorithm monitors all the executed processes during the simulation and classifies them into ranks based on their frequency. Once enough data have been gathered, a test is performed to assess whether downscaling of fast processes or upscaling of slow processes can be performed and by which factor. The main difference of SQERTSS as compared to the already presented methods is that the slow processes are upscaled so that they occur more often. A common characteristic of the above methods is that they introduce an approximation error due to the down- or up-scaling of the rate constants. More importantly, this error is not quantified and the success of the algorithm depends on the nature of the system [132]. A new research direction could endeavour in combining the above works along with our rate scaling algorithm for the acceleration of on-lattice KMC simulations with quantifiable error.

Lastly, in further development efforts, one could borrow ideas from research performed in other fields, for example the work of Rosta & Hummer [133]. These authors develop a formula to calculate the error and efficiency of simulated tempering (ST) simulations, where ST is a method to accelerate conformational sampling that is limited by the slow interconversion rate between the energetic basins.

Appendix I

1. Binary heap

In this section, we briefly discuss the operations implemented in the binary heap data-structure. A new element is inserted at the bottom level and at the left-most available position as illustrated by Figure 30(a) where the new node is the orange one with value 0.3. Comparison of this value is performed with that of its parent node and if the new node has priority over its parent, the two nodes are swapped. The value comparisons and swaps continue until the new node has a parent with smaller value than itself (Figure 30(c)), in which case no further swaps are performed. Executing the previous steps at every insertion ensures that partial order is never violated in the binary heap.

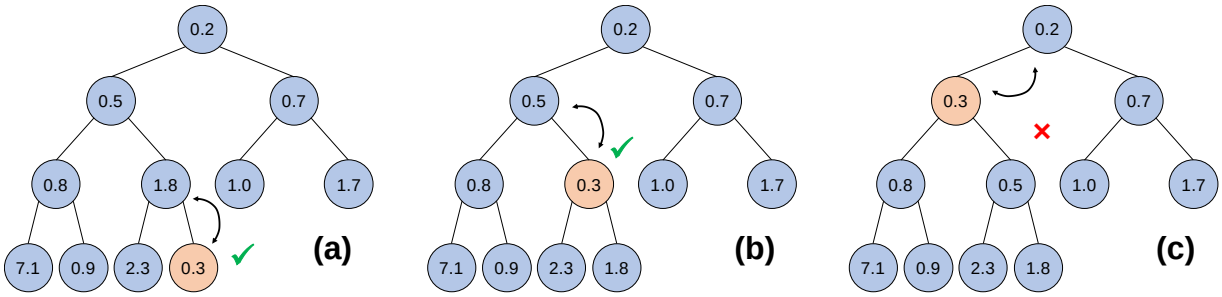


Figure 30: Graphical representation of the insertion operation

As for the removal operation, referring to Figure 31, we want to remove the element with value 0.5, coloured orange (Figure 31(a)). The last element of the binary heap, in this case the one with value 2.3, coloured green, replaces the element to be removed (Figure 31(b)). Now, both children have priority over the green node. The smallest one is swapped with the parent node until the green node reaches a level that does not violate partial order (Figure 31(d)). An optimised alternative of this algorithm, that does not require swapping the elements, goes as follows: the element to be removed is actually removed, leaving a gap in its place. Then, the last element takes the place of the gap only virtually, that is, when a numerical comparison is performed between a node and the gap, the empty node appears to have the value of the last node. The last node overwrites the empty node only when the float-up or sink-down operations are

completed. Using this approach, one avoids performing swaps that may have an impact on performance.

Due to its completeness, the binary heap can be represented using an array as shown in Figure 32. The nodes are numbered from left to right and from the top to the bottom. In one-based arrays, that is, the index of the first element is one, for an arbitrary node *i* we may find the indices of parent and child nodes as follows:

- left child: *2i*
- right child: *2i+1*
- parent: $\left\lfloor \frac{i}{2} \right\rfloor$

Figure 32 illustrates the correspondence between an occurrence time and its position on the binary heap. In addition, Table 8 shows the two extra arrays of the data-structure: array “labels” stores the labels of each one of these individual lattice processes, while array “map” is used to map the labels to the correct occurrence times. The “indices” and “occurrence times” rows are the same as the ones already presented in Figure 32. The “Labels” row links the lattice processes

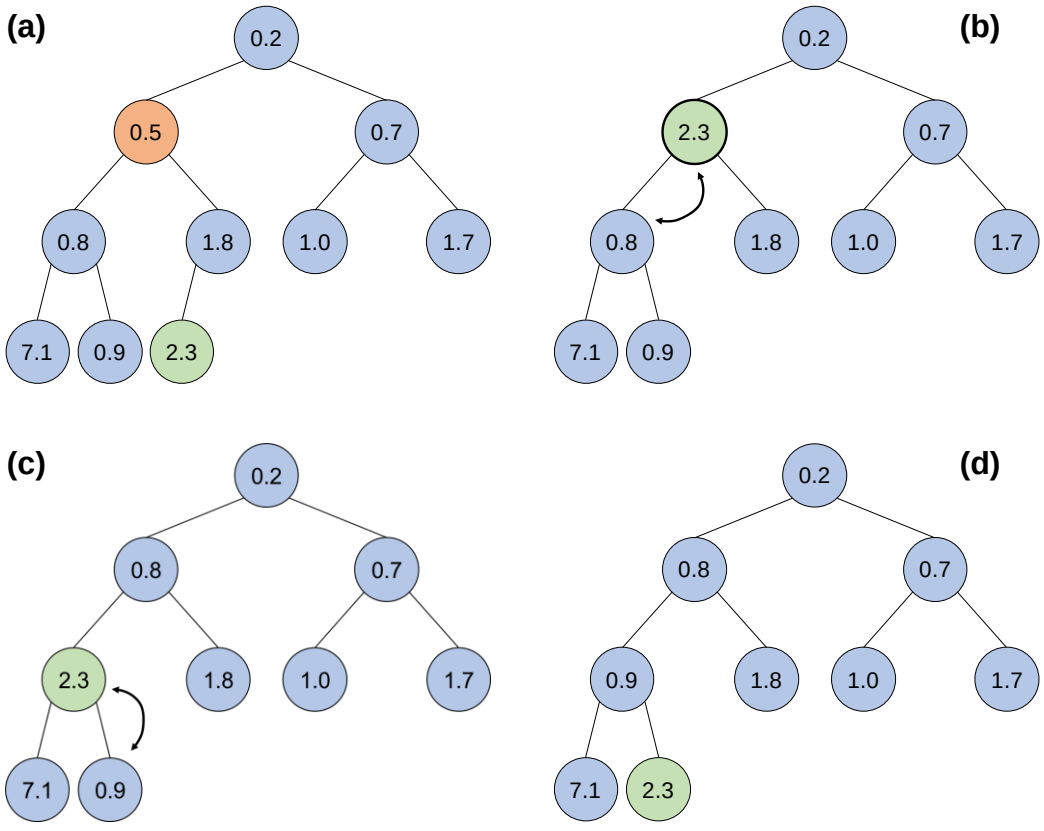


Figure 31: Graphical representation of the deletion operation. See text for details

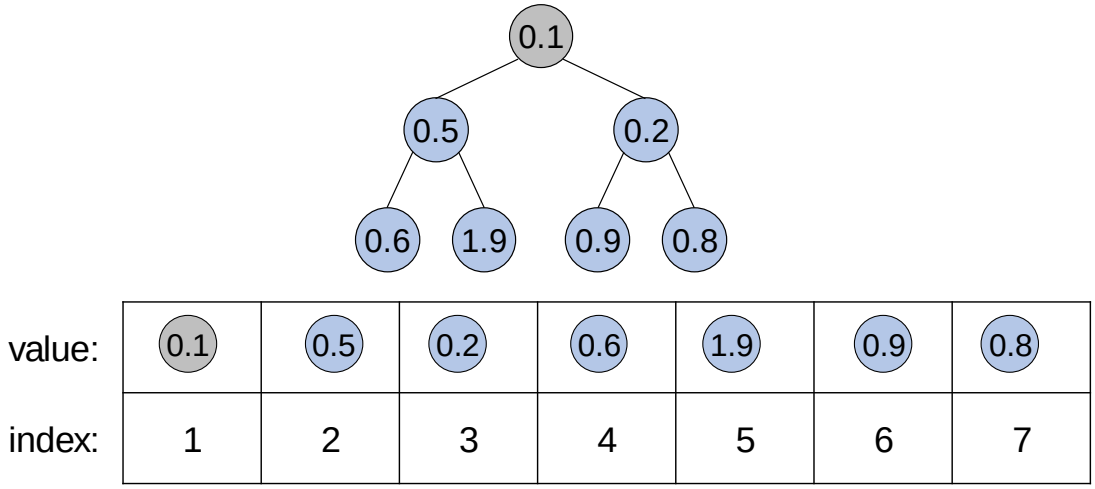


Figure 32: Representation of a binary heap using an array

with the corresponding occurrence times, e.g. the lattice process with label 3 has an occurrence time of 0.1 time units, the lattice process with label 7 has an occurrence time of 0.5 time units etc. For an efficient implementation, the "Labels" array is not enough and we show why. Let us assume that the lattice process with label 3 occurred and as a result, the lattice process with label 4 has been invalidated. In order to find the occurrence time of the lattice process with label 4 and remove it, one has to iterate over the "Labels" array, an operation with expected computational time O(N), N being the size of the "Labels" array. If one has another array that maps the position of every label into the "occurrence times" array then the operation described above takes constant time, O(1). The "Map" array, bottom row of Table 8, links an arbitrary label with its position on the "Labels" array and is used as follows: consider the query *what is the position of the lattice process with label 4*? To answer this, we evaluate the expression *Map[4]*, namely we retrieve the element found at the 4^{th} position of the "Map" array and we find that this is 7. We now know that the occurrence time of the lattice process with label 4 can be found on the 7^{th} position of the "occurrence times". Generalising this concept, we are able to retrieve the occurrence time of an arbitrary lattice process *j* by evaluating *Occurrence_times[Map[j]]*.

Indices	1	2	3	4	5	6	7
Occurrence_times	0.1	0.5	0.2	0.6	1.9	0.9	0.8
Labels	3	7	1	6	2	5	4
Map	3	5	1	7	6	4	2

Table 8: Arrays used for an efficient implementation of the binary heap data-structure (refer to text for the correspondence among the illustrated rows)

2. Pairing heap

As compared to the binary heap, the pairing heap is more flexible in the sense that it allows multiple children per node. Due of this feature, implementation might become tricky and computationally demanding if one is tempted to pre-allocate a sufficiently large number of pointers in order to build a pairing heap exactly as represented in Figure 33(a). Instead, the *binary heap representation* is a more suitable way to implement a pairing heap. In this representation, each node has only three pointers pointing to: **(i)** its left-most child, **(ii)** its next sibling, **(iii)** its previous node. As their name suggests, sibling nodes are those that have the same parent. The top node has only one non-null pointer pointing to its left-most child and has no sibling and no previous node. Graphically, the binary tree representation of the pairing heap of Figure 33(a) is illustrated in Figure 33(b) where the black arrows represent the left-most child pointer, the red arrows represent the next sibling and the previous pointers are omitted for visual clarity. The previous pointer of a certain node *X* points back to the node whose either the left-most child pointer or the next sibling pointer points to *X*. For example, referring to Figure 33(b), the previous node for element 15.1 in the second level is the node 0.2 whereas the previous node for element 2.5 in the third level is the node 1.0. This representation provides a way to implement, in a simpler way than that of Figure S4a, the pairing heap data-structure and store elements without any restrictions on e.g. the maximum number of children.

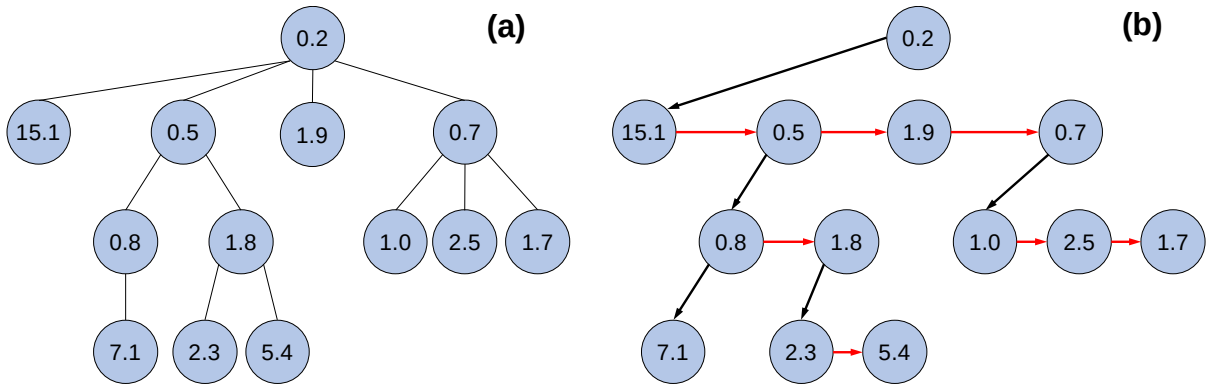


Figure 33: (a) Conventional representation of a pairing heap. (b) Binary tree representation of the pairing heap shown in (a).

3. Energetics for the water-gas shift reaction model

In our simulations, the reference set used was [Pt(111), CO, H_2O, H_2], which is the catalytic surface and three of the gas phase species. The formation energies of the surface species were taken as follows:

Surface Species	Form. Energy (eV)
CO*	-2.08
H_2O*	-0.36
OH*	0.83
O*	1.30
H*	-0.62
COOH*	-1.49

Table 9: Formation Energies of the surface species

The numerical values involved in the calculation of the rate parameters used in our simulations are summarised in the following table. For adsorption events, being spontaneous with zero activation energy, we calculated the pre-exponential using the equation:

$$A_{fwd} = \frac{A_{site}}{\sqrt{2\,\pi\,m\,k_B T}}$$

where A_{site} is the effective area of the catalytic site where the reaction takes place and it is taken as 1 $Å^2$, m is the mass of the adsorbing molecule, k_B is the Boltzmann's constant and T is the temperature, taken as 500 K. For surface events, the pre-exponential is estimated as $A_{fwd} = k_B T/h_{Planck}$ at T=500 K.

Elementary Event	A_{fwd} (s^{-1})	A_{fwd}/A_{rev}	$E_{a,fwd}$ (eV)
$CO(g) + * \leftrightarrow CO^*$	2.23×10^{7} bar^{-1}	2.14×10^{-6} bar^{-1}	0.00
$H_2(g) + 2* \leftrightarrow 2H^*$	8.30×10^{7} bar^{-1}	7.97×10^{-6} bar^{-1}	0.00
$H_2O(g) + * \leftrightarrow H_2O^*$	2.78×10^{7} bar^{-1}	2.67×10^{-6} bar^{-1}	0.00
$H_2O^* + * \leftrightarrow OH^* + H^*$	1.04×10^{13}	1.00	0.78

$OH^* + * \leftrightarrow\ O^* + H^*$	1.04×10^{13}	1.00	0.94
$CO^* + OH^*\ \leftrightarrow COOH^* + *$	1.04×10^{13}	1.00	0.41
$COOH^* + * \rightarrow CO_2(g) + H^* + *$	1.04×10^{13}	-	0.85
$CO^* + O^* \rightarrow CO_2(g) +\ 2*$	1.04×10^{13}	-	0.99

Table 10: Rate parameters of the elementary events taken into account for the water-gas shift reaction model. The last two steps are considered as irreversible.

4. Time scaling of operations

A way to estimate, at least theoretically, the performance of an algorithm is by counting the number of elementary operations executed by that algorithm. For example, referring to Figure 30(a), let us estimate the time scaling of the insertion operation in the binary heap. In the worst case, an inserted value will float up to become the top node. The number of value comparisons and swaps that have to be executed are equal to $\lfloor log_2 N \rfloor$, where *N* is the number of nodes in the binary heap. Assuming that each such operation takes a constant amount of time to be executed, the expected time scaling of the insertion operation is $O(\log_2 N)$ in the worst case. Following similar logic, we can show that the removal operation in the binary heap is also $O(\log_2 N)$. If we want to remove all elements from our binary heap then we have to perform the remove operation N times at $O(\log_2 N)$ cost each. Therefore, the total time needed to complete this task is $O(N \log_2 N)$.

Depending on the algorithm or operation being studied, its time scaling may vary and various functions are used to describe their time complexity. A few of these functions are plotted in Figure 34. We observe that $O(\log_2 N)$ scaling is increasing slowly with respect to the size of the system (or equivalently, with respect to the size of the data the operation/algorithm is applied to). $O(N)$ and $O(N \log_2 N)$ increase at almost the same rate, apart from the region corresponding to small system size.

Their similarity is the reason for which the binary and pairing heaps seem to scale linearly in our TPD benchmark results.

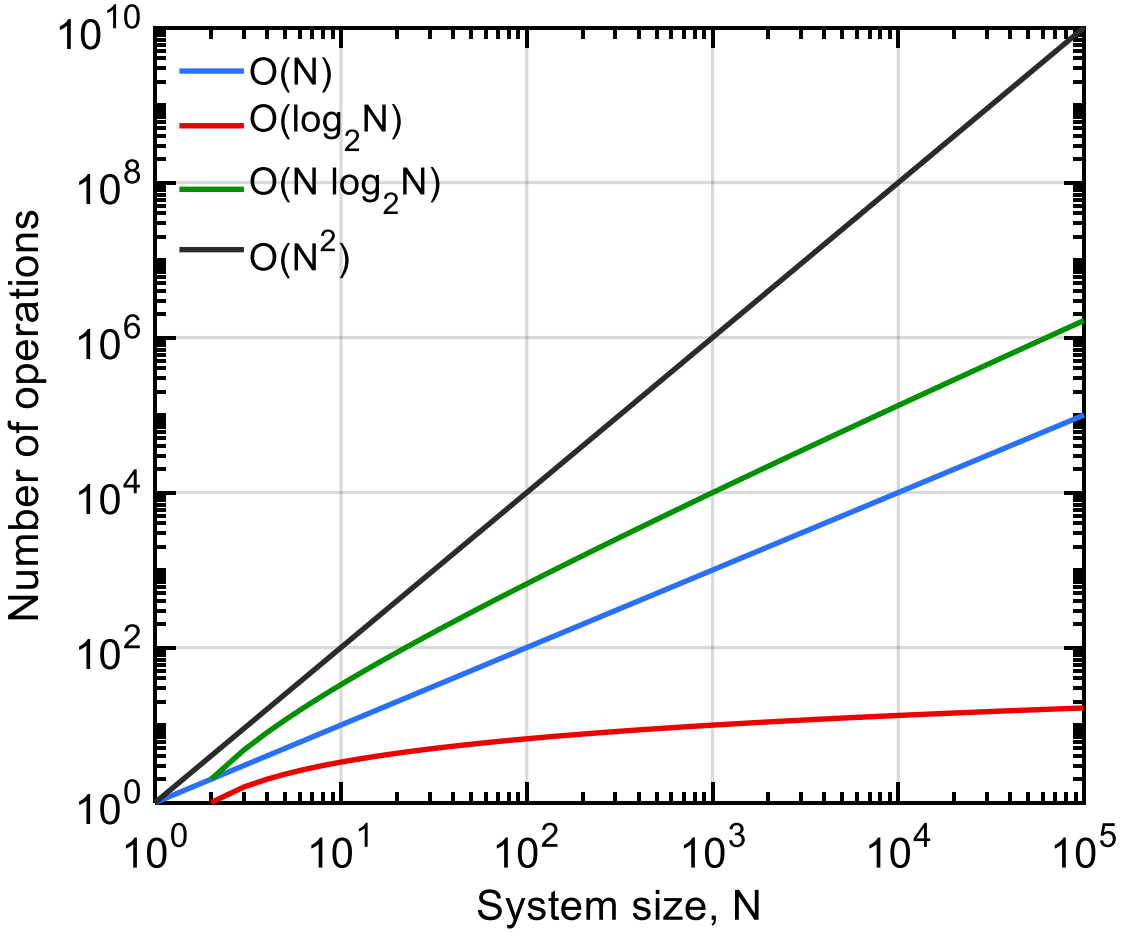


Figure 34: Graphical representation of various time complexity functions used to describe the number of operations performed by an algorithm.

Appendix II

A major difference of the Modified Next Reaction Method (Mod-NRM) by Anderson [46] as compared to the First Reaction Method by Gillespie [43] is that the former consumes just one random number per KMC step, excluding the initialisation stage of the simulation. In Mod-NRM, the reaction channels affected by the executed reaction have their inter-arrival times adjusted in order to reflect the change of their propensities. Due to the change in the representation of firing times in the Mod-NRM, the loss of memory property is not invoked [46], i.e. the firing times of the reactions are affected by the previous history and not just by the state of the system, i.e. the molecular populations.

The coupling between the fast and slow reactions, and the use of internal times T_i, combined with not using the loss of memory property causes the occurrence times of the downscaled trajectory with df = 1 to differ from the unscaled trajectory, even though the rate constants are the same and the same random numbers are used to generate the firing times of the slow reactions. The latter is also the source of the non-zero error norm we have shown for df = 1 (Figure 4). This error norm depends on the timescale separation between the fast and slow reactions.

Here, we demonstrate computationally the convergence of the firing times with respect to the timescale separation. Equivalently, the error norm gets smaller as the timescale separation becomes larger. To do so, we use the following “toy-model”:

$$A \xrightarrow{k_1} B$$

$$B \xrightarrow{k_2} A$$

$$B \xrightarrow{k_3} C$$

We use the Mod-NRM to propagate the above system in time. We start with an initial population of $A_0 = 50$ molecules and $B_0 = 30$ molecules. For the rate constants, we have chosen

$$k_1 = 5\times10^{N}\ \text{s}^{-1}$$

$$k_2 = 8\times10^{N}\ \text{s}^{-1}$$

$$k_3 = 1\ \text{s}^{-1}$$

where the exponent *N* controls the timescale separation. We use two different random number streams: one for the fast and reversible pair of reactions A ↔ B and another for the slow reaction B → C. The first random number stream is initialized randomly, whereas the second is initialized with a specific seed and is kept the same for all the runs.

To collect statistics on the occurrence time of the first B → C reaction, we run the system described above, with N = 1, for R = 5000 times. The distribution of the firing times of interest is plotted on Figure 35. It is easy to see that, despite the fact that the same random number is used to generate the firing time of the first B → C transition, the latter firing time is not exactly the same on every run.

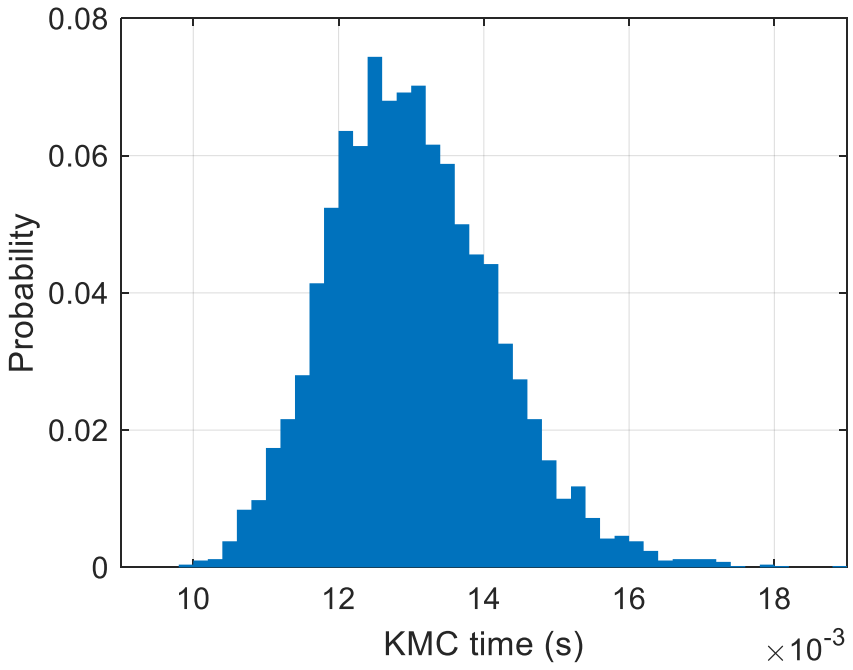


Figure 35: Distribution of the firing times of the first B → C reaction. The samples were collected from R = 5000 repetitions of the KMC simulation.

To investigate the dependence of the distribution on the timescale separation, we run the same simulation as above for different values of the exponent N. We collect samples from R=5000 KMC runs for each value of N. Our results are presented on Figure 36.

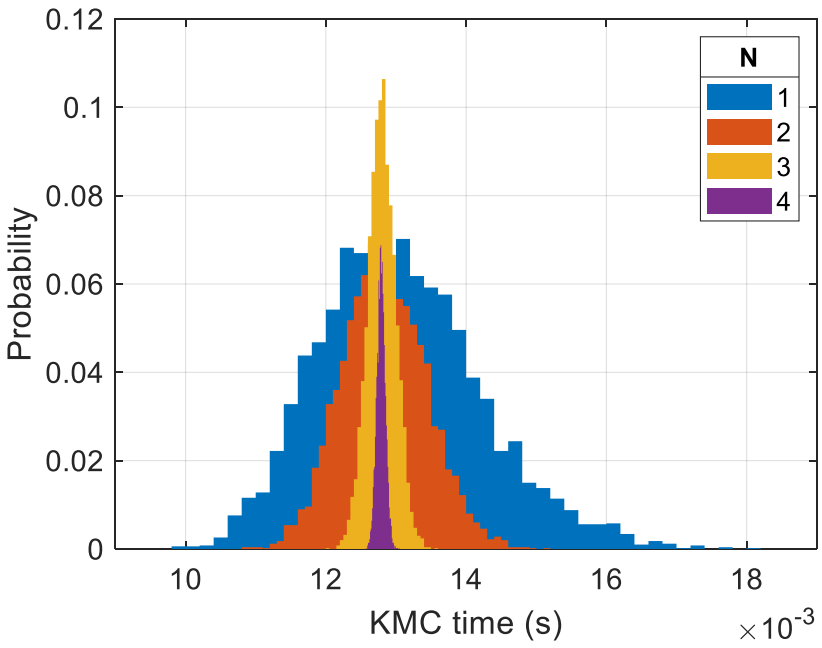


Figure 36: Distributions of the firing time of the first B → C reaction.

From the above results, we observe that the distribution becomes narrower as the timescale separation between the fast and slow reactions increases, equivalently, the standard deviation of the distributions decreases, whereas the mean of the distributions seems unchanged. Our results suggest that the firing times of the slow reaction, here B → C, would converge to a single value, had the A ↔ B reversible isomerisation was infinitely fast. However, in our toy-model here, and our benchmark model as introduced in the main text, the fast reversible pairs of reactions are not infinitely fast. The latter along with the with-memory Mod-NRM gives rise to the non-zero error norm, as defined in the main text, when the same system is simulated under the same conditions and the same random numbers for the slow reactions.

References

1. Randall, H., R. Doepper, and A. Renken, *Reduction of nitrogen oxides by carbon monoxide over an iron oxide catalyst under dynamic conditions.* Applied Catalysis B-Environmental, 1998. **17**(4): p. 357-369.
2. Freund, H.J., et al., *CO Oxidation as a Prototypical Reaction for Heterogeneous Processes.* Angewandte Chemie-International Edition, 2011. **50**(43): p. 10064-10094.
3. Deluga, G.A., et al., *Renewable hydrogen from ethanol by autothermal reforming.* Science, 2004. **303**(5660): p. 993-997.
4. Erisman, J.W., et al., *How a century of ammonia synthesis changed the world.* Nature Geoscience, 2008. **1**(10): p. 636-639.
5. Schlogl, R., *Catalytic synthesis of ammonia - A "never-ending story"?* Angewandte Chemie-International Edition, 2003. **42**(18): p. 2004-2008.
6. Marnellos, G. and M. Stoukides, *Ammonia synthesis at atmospheric pressure.* Science, 1998. **282**(5386): p. 98-100.
7. Stamatakis, M., *Kinetic modelling of heterogeneous catalytic systems.* Journal of Physics-Condensed Matter, 2015. **27**(1).
8. Honkala, K., et al., *Ammonia synthesis from first-principles calculations.* Science, 2005. **307**(5709): p. 555-558.
9. Norskov, J.K., et al., *Ch.8: The Electronic Factor in Heterogeneous Catalysis*. Fundamental Concepts in Heterogeneous Catalysis. 2014, Hoboken, NJ, USA: John Wiley & Sons, Inc.
10. van Duin, A.C.T., et al., *ReaxFF: A reactive force field for hydrocarbons.* Journal of Physical Chemistry A, 2001. **105**(41): p. 9396-9409.
11. Ludwig, J. and D.G. Vlachos, *Ab initio molecular dynamics of hydrogen dissociation on metal surfaces using neural networks and novelty sampling.* Journal of Chemical Physics, 2007. **127**(15).
12. Foppa, L., et al., *Adlayer Dynamics Drives CO Activation in Ru-Catalyzed Fischer Tropsch Synthesis.* ACS Catalysis, 2018. **8**(8): p. 6983-6992.
13. Reuter, K. and M. Scheffler, *First-principles kinetic Monte Carlo simulations for heterogeneous catalysis: Application to the CO oxidation at RuO2(110).* Physical Review B, 2006. **73**(4).
14. Jansen, A.P.J., *An Introduction to Kinetic Monte Carlo Simulations of Surface Reactions*. 2012: Springer, Berlin, Heidelberg.
15. Stamatakis, M. and D.G. Vlachos, *Unraveling the Complexity of Catalytic Reactions via Kinetic Monte Carlo Simulation: Current Status and Frontiers.* ACS Catalysis, 2012. **2**(12): p. 2648-2663.
16. Andersen, M., C. Panosetti, and K. Reuter, *A Practical Guide to Surface Kinetic Monte Carlo Simulations.* Frontiers in Chemistry, 2019. **7**.
17. Pineda, M. and M. Stamatakis, *Kinetic Monte Carlo simulations for heterogeneous catalysis: Fundamentals, current status, and challenges.* The Journal of Chemical Physics, 2022. **156**(12): p. 120902.
18. Dixon, A.G. and M. Nijemeisland, *CFD as a design tool for fixed-bed reactors.* Industrial & Engineering Chemistry Research, 2001. **40**(23): p. 5246-5254.
19. Deutschmann, O., *Ch.6.6: Computational Fluid Dynamics Simulation of Catalytic Reactors*, in *Handbook of Heterogeneous Catalysis*. 2008, Wiley-VCH Verlag GmbH & Co. KGaA: Weinheim, Germany. p. 1811-1828.
20. Duran, J.E., M. Mohseni, and F. Taghipour, *Modeling of annular reactors with surface reaction using computational fluid dynamics (CFD).* Chemical Engineering Science, 2010. **65**(3): p. 1201-1211.

21. Schaefer, C. and A.P.J. Jansen, *Coupling of kinetic Monte Carlo simulations of surface reactions to transport in a fluid for heterogeneous catalytic reactor modeling.* Journal of Chemical Physics, 2013. **138**(5).
22. Maestri, M. and A. Cuoci, *Coupling CFD with detailed microkinetic modeling in heterogeneous catalysis.* Chemical Engineering Science, 2013. **96**: p. 106-117.
23. Mei, D.H. and G. Lin, *Effects of heat and mass transfer on the kinetics of CO oxidation over RuO2(1 1 0) catalyst.* Catalysis Today, 2011. **165**(1): p. 56-63.
24. Zhdanov, V.P., *Arrhenius Parameters for Rate-Processes on Solid-Surfaces.* Surface Science Reports, 1991. **12**(5): p. 183-242.
25. Murzin, D.Y., *On surface heterogeneity and catalytic kinetics.* Industrial & Engineering Chemistry Research, 2005. **44**(6): p. 1688-1697.
26. Stamatakis, M., *Kinetic modelling of heterogeneous catalytic systems.* Journal of Physics: Condensed Matter, 2015. **27**(1): p. 013001.
27. Stamatakis, M. and D.G. Vlachos, *A graph-theoretical kinetic Monte Carlo framework for on-lattice chemical kinetics.* Journal of Chemical Physics, 2011. **134**(21).
28. Imbihl, R. and G. Ertl, *Oscillatory Kinetics in Heterogeneous Catalysis.* Chemical Reviews, 1995. **95**(3): p. 697-733.
29. Schuth, F., B.E. Henry, and L.D. Schmidt, *Oscillatory Reactions in Heterogeneous Catalysis.* Advances in Catalysis, Vol 39, 1993. **39**: p. 51-127.
30. Lubachevsky, B.D., *Efficient Parallel Simulations of Dynamic Ising Spin Systems.* Journal of Computational Physics, 1988. **75**(1): p. 103-122.
31. Jefferson, D.R., *Virtual Time.* ACM Transactions on Programming Languages and Systems, 1985. **7**(3): p. 404-425.
32. Savva, G.D. and M. Stamatakis, *Comparison of Queueing Data-Structures for Kinetic Monte Carlo Simulations of Heterogeneous Catalysts.* The Journal of Physical Chemistry A, 2020. **124**(38): p. 7843-7856.
33. Ravipati, S., et al., *Coupling the time-warp algorithm with the graph-theoretical kinetic Monte Carlo framework for distributed simulations of heterogeneous catalysts.* Computer Physics Communications, 2022. **270**.
34. Nielsen, J., et al., *Parallel kinetic Monte Carlo simulation framework incorporating accurate models of adsorbate lateral interactions.* Journal of Chemical Physics, 2013. **139**(22).
35. Stamatakis, M. *Advanced Lattice-KMC Made Easy*. 2013; Available from: http://www.zacros.org.
36. Rahman, T.S., *Molecular-Dynamics Simulation of Surface Phenomena*, in *Characterization of Materials*, E.N. Kaufmann, Editor. 2012, Wiley: New York. p. 253-264.
37. Voter, A.F., *Introduction to the Kinetic Monte Carlo Method*, in *Radiation Effects in Solids*, K.E. Sickafus, E.A. Kotomin, and B.P. Uberuaga, Editors. 2007, Springer: Dordrecht. p. 1-23.
38. Andersen, M., C. Panosetti, and K. Reuter, *A practical guide to surface kinetic Monte Carlo simulations.* Frontiers in Chemistry, 2019. **7**: p. 202.
39. Reuter, K., *First-Principles Kinetic Monte Carlo Simulations for Heterogeneous Catalysis: Concepts, Status, and Frontiers*, in *Modeling and Simulation of Heterogeneous Catalytic Reactions: From the Molecular Process to the Technical System*, O. Deutschmann, Editor. 2011, Wiley-VCH: Weinheim. p. 71-111.
40. Darby, M.T., S. Piccinin, and M. Stamatakis, *First principles-based kinetic Monte Carlo simulation in catalysis*, in *Physics of Surface, Interface and Cluster Catalysis*, H. Kasai and M.C.S. Escaño, Editors. 2016, IOP Publishing Ltd: Bristol, UK. p. 4.1-4.38.
41. Van Kampen, N.G., *Stochastic processes in physics and chemistry*. Vol. 1. 1992, Amsterdam: North-Holland: Elsevier.

42. Matera, S., et al., *Progress in Accurate Chemical Kinetic Modeling, Simulations, and Parameter Estimation for Heterogeneous Catalysis.* ACS Catalysis, 2019. **9**(8): p. 6624-6647.
43. Gillespie, D.T., *A general method for numerically simulating the stochastic time evolution of coupled chemical reactions.* Journal of Computational Physics, 1976. **22**(4): p. 403-434.
44. Jansen, A.P.J., *Monte Carlo simulations of chemical reactions on a surface with time-dependent reaction-rate constants.* Computer Physics Communications, 1995. **86**(1-2): p. 1-12.
45. Gibson, M.A. and J. Bruck, *Efficient exact stochastic simulation of chemical systems with many species and many channels.* Journal of Physical Chemistry A, 2000. **104**(9): p. 1876-1889.
46. Anderson, D.F., *A modified next reaction method for simulating chemical systems with time dependent propensities and delays.* Journal of Chemical Physics, 2007. **127**(21).
47. Gillespie, D.T., *Exact Stochastic Simulation of Coupled Chemical-Reactions.* Journal of Physical Chemistry, 1977. **81**(25): p. 2340-2361.
48. Lukkien, J.J., et al., *Efficient Monte Carlo methods for the simulation of catalytic surface reactions.* Physical Review E: Statistical Physics, Plasmas, Fluids, and Related Interdisciplinary Topics, 1998. **58**(2): p. 2598-2610.
49. Chatterjee, A. and D.G. Vlachos, *An overview of spatial microscopic and accelerated kinetic Monte Carlo methods.* Journal of Computer-Aided Materials Design, 2007. **14**(2): p. 253-308.
50. Nielsen, J.H., J. Hetherington, and M. Stamatakis. *ARCHER eCSE01-001 technical report: Zacros software package development: Pushing the frontiers of kinetic Monte Carlo simulation in catalysis*. 2015 August 26, 2020]; Available from: https://www.archer.ac.uk/community/eCSE/eCSE01-001/eCSE01-001_ZACROS_technical_report.pdf.
51. Stamatakis, M., et al., *Zacros: Advanced lattice-KMC made easy*. 2020: London, UK.
52. Plimpton, S., et al., *Crossing the mesoscale no-man's land via parallel kinetic Monte Carlo.* Sandia Report SAND2009-6226, 2009. **1**.
53. Plimpton, S., A. Thompson, and A. Slepoy. *SPPARKS kinetic Monte Carlo simulator*. June 17, 2020]; Available from: http://spparks.sandia.gov/.
54. Leetmaa, M. and N.V. Skorodumova, *KMCLib: A general framework for lattice kinetic Monte Carlo (KMC) simulations.* Computer Physics Communications, 2014. **185**(9): p. 2340-2349.
55. Hoffmann, M.J., S. Matera, and K. Reuter, *kmos: A lattice kinetic Monte Carlo framework.* Computer Physics Communications, 2014. **185**(7): p. 2138-2150.
56. Chill, S.T., et al., *EON: software for long time simulations of atomic scale systems.* Modelling and Simulation in Materials Science and Engineering, 2014. **22**(5): p. 055002.
57. Lukkien, J.J. and A.P.J. Jansen. *CARLOS Project: a general purpose program for the simulation of chemical reactions taking place at crystal surfaces*. June 17, 2020]; Available from: http://carlos.win.tue.nl/.
58. Jorgensen, M. and H. Gronbeck, *MonteCoffee: A programmable kinetic Monte Carlo framework.* Journal of Chemical Physics, 2018. **149**(11): p. 114101.
59. Kunz, L., F.M. Kuhn, and O. Deutschmann, *Kinetic Monte Carlo simulations of surface reactions on supported nanoparticles: A novel approach and computer code.* Journal of Chemical Physics, 2015. **143**(4): p. 044108.
60. Ramsey, S., D. Orrell, and H. Bolouri, *Dizzy: stochastic simulation of large-scale genetic regulatory networks.* Journal of bioinformatics and computational biology, 2005. **3**(2): p. 415-436.

61. Maarleveld, T.R., B.G. Olivier, and F.J. Bruggeman, *StochPy: A comprehensive, user-friendly tool for simulating stochastic biological processes.* PLoS One, 2013. **8**(11): p. e79345.
62. Kazeroonian, A., et al., *CERENA: ChEmical REaction Network Analyzer-A toolbox for the simulation and analysis of stochastic chemical kinetics.* PLoS One, 2016. **11**(1): p. e0146732.
63. Stamatakis, M. and D.G. Vlachos, *A graph-theoretical kinetic Monte Carlo framework for on-lattice chemical kinetics.* Journal of Chemical Physics, 2011. **134**(21): p. 214115.
64. Ziff, R.M., E. Gulari, and Y. Barshad, *Kinetic phase-transitions in an irreversible surface-reaction model.* Physical Review Letters, 1986. **56**(24): p. 2553-2556.
65. Stamatakis, M., Y. Chen, and D.G. Vlachos, *First-principles-based kinetic Monte Carlo simulation of the structure sensitivity of the water-gas shift reaction on platinum surfaces.* Journal of Physical Chemistry C, 2011. **115**(50): p. 24750-24762.
66. Darby, M.T., et al., *Carbon monoxide poisoning resistance and structural stability of single atom alloys.* Topics in Catalysis, 2018. **61**(5-6): p. 428-438.
67. Cormen, T.H., et al., *Introduction to Algorithms, Third Edition*. 2009: The MIT Press.
68. Fredman, M.L., et al., *The pairing heap: A new form of self-adjusting heap.* Algorithmica, 1986. **1**(1-4): p. 111-129.
69. Pugh, W., *Skip lists: A probabilistic alternative to balanced trees.* Communications of the ACM, 1990. **33**(6): p. 668-676.
70. Nielsen, J., et al., *Parallel kinetic Monte Carlo simulation framework incorporating accurate models of adsorbate lateral interactions.* Journal of Chemical Physics, 2013. **139**(22): p. 224706.
71. Ovesen, C.V., et al., *A kinetic-model of the water gas shift reaction.* Journal of Catalysis, 1992. **134**(2): p. 445-468.
72. Grabow, L.C., et al., *Mechanism of the water gas shift reaction on Pt: First principles, experiments, and microkinetic modeling.* Journal of Physical Chemistry C, 2008. **112**(12): p. 4608-4617.
73. Ratnasamy, C. and J.P. Wagner, *Water gas shift catalysis.* Catalysis Reviews - Science and Engineering, 2009. **51**(3): p. 325-440.
74. Prats, H., et al., *Kinetic Monte Carlo simulations of the water gas shift reaction on Cu (111) from density functional theory based calculations.* Journal of Catalysis, 2016. **333**: p. 217-226.
75. Yang, M. and M. Flytzani-Stephanopoulos, *Design of single-atom metal catalysts on various supports for the low-temperature water-gas shift reaction.* Catalysis Today, 2017. **298**: p. 216-225.
76. Stamatakis, M., et al. *Zacros: Advanced lattice-KMC made easy*. June 17, 2020]; Available from: http://zacros.org/.
77. Apostolopoulou, M., et al., *Quantifying pore width effects on diffusivity via a novel 3D stochastic approach with input from atomistic molecular dynamics simulations.* Journal of Chemical Theory and Computation, 2019. **15**(12): p. 6907-6922.
78. Stamatakis, M. and K. Zygourakis, *A mathematical and computational approach for integrating the major sources of cell population heterogeneity.* Journal of Theoretical Biology, 2010. **266**(1): p. 41-61.
79. Battaile, C.C., *The kinetic Monte Carlo method: Foundation, implementation, and application.* Computer Methods in Applied Mechanics and Engineering, 2008. **197**(41-42): p. 3386-3398.
80. Ustinov, E.A. and D.D. Do, *Application of kinetic Monte Carlo method to equilibrium systems: Vapour-liquid equilibria.* Journal of Colloid and Interface Science, 2012. **366**(1): p. 216-223.

81. Franco, A.A., et al., *Boosting Rechargeable Batteries R&D by Multiscale Modeling: Myth or Reality?* Chemical Reviews, 2019. **119**(7): p. 4569-4627.
82. Papanikolaou, K.G. and M. Stamatakis, *Chapter 7 - Toward the accurate modeling of the kinetics of surface reactions using the kinetic Monte Carlo method*, in *Frontiers of Nanoscience*, P. Grammatikopoulos, Editor. 2020, Elsevier. p. 95-125.
83. Chutia, A., et al., *A DFT and KMC based study on the mechanism of the water gas shift reaction on the Pd(100) surface.* Physical Chemistry Chemical Physics, 2020. **22**(6): p. 3620-3632.
84. Chatterjee, A. and A.F. Voter, *Accurate acceleration of kinetic Monte Carlo simulations through the modification of rate constants.* Journal of Chemical Physics, 2010. **132**(19).
85. Danielson, T., et al., *SQERTSS: Dynamic rank based throttling of transition probabilities in kinetic Monte Carlo simulations.* Computer Physics Communications, 2017. **219**: p. 149-163.
86. Dybeck, E.C., C.P. Plaisance, and M. Neurock, *Generalized Temporal Acceleration Scheme for Kinetic Monte Carlo Simulations of Surface Catalytic Processes by Scaling the Rates of Fast Reactions.* Journal of Chemical Theory and Computation, 2017. **13**(4): p. 1525-1538.
87. Schulze, T.P., *Kinetic Monte Carlo simulations with minimal searching.* Physical Review E, 2002. **65**(3).
88. Ravipati, S., et al., *A Caching Scheme To Accelerate Kinetic Monte Carlo Simulations of Catalytic Reactions.* The Journal of Physical Chemistry A, 2020. **124**(35): p. 7140-7154.
89. Hess, F., *Efficient Implementation of Cluster Expansion Models in Surface Kinetic Monte Carlo Simulations with Lateral Interactions: Subtraction Schemes, Supersites, and the Supercluster Contraction.* Journal of Computational Chemistry, 2019. **40**(30): p. 2664-2676.
90. Ouyang, M., et al., *Directing reaction pathways via in situ control of active site geometries in PdAu single-atom alloy catalysts.* Nature Communications, 2021. **12**(1).
91. Nettesheim, S., et al., *Reaction-Diffusion Patterns in the Catalytic Co-oxidation on Pt(110): Front Propagation and Spiral Waves.* Journal of Chemical Physics, 1993. **98**(12): p. 9977-9985.
92. Shim, Y. and J.G. Amar, *Semirigorous synchronous sublattice algorithm for parallel kinetic Monte Carlo simulations of thin film growth.* Physical Review B, 2005. **71**(12).
93. Merrick, M. and K.A. Fichthorn, *Synchronous relaxation algorithm for parallel kinetic Monte Carlo simulations of thin film growth.* Physical Review E, 2007. **75**(1).
94. Martinez, E., et al., *Synchronous parallel kinetic Monte Carlo for continuum diffusion-reaction systems.* Journal of Computational Physics, 2008. **227**(8): p. 3804-3823.
95. Oppelstrup, T., et al. *Spock: Exact parallel kinetic Monte-Carlo on 1.5 million tasks*. in *Proceedings of the 2016 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. 2016.
96. Prigogin.I and R. Lefever, *Symmetry Breaking Instabilities in Dissipative Systems .2.* Journal of Chemical Physics, 1968. **48**(4): p. 1695-&.
97. Murray, J.D., *Mathematical Biology I. An Introduction*. 3rd ed. Vol. 17. 2002, New York: Springer.
98. Arkin, A., J. Ross, and H.H. McAdams, *Stochastic kinetic analysis of developmental pathway bifurcation in phage lambda-infected Escherichia coli cells.* Genetics, 1998. **149**(4): p. 1633-1648.
99. Blake, W.J., et al., *Noise in eukaryotic gene expression.* Nature, 2003. **422**(6932): p. 633-637.
100. Stamatakis, M. and N.V. Mantzaris, *Comparison of Deterministic and Stochastic Models of the lac Operon Genetic Network.* Biophysical Journal, 2009. **96**(3): p. 887-906.

101. Schlögl, F., *Chemical reaction models for non-equilibrium phase transitions.* Zeitschrift für Physik, 1972. **253**(2): p. 147-161.
102. Vellela, M. and H. Qian, *Stochastic dynamics and non-equilibrium thermodynamics of a bistable chemical system: the Schlogl model revisited.* Journal of the Royal Society Interface, 2009. **6**(39): p. 925-940.
103. Gillespie, D., *Markov Processes: An Introduction for Physical Scientists*. 1991: Academic Press.
104. Gillespie, D.T., *Approximate accelerated stochastic simulation of chemically reacting systems.* Journal of Chemical Physics, 2001. **115**(4): p. 1716-1733.
105. Cao, Y., D.T. Gillespie, and L.R. Petzold, *Efficient step size selection for the tau-leaping simulation method.* Journal of Chemical Physics, 2006. **124**(4).
106. Pettigrew, M.F. and H. Resat, *Multinomial tau-leaping method for stochastic kinetic simulations.* Journal of Chemical Physics, 2007. **126**(8).
107. Rathinam, M., et al., *Stiffness in stochastic chemically reacting systems: The implicit tau-leaping method.* Journal of Chemical Physics, 2003. **119**(24): p. 12784-12794.
108. Tian, T.H. and K. Burrage, *Binomial leap methods for simulating stochastic chemical kinetics.* Journal of Chemical Physics, 2004. **121**(21): p. 10356-10364.
109. Chatterjee, A., D.G. Vlachos, and M.A. Katsoulakis, *Binomial distribution based tau-leap accelerated stochastic simulation.* Journal of Chemical Physics, 2005. **122**(2).
110. Anderson, D.F., *Incorporating postleap checks in tau-leaping.* Journal of Chemical Physics, 2008. **128**(5).
111. Haseltine, E.L. and J.B. Rawlings, *Approximate simulation of coupled fast and slow reactions for stochastic chemical kinetics.* Journal of Chemical Physics, 2002. **117**(15): p. 6959-6969.
112. Salis, H. and Y. Kaznessis, *Accurate hybrid stochastic simulation of a system of coupled chemical or biochemical reactions.* Journal of Chemical Physics, 2005. **122**(5).
113. Cao, Y., D.T. Gillespie, and L.R. Petzold, *The slow-scale stochastic simulation algorithm.* Journal of Chemical Physics, 2005. **122**(1).
114. Cao, Y., D. Gillespie, and L. Petzold, *Multiscale stochastic simulation algorithm with stochastic partial equilibrium assumption for chemically reacting systems.* Journal of Computational Physics, 2005. **206**(2): p. 395-411.
115. Peles, S., B. Munsky, and M. Khammash, *Reduction and solution of the chemical master equation using time scale separation and finite state projection.* Journal of Chemical Physics, 2006. **125**(20).
116. Mastny, E.A., E.L. Haseltine, and J.B. Rawlings, *Two classes of quasi-steady-state model reductions for stochastic kinetics.* Journal of Chemical Physics, 2007. **127**(9).
117. Rao, C.V. and A.P. Arkin, *Stochastic chemical kinetics and the quasi-steady-state assumption: Application to the Gillespie algorithm.* Journal of Chemical Physics, 2003. **118**(11): p. 4999-5010.
118. Samant, A., B.A. Ogunnaike, and D.G. Vlachos, *A hybrid multiscale Monte Carlo algorithm (HyMSMC) to cope with disparity in time scales and species populations in intracellular networks.* BMC Bioinformatics, 2007. **8**.
119. Samant, A. and D.G. Vlachos, *Overcoming stiffness in stochastic simulation stemming from partial equilibrium: A multiscale Monte Carlo algorithm.* Journal of Chemical Physics, 2005. **123**(14).
120. Rathinam, M., P.W. Sheppard, and M. Khammash, *Efficient computation of parameter sensitivities of discrete stochastic chemical reaction networks.* Journal of Chemical Physics, 2010. **132**(3).
121. Glasserman, P. and D.D. Yao, *Some Guidelines and Guarantees for Common Random Numbers.* Management Science, 1992. **38**(6): p. 884-908.

122. Stamatakis, M. and N.V. Mantzaris, *Intrinsic noise and division cycle effects on an abstract biological oscillator.* Chaos, 2010. **20**(3).
123. McAdams, H.H. and A. Arkin, *Stochastic mechanisms in gene expression.* Proceedings of the National Academy of Sciences, 1997. **94**(3): p. 814-819.
124. Tsimring, L.S., *Noise in biology.* Reports on Progress in Physics, 2014. **77**(2).
125. Trygubenko, S.A. and D.J. Wales, *Graph transformation method for calculating waiting times in Markov chains.* Journal of Chemical Physics, 2006. **124**(23).
126. Wales, D.J., *Calculating rate constants and committor probabilities for transition networks by graph transformation.* Journal of Chemical Physics, 2009. **130**(20).
127. Stevenson, J.D. and D.J. Wales, *Communication: Analysing kinetic transition networks for rare events.* Journal of Chemical Physics, 2014. **141**(4).
128. Sharpe, D.J. and D.J. Wales, *Numerical analysis of first-passage processes in finite Markov chains exhibiting metastability.* Physical Review E, 2021. **104**(1).
129. Sharpe, D.J. and D.J. Wales, *Graph transformation and shortest paths algorithms for finite Markov chains.* Physical Review E, 2021. **103**(6).
130. Sharpe, D.J. and D.J. Wales, *Nearly reducible finite Markov chains: Theory and algorithms.* Journal of Chemical Physics, 2021. **155**(14).
131. Athenes, M. and V.V. Bulatov, *Path Factorization Approach to Stochastic Simulations.* Physical Review Letters, 2014. **113**(23).
132. Andersen, M., C.P. Plaisance, and K. Reuter, *Assessment of mean-field microkinetic models for CO methanation on stepped metal surfaces using accelerated kinetic Monte Carlo.* Journal of Chemical Physics, 2017. **147**(15).
133. Rosta, E. and G. Hummer, *Error and efficiency of simulated tempering simulations.* Journal of Chemical Physics, 2010. **132**(3).