eCSE Technical Report for ARCHER2-eCSE01-13

Zacros Software Package Development: Towards Petascale Kinetic Monte Carlo Simulations with the Time-Warp Algorithm

Giannis D. Savva,[ab] Raz L. Benson,[a] Ilektra A. Christidi,[c] David Stansby,[c] and Michail Stamatakis*[a]

[a] *Department of Chemical Engineering, University College London, Torrington place, London, WC1E 7JE, United Kingdom.*

[b] *Theory and Simulation of Materials (THEOS), École Polytechnique Fédérale de Lausanne, 1015 Lausanne, Switzerland.*

[c] *Research Software Development Group, Advanced Research Computing Centre, University College London, Gower Street, London, WC1E 6BT, United Kingdom.*

* *Corresponding author; e-mail: m.stamatakis@ucl.ac.uk*

Abstract

Software *Zacros* employs a graph-theoretical approach to kinetic Monte Carlo (KMC) simulation of catalytic systems, coupled with the Time-Warp algorithm for parallel discrete event simulation. In this combined approach, the lattice, representing the catalytic surface, is divided into subdomains, each assigned to a single MPI process. Each MPI process simulates a trajectory asynchronously from the other MPI processes, and this inevitably leads to conflicts during the simulation when reactive events occur at subdomain boundaries. These conflicts are resolved in an exact way in the Time-Warp algorithm by saving snapshots of the KMC state, which enables roll-backs and re-simulations until all conflicts are corrected. This eCSE project focused on benchmarking and improving the Time-Warp implementation in *Zacros*, thereby delivering insight into how to optimise the tuneable parameters of the Time-Warp algorithm, as well as reducing the memory footprint of the implementation.

Keywords: ARCHER2; eCSE; Computational catalysis; Time-Warp algorithm; Discrete-event simulation; multiscale modelling.

Background

Kinetic Monte-Carlo (KMC) simulations have been instrumental in multiscale catalysis studies, enabling the elucidation of the complex dynamics of heterogeneous catalysts and the prediction of macroscopic performance metrics, such as activity and selectivity (1). However, the accessible length- and time-scales have been a limiting factor in such simulations. For instance, handling lattices containing millions of sites with "traditional" sequential KMC implementations is prohibitive due to large memory requirements and long simulation times. We have recently established an approach for exact, distributed, lattice-based simulations of catalytic kinetics, which couples the Time-Warp algorithm with the Graph-Theoretical KMC framework in the *Zacros* software package, enabling the handling of complex events within large lattices (2). In this eCSE project, the performance of this implementation was benchmarked as a function of algorithm parameters, its strong and weak scaling studied, and its memory footprint improved. As a result, the algorithm was found to scale well for all systems studied, for large enough lattices and number of processes. A methodology was developed to investigate the optimal algorithm parameters, that could improve the performance of the algorithm by at least an order of magnitude, depending on the chemical system simulated and HPC cluster that the simulation is running on. Finally, the memory usage and footprint of the algorithm has been improved and is now tuneable by the user.

The Algorithm

In order for *Zacros* to simulate chemical processes in parallel, the catalytic surface is decomposed into smaller subdomains and the simulation of each subdomain's history is assigned to a processor. Since the individual histories are simulated concurrently, events happening at the shared boundaries between domains can lead to causality violations. These violations are corrected by means of the Time-Warp algorithm (3), an optimistic approach to Parallel Discrete Event Simulation, which corrects the histories simulated by the "collaborating" processors up to the point that the final simulation history reproduces the exact dynamics of the underlying model.

In our MPI implementation of Time-Warp in *Zacros* (2), we use point-to-point communication between processors handling neighbouring subdomains, to send asynchronous messages of events that occurred in one subdomain but affect the simulation history of its neighbour. Since the local simulation time progresses independently in each process, it is possible that such a sent message may correspond to the past of its recipient process (thus the causality violation). To resolve this, each process keeps a history of the local system state at regular time intervals, so that a given state of the past can be restored in the event of a received message with a timestamp in the past of the local process. Such a "rollback" event may also require that past messages already sent by the process to its neighbours be undone, which is achieved by the process sending "anti-messages" to its affected neighbours. All the processes collectively communicate in regular intervals in order to establish the overall system simulation time, which is (roughly) the local time of the slowest process. This information is used for two purposes: to remove elements from state and message queues that are no longer needed due to being too far in the past, and thus free up memory for more elements to be stored; and to decide when the simulation is finished due to all the processes having reached their end time and handled all the messages that have been sent to each other.

Scaling Studies

Three chemical systems were used for benchmarking the parallel implementation of *Zacros* (2). System 1 included only adsorption, desorption, as well as a diffusion, which is a two-site event and leads to coupling between parallel domains since a single event can cross domain boundaries. System 2 included adsorption, desorption, and nearest-neighbour pairwise lateral interactions, which lead to inter-domain coupling since the corresponding energetic cluster/pattern can cross domains. System 3 included several elementary reactions of a complex CO oxidation mechanism as well as a detailed model of energetic interactions (4), which require large halos and lead to strong coupling between domains.

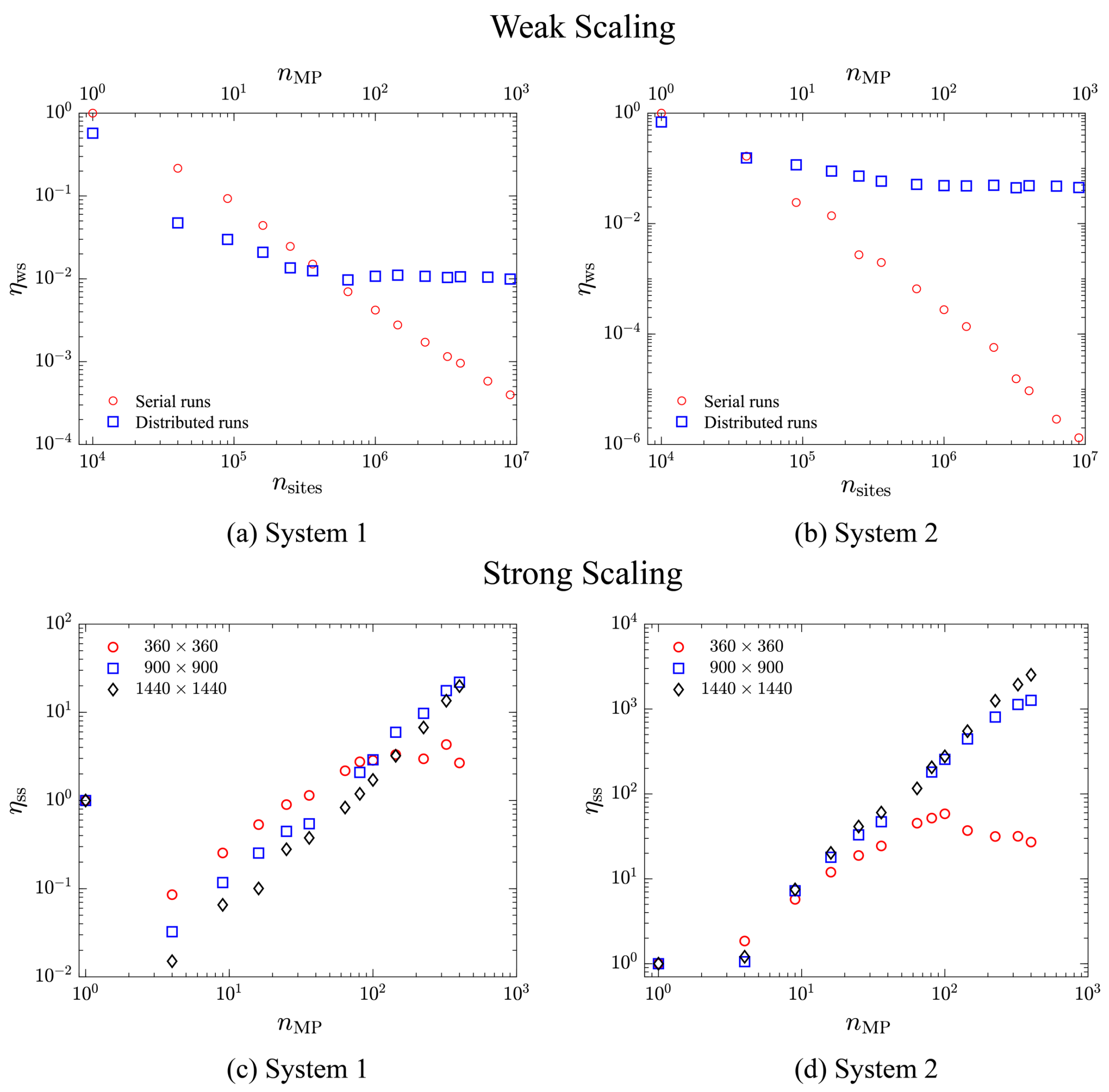

Figure 1: Weak (top) and strong (bottom) scaling plots for two different systems simulated with the Time-Warp algorithm in Zacros (2).

The results of these benchmarks appear in Figure 1. For the calculation of efficiency, we define the scaled time (t^*) as the ratio between KMC time versus real time. The higher this ratio, the faster the progression of the simulation. For the weak scaling benchmarks, we thus define the efficiency (η_{ws}) as the ratio between the scaled time of a simulation with n_{sites} distributed over n_{MP} MPI processes, versus the scaled time of a serial simulation on a lattice of n_{sites}/n_{MP} sites (which is the minimum number of sites), i.e.:

$$\eta_{ws} = \frac{t^*\left(n_{sites} : n_{MP}\right)}{t^*\left(n_{sites}^{\min}\right)}$$

In Figure 1(a) and (b) we plot the weak scaling efficiency for the parallel as well as the serial runs. Clearly for the latter the efficiency follows a scaling of approximately $1/n_{sites}$, because the computational effort for reaching a fixed KMC time, scales with the number of sites. Some additional inefficiencies are seen in System 2 for large lattice sizes. We observe that the efficiency of the parallel runs plateaus after some drop at small lattices, which can be attributed to the increasing use of communications and handling of more messages, as well as the treatment of causality violations via rollbacks. We also see that a "truly serial" run is slightly more efficient than a "distributed" run with only one MPI process, due to the MPI-related procedures that are still invoked in the latter run. In particular, the single MPI process, still checks at every KMC iteration whether an event involves halo sites (even though there are none in this case); in addition, MPI subroutines that e.g., probe for messages (even though there wouldn't be any), or broadcast the local virtual time (LVT) (to just one MPI process) are still called as usual. Overall, we see encouraging speedups, e.g., for system 2, the distributed runs being more than four orders of magnitude faster than serial ones.

For the strong scaling benchmarks, we define the efficiency (η_{ss}) as the ratio between the scaled time of a simulation with n_{sites} distributed over n_{MP} MPI processes, versus the scaled time of a serial simulation on the same number of sites, i.e.:

$$\eta_{ss} = \frac{t^*\left(n_{sites} : n_{MP}\right)}{t^*\left(n_{sites}\right)}$$

The results of these benchmarks for systems 1 and 2, are in Figure 1(c) and (d). For runs with more than just one MPI process, we generally observe an improvement in performance as more MPI

processes are employed in our simulations. The number of MPI processes for which a distributed run starts to outperform the single-MPI-process run depends on the system and the lattice size. For example, distributed runs with more than 100 MPI processes outperform single-MPI-process run for all lattices of system 1, while for system 2, runs with 9 MPI processes already outperform

the single-MPI-process runs quite significantly. We also observe that the efficiency plateaus for the smallest lattice considered (360×360), due to the increasing portion of halo sites. Indeed, as the number of MPI processes increases, the overall length of boundaries also increases, and so does the ratio between the number of sites that belong in halos over the total number of sites. In turn, the probability that an event will involve at least one halo site becomes higher, leading to inefficiencies due to causality violations and the ensuing roll-backs.

Finally, in Figure 2, we present results on the more realistic and more complicated System 3 (developed in Ref. (4)), for CO oxidation on Pd(111). We observe a steady rise in efficiency with an increase in the number of MPI processes, though the speedup is sublinear. For the largest simulation performed, which employed 27×27 = 729 MPI processes, we obtained a speedup by a factor of about 110. This behaviour is attributed to the large halo width of this system, which results in frequent causality violations that must be resolved via rollbacks.

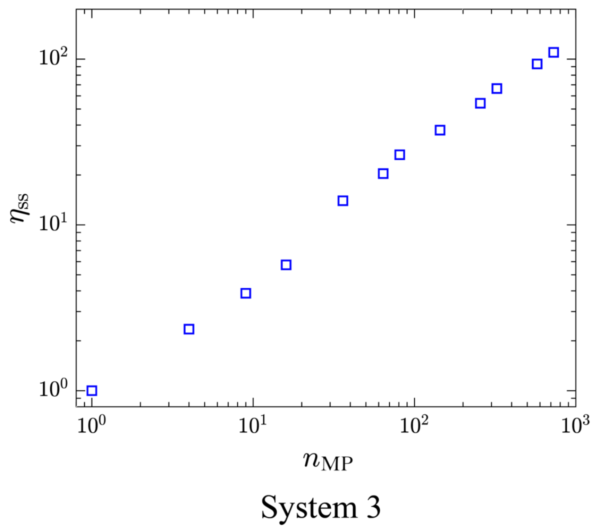

Figure 2: Results of strong-scaling benchmarks for System 3, i.e. the detailed CO oxidation model of Ref. (4), on a 2592×2592 lattice (more than 13.4 million sites) (2).

Parametric Studies

The performance of our parallel KMC implementation broadly depends on the amount of available memory, the type of data-structure used for storing and accessing these snapshots, and tuneable parameters that control the frequency of snapshot taking and global communications. Therefore, the performance as a function of these parameters was extensively studied on the benchmark systems introduced above. The snapshots of the system saved to each KMC state queue can occupy large amounts of memory and may need to be accessed frequently, so it is pertinent to consider the most appropriate data structure for this purpose. Two data structures were initially implemented in *Zacros*: the "linked list" and the "vector". In the former, the nodes (KMC-state objects) are not necessarily stored contiguously in memory, rather each node points/links to the next in the sequence. This means that memory can be allocated and deallocated as needed each time a snapshot is saved or deleted. In contrast, the "vector" has a fixed number of slots in a one-dimensional array of type KMC state, plus an additional one-dimensional array for indexing purposes. The memory thus needs to be allocated once and for all at the start of the simulation.

Whichever data structure is chosen, to avoid exhausting the memory available, it is also important to have a robust protocol by which MPI processes can delete any snapshots that are no longer needed. This leads naturally to the concept of global virtual time (GVT), t_{glob}, which is defined as the minimum among all the KMC times and time-stamps of buffered messages (i.e., those sent but not yet acted upon) across all MPI processes (3). On each MPI process, the earliest KMC state that could need to be reinstated to restore causality is the last one saved with a time-stamp $t_{state} = t_{GVT-\mathrm{state}} < t_{glob}$. All those with $t_{state} < t_{GVT-state}$ are obsolete and can be safely deleted. Likewise, any obsolete messages may be deleted from the message queue. In practice, t_{glob} is calculated by means of a global communication event at regular clock-time intervals of $\Delta\tau_{GVT}$. Knowledge of t_{glob} is also used in deciding when to terminate the simulation.

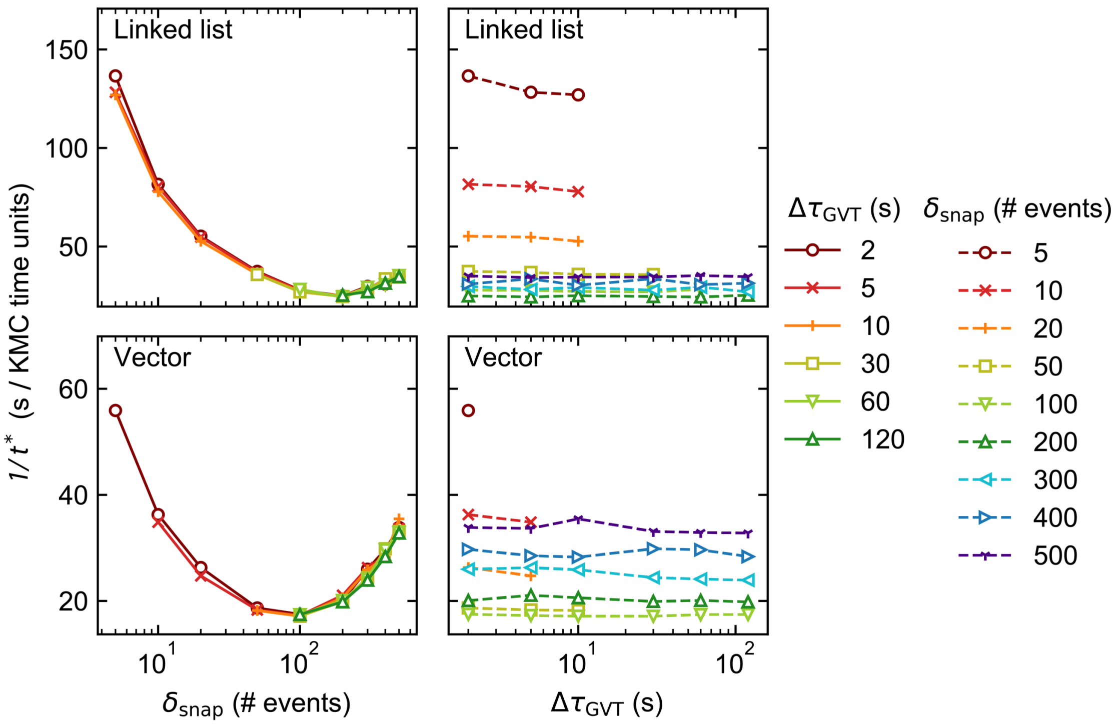

Figure 3: Results of the parametric studies of performance of System 1 with lattice size 1200×1200 distributed over 144 processors (5). The inverse of the KMC-time advancement per unit of clock time, 1/t, is plotted against the state saving interval, δ_{snap} (left) and GVT computation interval, $\Delta\tau_{GVT}$ (right). Lower values of 1/t* indicate higher efficiency.*

In Figure 3, we plot the computational time needed for 1 unit of KMC-time advancement (this is equivalent to $1/t^*$) for System 1 against δ_{snap} (left) and $\Delta\tau_{GVT}$ (right), for simulations on a 1200 × 1200 lattice. Note that the left-hand and right-hand plots for each KMC state queue data structure contain the same data, only presented differently. Regarding the effect of the tuneable parameters, our first observation is that faster KMC-time advancement is achieved with the vector data structure than with the linked list, which can be attributed to the additional time spent allocating and deallocating memory to KMC state queue when the linked list is employed. In contrast, the size of the vector structure is fixed, and all its needed memory allocated only once, at the beginning of the simulation. Unsurprisingly, the performance of each KMC simulation is seen to depend strongly on δ_{snap}. Naively, one may expect a monotonic improvement in performance as δ_{snap} is reduced, since this reduces the total amount of time spent in rollback propagation. Yet, the performance is observed to improve only up to a point, upon reducing δ_{snap}. In fact, we observe optimum performance (i.e., minimum $1/t^*$) around $\delta_{snap} = 100$ when using the vector state queue data structure, and slightly higher ($\delta_{snap} = 200$) for the linked list. The sharp rise in $1/t^*$ for smaller values of δ_{snap} is attributed to the additional time spent saving and deleting snapshots, which constitute the simulation bottleneck in this regime.

On the other hand, the choice of $\Delta\tau_{GVT}$ hardly affects the overall performance, indicating that the global communication overhead is negligible. That said, one should refrain from choosing very small values of $\Delta\tau_{GVT}$ lest the simulation output files occupy vast quantities of disk space. One must also ensure that, for a given choice of δ_{snap}, $\Delta\tau_{GVT}$ is sufficiently small such that obsolete snapshots are deleted before the memory allocated to KMC state queue is filled up. This is exemplified by the several "missing" data points in Figure 3, e.g., all points for which $\delta_{snap} = 5$, $\Delta\tau_{GVT} > 10$ ($\Delta\tau_{GVT} > 2$) are absent with the linked list (vector) KMC state queue structure. A data point is omitted wherever the KMC state queue in at least one MPI process became too large to fit in the available memory before the allocated 1 hour of clock time had passed. It is important to stress that the missing data points just described do not imply failed simulations. This is because, when memory does fill up, *Zacros* is configured to "sparsify" the state queue by deleting every second snapshot. The frequency with which future KMC states are saved is correspondingly reduced by doubling δ_{snap}. This sparsification procedure can occur, in principle, arbitrarily many times on each MPI process, such that a poorly chosen input (initial) value for δ_{snap} will not result in simulation failure, but it will adversely affect performance. In Systems 1 and 2, we found that sparsification tended to occur either permanently throughout most of the MPI processes, or not at all. This behaviour can be attributed to the spatial homogeneity of the dynamics, with the upshot that $1/t^*$ for such simulations is not truly reflective of the input δ_{snap} value since the latter changes during the run. Thus, we opted to omit the results of any simulations during which sparsification of the KMC state queue occurred.

Proof of Concept for Large Systems

In order to study the stability and behaviour of the implementation for large number of processes and long simulation times, a large chemical oscillator system (Brusselator) that exhibits large-scale pattern formation was simulated (Figure 4). The Brusselator reaction mechanism was introduced by Prigogine and Lefever in the late 60's to study symmetry-breaking instabilities in dissipative systems (6). It involves two main species, one of which promotes its own production in an autocatalytic manner. This results in rich dynamic behaviour, specifically oscillations, under certain parametric constraints and assuming fast diffusion (well-mixed system). Furthermore, if the reaction is embedded into a spatially extended medium into which molecular species can diffuse (in addition to reacting), an instability occurs when diffusion is slow compared to reaction, and as a result, the system can exhibit spatiotemporal pattern formation. Such a system was therefore deemed ideal for demonstrating the capabilities of our distributed KMC approach by reproducing such spatiotemporal patterns at large scales (7).

The Brusselator system was thus simulated in a lattice with 4000×4000 sites, i.e., 16 million sites and the simulation was distributed over 25×25=625 PUs, so that each one of them is assigned a subdomain of 160×160 sites (Figure 4). The entire simulation was run on Thomas, the UK National Tier 2 High Performance Computing (HPC) Hub in Materials and Molecular Modelling. Because of the wall time restrictions, this simulation was broken into "chunks" of 24 or 48 hours each. *Zacros*'s core implementation provides a functionality for stopping and resuming a simulation, and consequently, runs that use this checkpointing feature produce identical results with continuous runs, while being more robust against system faults. The simulation reached an overall KMC time of about 620 KMC seconds and involved more than 1.6 trillion elementary events (though, due to the rollbacks of the Time-Warp algorithm, the actual times and number of events executed by the PUs were larger, as will be discussed later). In terms of real time, the distributed simulation was running for 38 days.

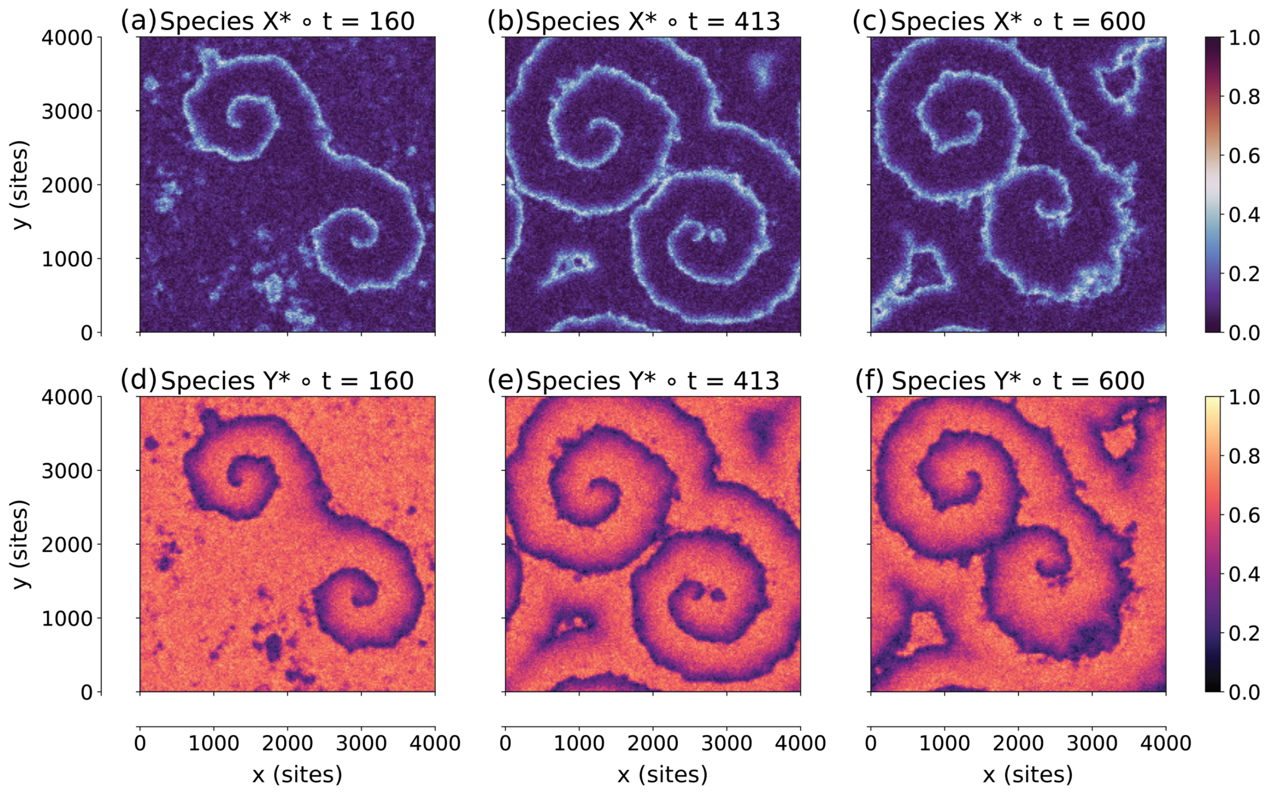

Figure 4: Snapshots of the fractional coverages of the activator species X (panels a, b, c) and the inhibitor species Y* (panels d, e, f) at various times (in units of s) during the simulation. Two spirals are reproduced rotating in opposite directions (7). At time 413 s, a secondary wavefront emerges close to the tip of the lower spiral, and eventually pushes the tip closer to the centre of the domain.*

Memory Usage Improvements

Early profiling revealed that inter-process communication is not a performance bottleneck for our implementation. Therefore, the effort was concentrated in improving the single-core performance via improving the algorithm's memory footprint. To this end, a memory amortisation scheme was implemented, by which the main data-structures of the KMC state are allocated conservatively, i.e.

with a small size, at start-up and their sizes are increased on an "as needed" basis. For instance, the data-structure that holds the list of elementary reaction events that can happen on the lattice, given an adsorbate configuration, could be initialised with a number of elements equal the number of lattice sites. Then, if the number of elementary events exceeded the size of the datastructure, the size of the latter would be doubled to store the additional elementary events. Such size increases are allowed to happen as many times as necessary throughout the simulation.

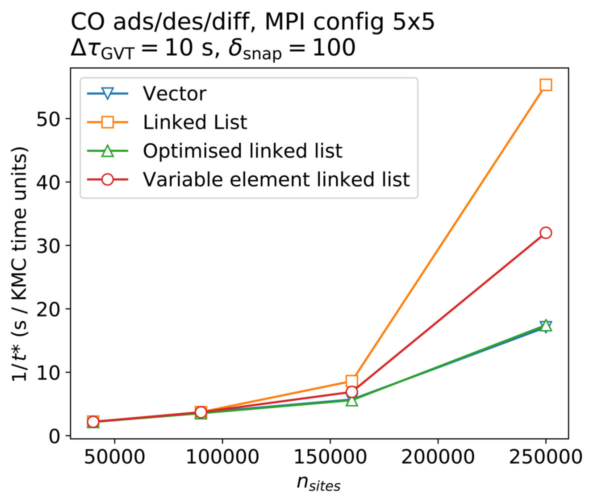

Figure 5: Performance benchmarks of the four KMC state queues, for System 1 (adsorption, desorption, diffusion) on lattices of different sizes distributed over 25 MPI processes. A weak scaling approach was adopted, i.e., the 200×200 lattice is run with 2×2 MPI processes, the 300×300 lattice with 3×3 MPI processes, etc. The clock time per KMC-time advancement, 1/t, is plotted against the number of sites for different KMC state queue data-structures. Lower values of 1/t* indicate higher efficiency.*

To improve the performance of the KMC state queues (which are integral parts of the roll-back machinery of Time-Warp as discussed earlier), but to also accommodate variable-sized KMC states we developed two novel KMC state queue data-structures. The first one, referred to as optimised linked list, avoids the continuous allocation and deallocation of slots holding KMC states, but works with fixed sizes of KMC states. Thus, instead of deallocating KMC state slots upon clean-up (after a global communication event), this data-structure simply moves them to the end of the queue to be re-used later by copying a new KMC state onto an obsolete one. The second state queue data-structure, referred to as variable-element linked list, is also an optimised linked list but with the functionality of storing KMC states of variable size. In this case, allocation of new KMC state slots is done only if the size of the KMC state increases, due to the invoking memory amortisation procedures. Obsolete KMC state slots are only deleted if their size is smaller than that of the current KMC state.

Performance benchmarks of all four KMC state queues is shown in Figure 5. As expected, the optimised linked lists perform similarly to the vector state queue, since, when all the necessary slots have been allocated, the queue becomes "static", no longer performing memory allocation or deallocation. The "original" linked list is the least efficient due to the continuous memory allocation and deallocation, while the variable-element linked list exhibits and intermediate level of efficiency.

Acknowledgements

This work was funded under the embedded CSE programme of the ARCHER2 UK National Supercomputing Service (https://www.archer2.ac.uk/).

References

1. Pineda M, Stamatakis M. Kinetic Monte Carlo simulations for heterogeneous catalysis: Fundamentals, current status, and challenges. J Chem Phys. 2022;156(12):120902.

2. Ravipati S, Savva GD, Christidi I-A, Guichard R, Nielsen J, Réocreux R, Stamatakis M. Coupling the Time-Warp algorithm with the Graph-Theoretical Kinetic Monte Carlo framework for distributed simulations of heterogeneous catalysts. Comput Phys Commun. 2022;270:108148.
3. Jefferson DR. Virtual Time. ACM Transactions on Programming Languages and Systems. 1985;7(3):404-25.
4. Piccinin S, Stamatakis M. Steady-State CO Oxidation on Pd(111): First-Principles Kinetic Monte Carlo Simulations and Microkinetic Analysis. Top Catal. 2017;60(1-2):141-51.
5. Savva, G. D., Benson, R. L., Christidi, I.-A. and M. Stamatakis (2023). "Large-scale benchmarks of the Time-Warp/Graph-Theoretical Kinetic Monte Carlo approach for distributed on-lattice simulations of catalytic kinetics". *Physical Chemistry Chemical Physics*, 25: 5468-5478. (doi: 10.1039/D2CP04424B).
6. Prigogine I, Lefever R. Symmetry Breaking Instabilities in Dissipative Systems. II. J Chem Phys. 1968;48(4):1695-700.
7. Savva, G. D., Benson, R. L., Christidi, I.-A. and M. Stamatakis (2023). "Exact Distributed Kinetic Monte Carlo Simulations for On-Lattice Chemical Kinetics: Lessons Learnt from Medium- and Large-Scale Benchmarks". Philosophical Transactions of the Royal Society A, Accepted. (doi: 10.1098/rsta.2022.0235).