# Scalability strategies for automated reaction mechanism generation

Agnes Jocher[a], Nick M. Vandewiele[a], Kehang Han[a], Mengjie Liu[a], Connie W. Gao[a], Ryan J. Gillis[a], and William H. Green[a]

a. Department of Chemical Engineering, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, MA 02139, USA

ajocher@mit.edu, nickvandewiele@gmail.com, kehanghan@gmail.com, mjliu@mit.edu, connie.w.gao@gmail.com, rgillis@mit.edu, whgreen@mit.edu

## Abstract

Detailed modeling of complex chemical processes, like pollutant formation during combustion events, remains challenging and often intractable due to tedious and error-prone manual mechanism generation strategies. Automated mechanism generation methods seek to solve these problems but are held back by prohibitive computational costs associated with generating larger reaction mechanisms. Consequently, automated mechanism generation software such as the Reaction Mechanism Generator (RMG) must find novel ways to explore reaction spaces and thus understand the complex systems that have resisted other analysis techniques. In this contribution, we propose three scalability strategies — code optimization, algorithm heuristics, and parallel computing —

that are shown to considerably improve RMG's performance as measured by mechanism generation time for three representative simulations (oxidation, pyrolysis, and combustion). The improvements create new opportunities for the detailed modeling of diverse real-world processes.

# 1 Introduction

Reaction mechanisms are valuable engineering tools that provide insight into the evolution and fate of chemically reactive systems. Coupled with or integrated into multiscale-multiphysics models, reaction mechanisms can help elucidate physical phenomena that are driven by chemical kinetics. Ultimately, this insight can be used to design and optimize a wide range of processes and materials. Traditionally, reaction mechanisms are generated manually, but in the past decade, the computer-assisted development of reaction mechanisms has emerged as a promising tool leveraging the combined power of theoretical insight and extensive experimental data (Burke, 2016). Several software packages are available that harness the power of automation in the field of chemical kinetics (Vandewiele et al., 2012; Broadbelt et al., 1996; Van de Vijver et al., 2015; Rangarajan et al., 2012; Warth et al., 2000; Blurock, 1995). In the following work, we focus on the Reaction Mechanism Generator (RMG) project, which provides a framework for the automated, computer-assisted development of reaction mechanisms, with the goal of producing high-fidelity, predictive, and reproducible kinetic models (Green, 2007).

Several recent advancements in kinetic modeling (Vandewiele et al., 2015; Allen et al., 2014; Gao et al., 2015; Prozument et al., 2014; Carr et al., 2015; Seyedzadeh and West,

2016; Class et al., 2016) and novel chemical kinetic applications (Jalan et al., 2010, 2013; Suleimanov and Green, 2015) are posing new challenges to RMG. For instance, to improve fidelity, single-component fuel surrogates are being replaced by more realistic multicomponent formulations (Narayanaswamy et al., 2016). This enhances understanding of the impact of existing and novel feedstock blends on the overall process, but greatly increases the complexity of the modeling, challenging automated strategies. Other advancements in kinetic modeling aim to reduce the uncertainties in model parameters. For instance, the Quantum Mechanics Thermodynamic Property (QMTP) module for estimating thermodynamic properties using on-the-fly quantum chemistry calculations (Magoon and Green, 2013) complements the existing Benson group additivity methods (Benson, 1968) that suffered from limited accuracy for some classes of molecules, including polycyclic species. Quantum chemistry has been used to improve the thermochemical parameters in a kinetic model, at the cost of significantly increasing the CPU time required (Magoon and Green, 2013). Methods for calculating pressure-dependent rate coefficients improve estimates for rate coefficients of pressure-dependent reactions (Allen et al., 2012). In addition, capabilities such as sensitivity analysis (Gao et al., 2016) and uncertainty quantification (Gao et al., 2019) are being added to RMG. All the additions may facilitate the construction of high-fidelity kinetic models, but they also increase computational demands. Consequently, without addressing the scalability of the algorithms, many of the current and upcoming features will be available in theory, but unaffordable in practice. Although considerable effort has been spent in developing robust and scalable numerical schemes for solving the sets of equations associated with specific chemical processes, such as combustion (Shi et al., 2011), there has been little focus on

the scalability of computer-assisted reaction mechanism generation. This shortcoming is addressed in the present contribution. We begin by assessing the computational complexity of RMG simulations and identifying the most time-consuming bottlenecks for three simulations selected to represent the diversity of computational challenges faced by automatic mechanism generation. Barriers that inhibit scalability and performance are discussed and guide the proposed changes to code optimization, algorithm heuristics, and parallel computing. These enhancements are then explained and analyzed in detail. Finally, the strategies are evaluated with respect to walltime and memory consumption in the three RMG test simulations.

# 2 Automated reaction mechanism generation

## 2.1 RMG - Overall structure and key components

This section outlines the overall structure and the key components that govern the computational performance of RMG simulations. Details regarding the chemical meaningfulness and accuracy of the employed methods have been published previously (Gao et al., 2016; Hansen et al., 2013; Zhang et al., 2018; Gudiyella et al., 2018). RMG uses the rate-based algorithm of Susnow et al. (1997) to iteratively grow (“enlarge”) the reaction mechanism one or more species at a time. The starting point in this enlargement scheme is a user-defined set of species that corresponds to the reactants of the modeled chemical process and acts as the initial species of the reaction mechanism to be enlarged. An iterative procedure generates all reactions between the species of the current mechanism, called the “core”. These reactions are synchronized to avoid double occurrences and the resulting product species that are not yet part of the core are added

to the “edge”. The rate-based enlargement algorithm dictates that an edge species is moved to the core when the flux $R_i$ to the edge species $i$ exceeds a certain threshold value, Eq. 1.

$$R_i = \frac{dC_i}{dt} > \epsilon \cdot R_{char} \quad \textbf{Eq. 1}$$

with $\varepsilon$ being the user-specified error tolerance and $R_{char}$ the characteristic flux of the system, defined by Eq. 2

$$R_{char} = \sqrt{\sum_j R_j^2} \quad \textbf{Eq. 2}$$

for all core species $j$. An isothermal, isobaric batch reactor model is used to evaluate species fluxes and species concentrations. A system of ordinary differential equations (ODEs) containing only the reactions from the mechanism core is integrated in time until the rate of formation of a species in the edge exceeds the threshold value. Whenever an edge species is added to the core, the reactor is initialized back to time zero and species fluxes and concentrations are re-evaluated through integration. This iterative enlargement procedure is the main loop through which the mechanism is grown. The enlargement procedure of the mechanism is complete when none of the edge species fluxes meets the threshold value and the user-defined termination criteria, i.e., residence time for the simulation ($t>t_{term}$) or a specific conversion of a given initial species ($X>X_{term}$) is reached. The final model includes all the species and reactions in the core. To obtain all the information that is required to decide which species should be added to the core, the enlargement procedure passes through four key phases: reaction generation,

synchronization, chemical data computation, and finally, reaction system integration, see Fig. 1.

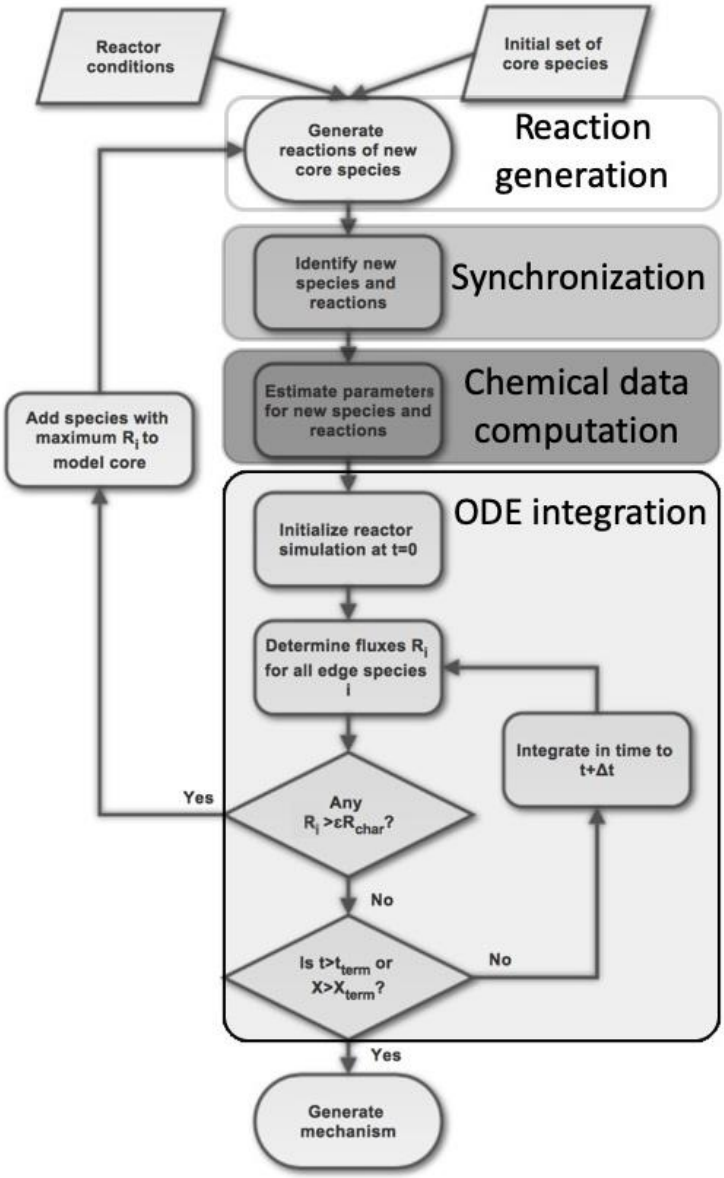


**Figure 1: The schematic outlines the workflow for automated reaction mechanism generation with one reactor for an isothermal, isobaric condition. The enlargement step that integrates a species from the edge into the core consists of four key components: reaction generation, synchronization, calculation of chemical data such as thermodynamic properties of species and finally solving the system of ODEs including the calculation of species concentrations and fluxes.**

### 2.1.1 Reaction generation

In the reaction generation phase, the species that was added in the previous enlargement iteration is used as a reactant to generate new reactions and species. For unimolecular reactions, this is done by iteratively matching the new reactant to the templates of a series of built-in unimolecular reaction families. For bimolecular reactions, the new species is paired with each species already present in the model core and then matched to the bimolecular templates. When a set of compatible reactants and template are found, products are created by applying the reaction family-specific recipe. This recipe precisely describes the transformation from reactants to products with instructions to break and form specific bonds, gain or lose electrons, etc. The reaction generation step results in the creation of a large number of, potentially redundant, new reactions and species which requires additional processing called synchronization.

### 2.1.2 Synchronization

During synchronization, RMG iterates over the list of newly generated reactions and species and determines if any are redundant. If a newly generated species is identical to a species in memory, RMG deletes the new structure and replaces it with a reference to the existing structure in memory. A similar mechanism is in place for reactions. RMG requires synchronization because tracking unique species and reactions allows distinction between the model's core and edge. This distinction is required for the iterative enlarging procedure and allows for the proper formulation of the system of ODEs to solve. Furthermore, synchronization avoids the redundant computation of chemical data. The uniqueness of species is established through isomorphism comparisons of the corresponding chemical graphs that represent the species. The uniqueness of reactions

is determined by comparing the reactants and products of the reactions and hence also relies on graph isomorphisms. If there are multiple transition states of comparable energy connecting these species then the rates should be summed together to give the overall flux from some set of reactants to some set of products.

### 2.1.3 Chemical data computation

After the synchronization phase has determined a list of new species and reactions, the chemical data computation phase calculates thermodynamic properties of the species or kinetic parameters of the reaction. Both thermodynamic property and kinetic parameter calculations rely heavily on performing subgraph isomorphism queries on a large number of substructures contained in RMG's databases. The uncertainty of the models due to the uncertainty of the thermodynamic and kinetic parameters can be estimated using recently added tools (Gao, 2016). Different models and observables have different degrees of sensitivity to these uncertainties precluding any blanket statements about uncertainty requirements.

### 2.1.4 Reaction system integration

In this final step, the state variables are re-computed, and a new species is selected and added to the core. The system of ODEs that corresponds to the new state of the reaction mechanism "core" is solved. The first edge species whose formation rate exceeds a threshold, see Eq. 1 and Eq. 2, is chosen as the next species to be added to the core. From here the cycle restarts, with the next reaction generation step based on the newly added species.

## 2.2 Performance bottlenecks

To assess performance bottlenecks of the automated reaction mechanism generation tool, three simulations are analyzed in depth. They are selected to represent a diverse collection of cases that make use of a large part of RMG's features and settings, Tab. 1.

**Table 1: Characteristics of three representative RMG simulations.**

| | | **Initial stage of diesel oxidation** | **Hexadiene pyrolysis** | **Rich natural gas combustion** |
|---|---|---|---|---|
| **Initial mixture** | | Six-component diesel surrogate, oxygen | Hexadiene, methane, hydrogen, nitrogen | Natural gas, air |
| **Isothermal and isobaric operating conditions** | | 500 K | 1350 K | 1400 K |
| | | 200 bar | 1 bar | 20 bar |
| **Thermodynamic property calculation method** | | Benson group additivity | QMTP | Benson group additivity |
| **Initial molar ratio** | | n-$C_{11}$ : n-$C_{13}$ : n-$C_{16}$ : n-$C_{19}$ : n-$C_{21}$ : n-decylbenzene : $O_2$ = 1.00 : 1.27 : 1.67 : 1.20 : 0.67 : 0.80 : 0.33 | 1,3-hexadiene : $CH_4$ : $H_2$ : $N_2$ = 1.00 : 152.00 : 23.00 :1288.00 | $CH_4$ : $C_2H_6$ : $C_3H_8$ : $O_2$ : $N_2$ = 1.00 : 0.09 : 0.02 : 0.71 : 0.33 |
| **Kinetics source** | | ▪ Training reactions<br>▪ RMG 'default' kinetics families<br>▪ Rate rules | ▪ Training reactions<br>▪ RMG 'default' kinetics families<br>▪ Rate rules | ▪ Training reactions<br>▪ RMG 'default' kinetics families<br>▪ Rate rules |
| **Final model size (Base case)** | **Core** | Species: 26<br>Reactions: 24 | Species: 131<br>Reactions: 4689 | Species: 376<br>Reactions: 17,739 |
| | **Edge** | Species: 13,429<br>Reactions: 41,355 | Species: 44,604<br>Reactions: 156,906 | Species: 679,127<br>Reactions: 1,691,134 |

The first case is a partial oxidation of a diesel surrogate, which consists of five larger linear alkanes and n-decylbenzene. The automatic mechanism generation of this system leads to a large number of generated reactions even in the early stages of the simulation. In the second case, a kinetic mechanism for the pyrolysis region of a 1,3-hexadiene doped methane flame is generated. This model could be employed to study soot formation (Sharma et al., 2007; Sharma et al., 2010, Vermeire et al., 2017). To address the need for accurate thermodynamic properties of cyclic and polycyclic species in these studies, the Quantum Mechanics Thermodynamic Property (QMTP) method is used rather than Benson group additivity. Case three generates a mechanism for natural gas combustion containing methane, ethane, and propane as the initial reactants. Due to the reactant molecules being small in size, only a small number of reactions per iteration is generated early in the simulation, contrasting case one. However, this simulation is run long enough that the generated mechanism grows comparatively large in size. The main purpose of the present work is to improve the scalability of our automated reaction generation tool. To test the improvements within a reasonable time frame, the initial stage of diesel oxidation and rich natural gas combustion simulations were evaluated after fourteen days of wall clock time. For the hexadiene pyrolysis case, that is dominated by the computation of thermodynamic properties with the QMTP method, results are evaluated for 130 core species.

Figure 2 compares the four key phases of mechanism enlargement and their contribution to the overall mechanism generation time for each of the three cases. These cases

will be called “base cases” in the remainder of the manuscript.

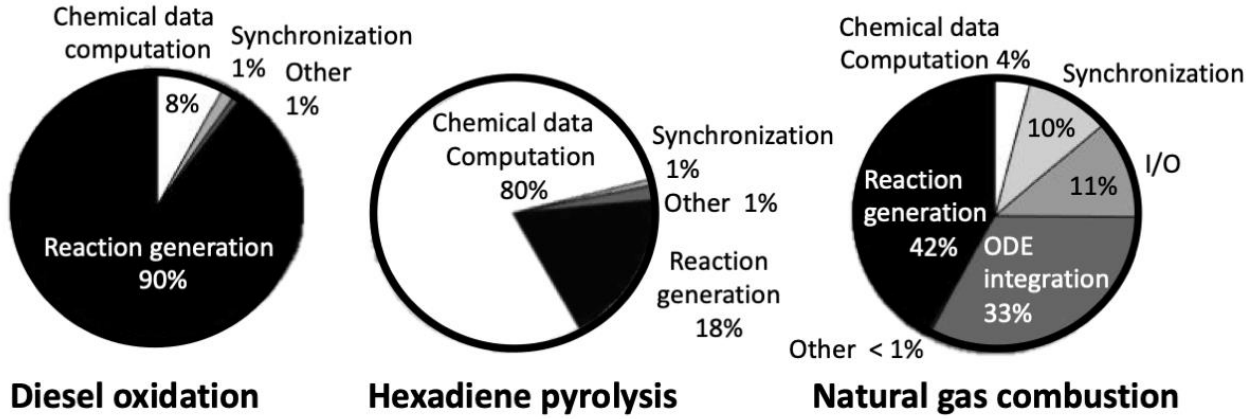


**Figure 2: Contribution of key enlargement phases to overall kinetic mechanism generation time.**

For the diesel oxidation case, which involves longer n-alkanes with multiple reacting sites, the reaction generation phase is by far the largest performance bottleneck. The synchronization step, the chemical data computation phase, and the ODE integration step have minor or negligible contributions to the overall simulation time. In case two, hexadiene pyrolysis, which employs the computationally expensive QMTP method for estimating thermodynamic properties, the chemical data computation phase becomes the limiting factor. Finally, as the RMG simulation advances to larger mechanism sizes in the natural gas combustion case, a significant amount of time is spent in both the reaction generation phase as well as the ODE integration phase. In this third case, the chemical data computation, synchronization and input/output (I/O) events play a minor role. These observations suggest that the computational load of RMG simulations cannot be isolated in a single routine, but rather, is dynamic in nature and heavily dependent on the chemical system under investigation. For example, while improvements to assignment, tree-structure search, and reaction path degeneracy calculations could accelerate mechanism generation in the diesel oxidation case, these strategies would have negligible impact in the hexadiene pyrolysis case. As a result, multi-faceted strategies that focus on

computational performance across the spectrum of simulations are required to significantly reduce the wall clock time spent generating the kinetic reaction mechanisms.

# 3 Scalability strategies

## 3.1 Challenges and tradeoffs

Scalability strategies for automated reaction mechanism generation face four main challenges: continuity, programming language, information-rich data structures, and data consistency.

### 3.1.1 Continuity

The RMG project has existed for almost two decades. Many incremental developments have grown RMG into a codebase containing about 125 400 lines of Python code and 13 900 lines of Cython code, including a separate text-based database with about 277 700 lines of Python code for chemical data storage. A single codebase simultaneously maintains a Django-powered website (http://rmg.mit.edu), serves as production code on high performance clusters, and supports desktop versions on Linux and Mac OSX. Furthermore, as a collaborative, open-source project with over 2000 commits added in the last year by more than twenty developers in distributed development teams, the capabilities of RMG are constantly expanding. However, each new update poses a risk to the accuracy and performance of the software. While more than one thousand unit and regression tests are embedded in the automated continuous integration system to catch blatant bugs, subtler changes resulting in structural or numerical deviations in the final converged mechanisms can arise from the synergistic effects of coupling multiple features. These errors are more difficult to discover. Therefore, any code modification

must strike a balance between performance and software reuse and robustness, which inevitably results in compromises to both.

### 3.1.2 Programming language

One such compromise is the choice of the programming language. RMG is implemented in Python, after a 2012 rewrite from the original version in Java (Song, 2004). Python has a rich ecosystem of scientific libraries that enable fast-paced development and ease of portability. RMG currently contains over twenty external dependencies ranging from low-level numerical libraries such as NumPy (van der Walt, 2011) and SciPy (Jones and Oliphant) to graph visualization toolkits such as GraphViz (Gansner and North, 1999). However, due to the interpreted, dynamically typed nature of Python, it bears significant performance penalties compared to other popular choices of programming languages for scientific software.

### 3.1.3 Information-rich data structures

The fundamental entities that are manipulated in RMG are “Species” and “Reaction” data structures. They are highly nested, object-oriented data structures that systematically categorize the layers of molecular information needed by the different parts of RMG. Moreover, the species and reaction data structures do not merely store the primitive variables required within a simulation, they also contain meta-variables for analysis, visualization and post-processing of the generated reaction mechanisms. For example, upon calculation of the thermodynamic properties of a species via the Benson group additivity method (Benson, 1968), the matched groups and origin of the groups will be stored as part of the Species data structure and can be verified *posteriori* to confirm their validity. The downside of using such high-level, information-rich data structures is that

their manipulation is difficult to translate into the arithmetic operations of mathematical data structures such as arrays of floating-point numbers. Instead, the types of operations that species and reactions undergo are complex and require accessing substantial out-of-cache data, which in turn leads to suboptimal performance on architectures designed for compute-bound mathematical problems.

#### 3.1.4 Data consistency

RMG extensively makes use of references to existing objects in memory. This strategy, inherent to many object-oriented software paradigms, not only prevents anomalies during the RMG simulation, but also reduces memory requirements and avoids redundant computation and duplication of associated data. The extensive use of object references is not only a result of the choice of programming model but is also inherent to the nature of automated reaction mechanism generation. Typically, for mechanisms generated by RMG consisting of about 100 species and 3000 reactions, every unique species will be created 300 times during each reaction generation phase before being called in synchronization. Further, each species appears in seventy-five distinct reactions on average. The extensive use of object references in lieu of newly created objects in memory undoubtedly leads to performance gains, it does, however, pose problems for parallel computing.

### 3.2 Proposed scalability strategies

Accounting for the iterative nature of RMG's mechanism generation approach, the identified performance bottlenecks, and the discussed challenges and tradeoffs for scalability strategies, we have focused on performance improvements within the individual phases of a mechanism enlargement loop. While post-processing features such

as periodic checkpointing and intermediate mechanism writing may significantly increase the computational cost, they can be deactivated and are therefore not further considered for performance optimization. Conventional strategies to reduce the computational load of simulations, such as those that are employed by computational fluid dynamic software, e.g., domain discretization and decomposition, don't apply to RMG. The strategies proposed in this work can be roughly categorized into code optimization, algorithm heuristics, and parallel computing.

### 3.2.1 Code optimization

RMG uses several code optimization techniques. To reduce the performance penalty imposed by Python, large parts of RMG that contain critical code, such as the VF2 (Foggia et al., 2001; Cordella et al., 2004) algorithm used for (sub-)graph isomorphism have been converted to Cython (Behnel et al., 2011). The Cython project compiles Python-style code to C through static typing, leading to an expected speed up of about an order of magnitude for numerically intensive code. Another technique involves prescreening to determine that molecule pairs are non-identical before calling the graph isomorphism algorithm. Currently, the prescreening uses the molecular formula of the molecule as a fingerprint and filters out about 80% of the molecule pairs. Finally, the integration of the ODE system is performed by DASPK (Li and Petzold, 1999), a Fortran-based differential algebraic system solver accessed through the PyDAS (PyDAS) Python interface.

### 3.2.2 Algorithm heuristics

To further target the reaction generation bottleneck, especially for simulations with large molecules, a "reaction filtering" approach is implemented. Reaction filtering reduces the number of reactions generated with each iteration trading off kinetic mechanism

completeness for speed-up by neglecting species with extremely low concentrations. Per the laws of mass action, the rates of reactions whose reactants have very low concentrations must be very low as well. Very slow reactions necessarily contribute little to the overall rate of formation of high flux, and thus important, species. If a very slow reaction contributes significantly to the overall rate of formation of a species, that species must be formed in only low concentrations and thus can be excluded from the mechanism. Despite the limited importance of slow reactions to the overall mechanism, the time spent in the reaction generation phase is independent of the kinetic (un)importance of a reaction. A heuristic approach is chosen that prevents species from undergoing reactions if the highest achievable reaction rate between those species is below a threshold. In each iteration, prior to the generation of reactions, the following inequality is verified:

$$k_{max} \cdot \prod_i c_i < \epsilon \cdot R_{char} \ , \qquad \textbf{Eq. 3}$$

with $k_{max}$ being an upper limit for the rate coefficient of a reaction between the reactants $i$, $c_i$ being the concentration of reactants evaluated at any given time in the reaction system, $\varepsilon$ being the model error tolerance and $R_{char}$ being the characteristic flux defined in Eq. 1~~Eq. 1~~. The left-hand side of Eq. 3 can be interpreted as the highest reaction rate possible for the examined reaction. The choice of $\varepsilon \cdot R_{char}$ allows user input on the tolerance through $\varepsilon$, while ensuring that the threshold follows the dynamics of the simulated reaction system through $R_{char}$. Values for the upper limit of $k_{max}$ are chosen based on the reaction order. For unimolecular reactions, $k_{max}$ is set to $(k_B \cdot T)/h$ , where $k_B$ is the Boltzmann constant, $T$ is the temperature and $h$ is Planck's constant. This maximum can be interpreted as the rate coefficient from the Eyring equation (Eyring, 1935) when the Gibbs free energy of activation is equal to zero. For bimolecular reactions,

$k_{max}$ is set to $10^8$ $m^3$ $mol^{-1}$ $s^{-1}$, a typical high-pressure-limit rate coefficient for radical recombination reactions where the radicals are polyatomic molecules. Since barrier-less radical recombination reactions are among the fastest bimolecular reactions, it is expected that this value for $k_{max}$ should not exclude important reactions. To ensure the chemical accuracy of a generated model the validity of the chosen $k_{max}$ has to be carefully studied. Here we just employ the reasonable default RMG values.

On a similar basis, another heuristic approach, called “species flux pruning” (Han et al., 2017), was implemented to reduce the memory requirements of RMG simulations. In this algorithm, normalized fluxes are used as a metric to identify negligible species during model generation to prune them and the reactions in which they participate. A potential risk of species flux pruning is the erroneous identification of important species as unimportant and their subsequent deletion, leading to a loss of model accuracy. However, careful selection of pruning parameters allows reduced memory requirements for building accurate kinetic models, without loss of accuracy. For example, for a model with 200 to 300 species the memory requirement can be reduced by about a factor of four, allowing creation of larger models that were unreachable using earlier versions of RMG due to limited hardware resources.

### 3.2.3 Parallel computing

For several years processor clock speeds have been relatively stable and are not expected to increase substantially in the near future (Millett and Fuller, 2011). However, new chip designs allow for the addition of more cores to a single processor to increase code performance. In addition, depending on the hardware manufacturer, each core can allocate a number of threads. Taking advantage of the hardware architecture,

computationally intensive parts of RMG such as reaction generation and chemical data computation can be executed in parallel. Python's multiprocessing package effectively side-steps the *Global Interpreter Lock* (GIL) by using processes instead of threads. The GIL mechanism is used by the CPython interpreter to assure that only one thread executes Python bytecode at a time aiming to make the object model, e.g., Python dictionaries, implicitly safe against concurrent access (Python). In this work, Python's multiprocessing package's pool object is used to parallelize function execution across multiple input values by distributing the input data across processes, so called data parallelism (Python).

Due to the dominant overall contribution of the reaction generation and chemical data calculation phases to the automated mechanism generation time, parallelizing these two computationally intensive parts of RMG is expected to result in considerable performance gains. The chemical data computing phase may become more demanding in the future as more advanced quantum chemical methods replace the semi-empirical methods currently supported (MOPAC2016). A schematic of the parallel workflow is presented in Fig. 3.

Despite the potential for parallel computing in the aforementioned target areas, highly non-uniform task workloads hamper the parallel efficiency as it becomes increasingly

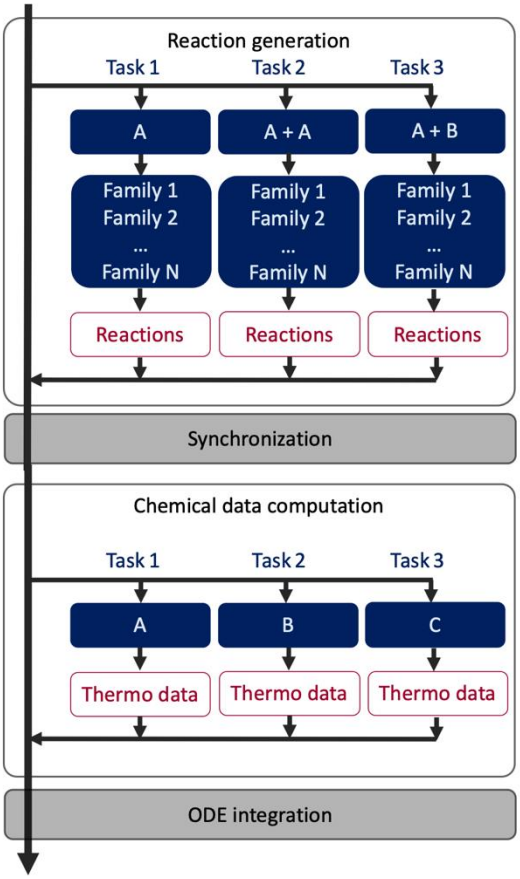


**Figure 3: Parallelization scheme for the reaction generation and chemical data computation phases as part of one enlargement iteration in RMG. The scheme distributes the generation of reactions by creating one task per molecule and molecule-molecule pair and subsequently creates reactions through application of the list of reaction families. The generated reactions created per task are sent to the base process, bundled together and further processed in the synchronization phase, where newly generated species are identified that have no thermodynamic properties associated before creating one task per species in the chemical data computation phase for submission to the QMTP part of RMG. After all tasks are completed the spawned processes are joined together to continue the ODE integration.**

difficult for the task scheduler to achieve a uniform workload across the available processes. Figure 4 presents the percentage of generated reactions per reaction family

over the course of the simulation for the selected cases in Tab. 1.

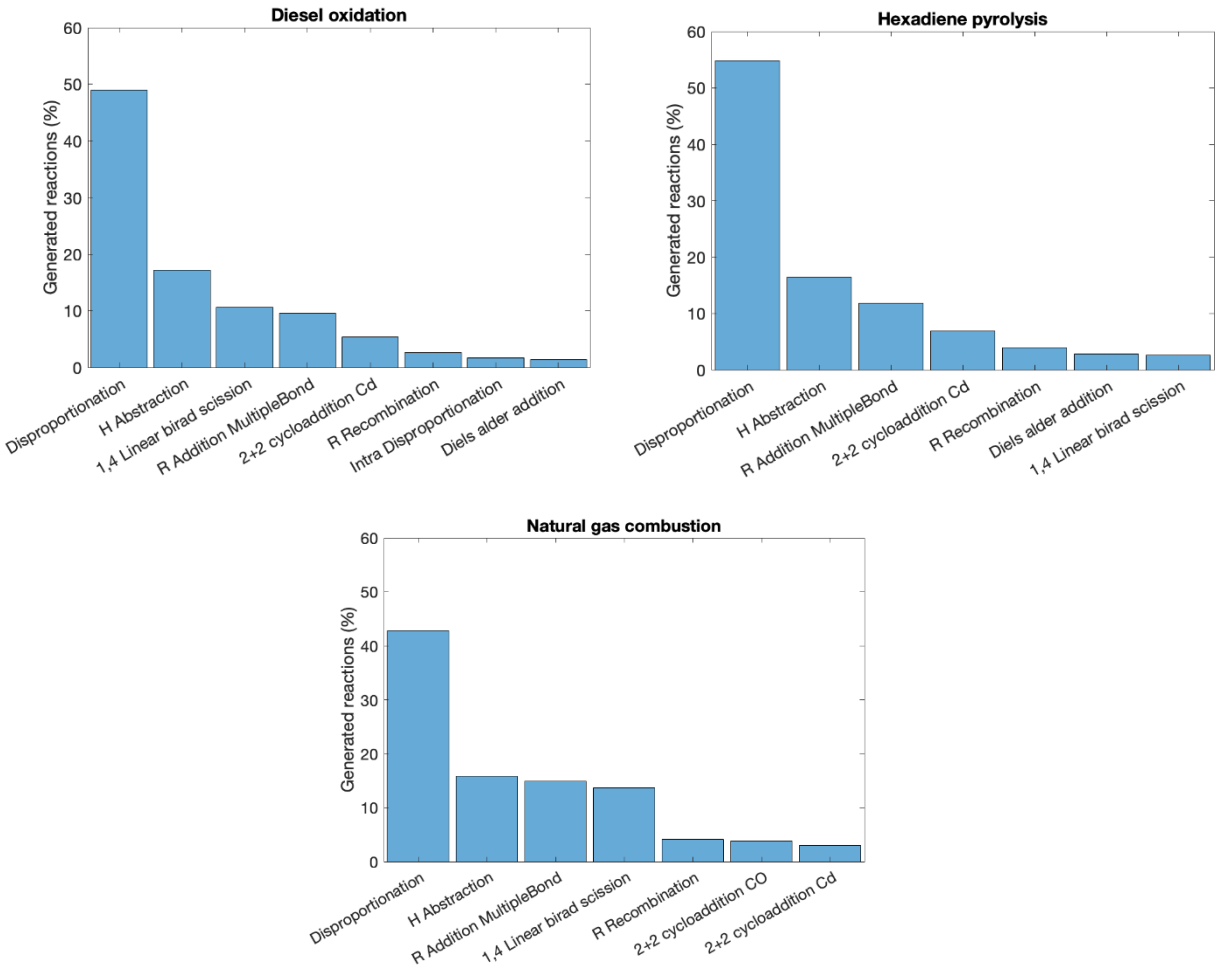


**Figure 4: Generated reactions per reaction family through the course of a simulation. Out of 66 available reaction families in RMG, only a limited number of families is responsible for the large majority of the generated reactions. Only reaction families that contribute more than 1% to the generated reactions are presented.**

For all three cases most reactions are generated using the disproportionation and H-abstraction reaction family. However, the number of generated reactions is highly dependent on the molecules that are paired together. For a molecule pairing that is highly reactive, often with a larger number of atoms, submitting one task including all

available reaction families in RMG will lead to rather large loads in comparison to the load associated with a more unreactive molecule pairing. Figure 5 is a heat map of the number of generated reactions throughout an RMG simulation between two species represented as SMILES. It can be observed that the number of reactions per molecule pair is highly dependent on the nature of the species and varies widely.

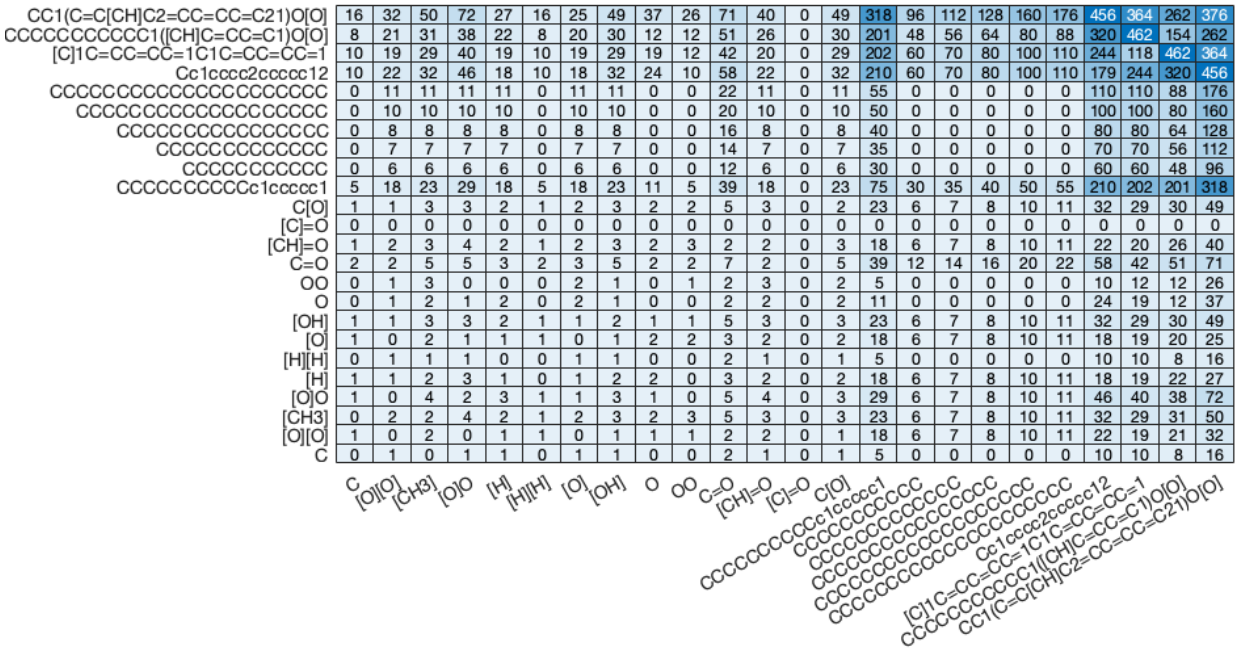


**Figure 5: Heat map showing the number of generated reactions between two species presented as SMILES during an RMG simulation. Darker colors imply a higher number of generated reactions.**

In response to this imbalance, parallelization across the reaction families for molecules larger than ten non-hydrogen atoms is implemented in RMG to create tasks with a computational load that is more uniformly distributed. A threshold of ten non-hydrogen atoms has been chosen based on scaling tests exhibiting a performance loss when employing family splitting with species that have a smaller number of atoms. The

schematic of the workflow is outlined in Fig. 6.

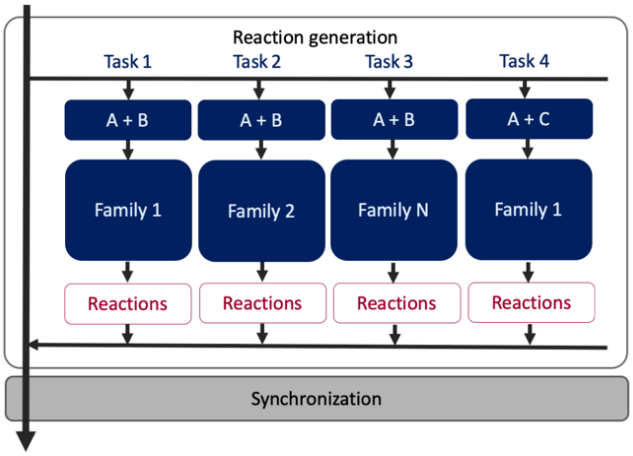


**Figure 6: Reaction generation schematic with family splitting to address unbalanced task loads. In Task 1, bimolecular reactions using reactant A and B and RMG's reaction family "Family 1" are created. In Task 2, bimolecular reactions using reactant A and B and "Family 2" are created and in Task 3, bimolecular reactions using several families "Family N" are created. The same task generation method is then applied for bimolecular reactions using reactant A + C and "Family 1" like presented in Task 4.**

Load distribution is relatively well balanced for the parallel calculation of thermodynamic properties. Figure 7 shows a histogram of the time to completion for the calculation of the

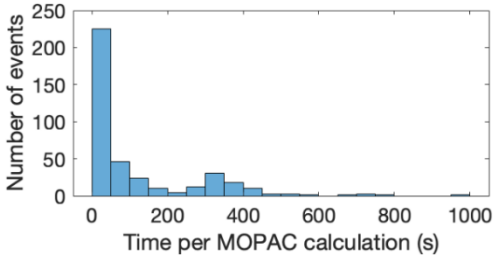


**Figure 7: Distribution of the individual times to successfully complete an QMTP calculation for 390 unique hydrocarbons ($C_2$ – $C_{18}$) using MOPAC2016 at the PM7 level.**

thermodynamic properties of 390 hydrocarbons ranging from $C_2$ to $C_{18}$ using QMTP methods. The majority of the calculations take less than 25 s to complete, although some require more than 50 s. Due to the large number of events less than 25 s, load balancing

is expected to be less of an issue in this case. Furthermore, it is outside the scope of this work to modify the computation time of semi-empirical methods called from within RMG.

# 4 Benchmarking the scalability strategies

## 4.1 Parallel efficiency and load distribution

To evaluate the parallel performance, the elapsed wall clock time or real-time and the parallel efficiency for the reaction generation and thermodynamic property calculation phase are analyzed outside an RMG simulation in three standalone test cases, A to C, with results shown in Tab. 2.

**Table 2: Standalone testcases to analyze wall clock time and parallel efficiency for parallel reaction generation and chemical data computation. The reactants are obtained from cases presented in Tab. 1.**

| | | A | B | C | Thermodynamic property calculation |
|---|---|---|---|---|---|
| **Number of reactants** | | 52 | 50 | 100 | 390 |
| **Reactants obtained from** | | Initial stage of diesel oxidation | Hexadiene pyrolysis | Rich natural gas combustion | Unique hydrocarbons ranging from $C_2$ to $C_{18}$ |
| **Generated reactions** | | 25 987 | 51 953 | 37 675 | - |
| **Wall clock time reduced to** | **Family splitting** | 10% | 10% | 4% | 5% |
| | **No family splitting** | 15% | 12% | 14% | |
| **Speedup factor 40 processes** | **Family splitting** | 10.3 | 10.2 | 24.0 | 18.0 |
| | **No family splitting** | 6.6 | 8.3 | 7.0 | |

The average number of atoms per reactant decreases from test A to C. Strong scaling tests were chosen for the analysis meaning the scaling behavior of a fixed problem size

with an increasing number of processes was assessed. The parallel efficiency measures the efficiency of processes to execute a given parallel algorithm and is calculated with Eq. 4:

$$E = \frac{T_1}{m \cdot T_m}, \qquad \textbf{Eq. 4}$$

where $T_1$ and $T_m$ are wall clock times to complete the problem, using one up to $m$ equipped

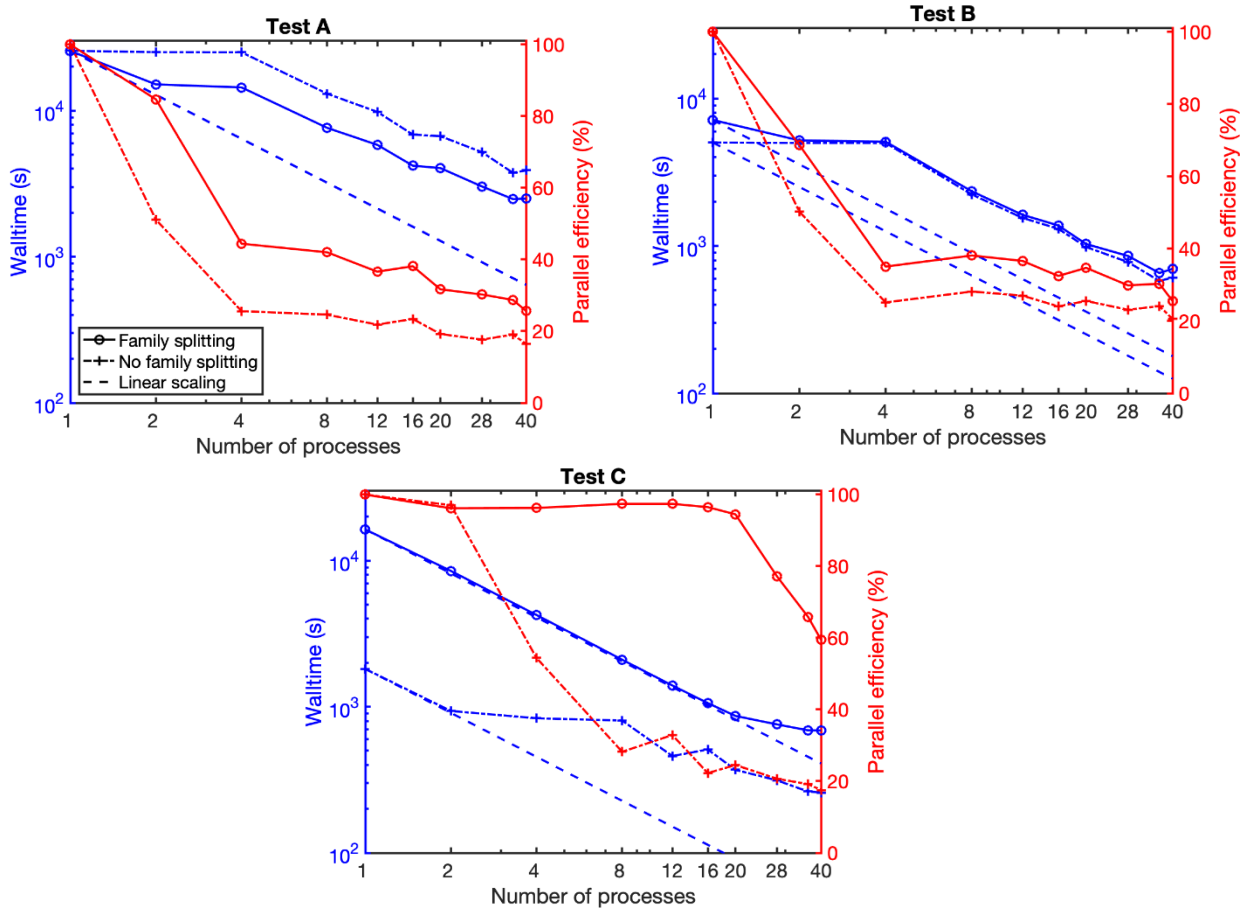


**Figure 8: Wall clock time and parallel efficiency for strong scaling tests using core species generated in the initial diesel oxidation, hexadiene pyrolysis, and natural gas combustion cases as reactants for reaction generation. The maximum achievable speed-up is presented as dashed lines for linear scaling.**

processes, respectively. The scaling tests were carried out on a MacBook Pro (Early 2015, 2.9 GHz Intel Core i5, 8 GB 1867 MHz DDR Memory) and our in-house cluster

(Silicon Mechanics, Rackform R4422.v6, two Intel® X®(R) CPU E5-2630 v4 @ 2.20GHz each equipped with ten cores, 128 GB 2400 MHz DDR4 Memory). Hyperthreading allows for the execution of up to forty processes per node. Both environments showed similar scaling, while the MacBook was only tested up to four processes. The cluster results are presented here. Scaling tests were performed with and without splitting the reaction families over several tasks to improve load balancing, see Fig. 8. In test C family splitting allows for almost linear scaling and a parallel efficiency of over 90% up to twenty processes. The parallel efficiency then decreases to 60% for forty processes, probably due to the use of hyperthreading. Without family splitting, the parallel efficiency in test C decreases to about 50% with just four processes and to less than 20% for forty processes. However, due to relatively small reactant sizes and low reactivities in case C, reaction generation is a rather fast process and using family splitting considerably increases the number of tasks to be submitted to the task scheduler. This larger number of tasks causes the test C wall clock time with family splitting to be significantly higher than the wall clock time without family splitting when using one to forty processes. For test B, with medium sized reactants, the wall clock times with or without family splitting are very similar. In test A, with relatively large and reactive species, wall clock times are reduced through family splitting. These observations lead to the heuristic of employing family splitting only for reactants larger than ten non-hydrogen atoms. Finally, using forty processes instead of a single process reduces the wall clock time to at least 15% of its original value. These improvements correspond to speed-up factors ranging between 6.6 and twenty-four. The parallel efficiency was improved with family splitting, but in tests A and B still fell rapidly with only a few processes in use.

Similar to the analysis for reaction generation, Fig. 9 presents the elapsed wall clock time and parallel efficiency for the calculation of thermodynamic properties in an isolated test, i.e. not embedded in an RMG simulation.

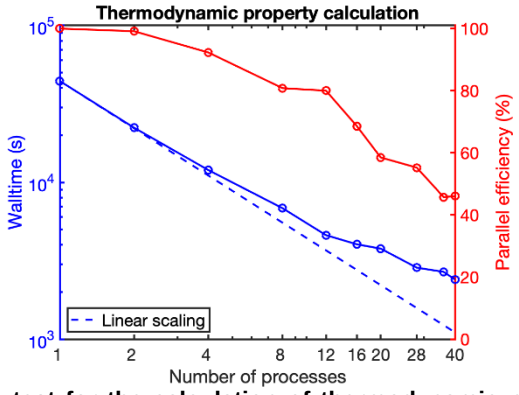


**Figure 9: Strong scaling test for the calculation of thermodynamic properties of 390 unique hydrocarbons ranging from $C_2$ to $C_{18}$ with MOPAC2016 at the PM7 level.**

Parallelization allowed for a wall clock time reduction to about 5% of its original value, corresponding to a speed-up factor of eighteen when using forty processes. Only small deviations from linear scaling as well as a smooth decline in parallel efficiency can be observed up to twelve processes. For twelve to forty processes the parallel efficiency decrease is relatively steep, ending below 50%. As in the reaction generation tests, the decrease in parallel efficiency implies that further increasing the number of processes will result in only marginal reductions of the overall wall clock time. To improve our understanding of the decreasing parallel efficiency, two more possible bottlenecks besides load balancing are investigated: contention for shared resources and communication time.

### 4.1.1 Shared resources

Shared resources can refer to shared variables within sections of the code designated as critical. Critical sections are defined to ensure shared data is accessed in a serial way, thus avoiding race conditions. As a result, only one process may execute a critical section, or access a shared variable within a critical section simultaneously. Contention for shared resources during the parallel generation of reactions and calculation of thermodynamic properties is assessed by calculating the sum of the time spent in individual tasks as a function of the number of processes. Figure 10 shows that the total time spent in the tasks remains generally constant up to twenty processes, after which it gently increases for most of the tests.

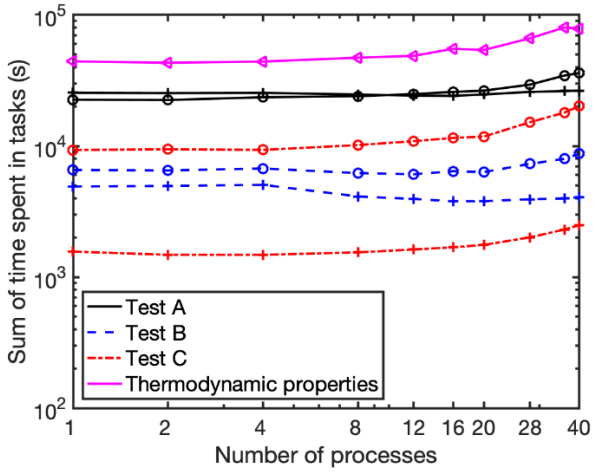


**Figure 10: Sum of the times spent in tasks generating reactions or calculating thermodynamic properties as a function of the number of processes. Circles represent tests including the splitting of reaction families across multiple tasks while crosses represent tests without family splitting.**

Based on the results shown in Figure 10, it becomes clear that writing to shared variables within critical sections is not an important factor when analyzing the parallel efficiency decrease. This is likely mainly due to the internal forking algorithm used in Python's multiprocessing package that generates copies of the base process when spawning

processes and the fact that for reaction generation in RMG, processes only read the static RMG databases but never write data to them, which would require a locking mechanism to avoid race conditions. Writing to shared variables only occurs when the generated reactions from a process are added to the collection of reactions on the base process or the calculated thermodynamic properties are returned to the molecule. Both actions are handled internally by the `multiprocessing` package.

#### 4.1.2 Communication overhead

In the case of a physical shared resource such as system memory, the off-chip bandwidth puts a limit on the maximum data transfer rate that can be achieved between the system memory and the CPU chip. As discussed before, Python's GIL prevents the simultaneous execution of multiple threads. Therefore, the parallel execution of tasks is achieved by running them through processes with separate memory spaces and, as a result, data communication occurs through message passing. This step requires the serialization of the object hierarchy into a binary representation, the transfer of that byte stream, and finally the inversion operation, or deserialization of the stream into the object hierarchy containing the original data. The current parallel reaction generation scheme requires two communication operations: sending the resonance structure pair from the base process to the parallel processes and sending a list of generated reactions back from the parallel processes to the base process. Since the transferred objects consist of highly layered

and complex objects, data communication contributes significantly to the overall wall clock time.

The time spent on data communication $T_{DC}$ is calculated with

$$T_{DC} = T_m - \frac{\sum T_{Task}}{m}, \qquad \text{Eq. 5}$$

where $T_{Task}$ is the time spent in one task, treated in parallel with $m$ other tasks. Therefore, the second term on the right-hand side represents the average sum of time spent in a task. As the time spent on data communication varies for different processes due to the

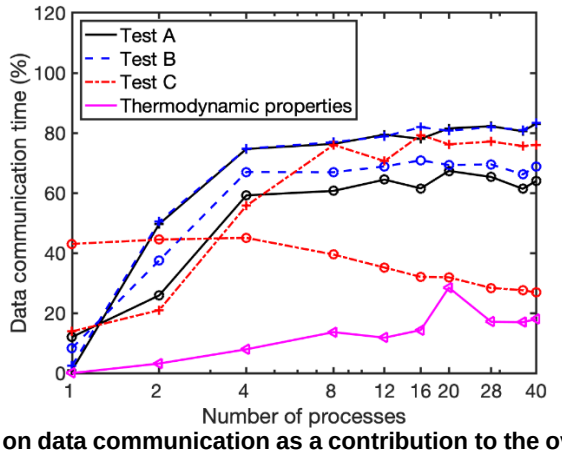


**Figure 11: Time spent on data communication as a contribution to the overall wall clock time and as a function of the number of processes. Circles represent tests including the splitting of reaction families across multiple tasks while crosses represent tests without family splitting.**

distribution of task sizes, $T_{DC}$ represents an average across the processes as opposed to a process-specific metric. The percentage of time spent on data communication as a portion of the overall wall clock time is presented as a function of the number of processes in Fig. 11. The test cases are identical to those used previously and defined in Tab. 2. Generally, for reaction generation, a larger fraction of time is spent on data communication as the number of processes increases, eventually taking a majority of wall clock time. The exception to this is test C with family splitting. Due to the submission of

a large number of tasks with relatively short execution times, the data communication time as a percentage of the overall wall clock time is larger for a smaller number of processes. This trend also does not hold for thermodynamic property calculation. In that case, the data communication time in percent is considerably lower as compared to the reaction generation cases.

Overall, the results suggest that the drop in parallel efficiency for reaction generation is mainly due to time spent on data communication, which is observed to take up to 80% of the overall wall clock time.

## 4.2 RMG simulations

After having assessed the scalability characteristics of the individual components, we now assess their impact on wall clock time and memory consumption for the RMG simulations summarized in Tab. 1. First, we compared reaction filtering, pruning, or multiprocessing in isolation to the base cases. Next, we employed the heuristic algorithm improvements — reaction filtering and pruning — together. Finally, we used all the improvement strategies together. Table 3 presents the obtained wall times for each case.

Figure 12.a presents the wall clock time and memory consumption for the diesel oxidation case. Pruning or multiprocessing alone did not have a significant impact on the number of core species in the model after a wall clock time of fourteen days. However, reaction filtering allowed for a larger number of core species. A combination of pruning, reaction filtering and multiprocessing allowed for about twice as many core species to be generated in the same wall clock time as compared to the base case. Steep increases in

**Table 3: Wall times for model generation with the suggested changes in RMG. Evaluated at 24, 125, and 218 core species generated for diesel oxidation, hexadiene pyrolysis, and natural gas combustion, respectively. FS indicates splitting reaction families over several tasks for reactants containing more than ten non-hydrogen atoms. No FS indicates that all reaction family matches for one reactant or reactant pair are evaluated in a single task.**

| Wall times (h) | Base case | Multiprocessing | Pruning | Reaction filtering | Pruning + Reaction filtering | Pruning + Reaction filtering + Multiprocessing | |
|---|---|---|---|---|---|---|---|
| | | | | | | No FS | FS |
| Diesel oxidation | 249.6 | 118.9 | 199.0 | 144.9 | 37.1 | 22.1 | 18.7 |
| Hexadiene pyrolysis | 95.2 | 47.4 | 81.9 | 16.0 | 15.8 | - | 7.4 |
| Natural gas combustion | 11.1 | 9.9 | 130 | 1.9 | 4.5 | - | 1.9 |

wall clock time for the addition of a single species were observed even when splitting the reaction families over several processes. We suspect that this is because a large computational load is associated with a single reactant pair and reaction family. Such a task is not divisible under the current family splitting strategy. A further reduction in wall clock time, especially for the first steep increase, could be obtained by generating more and smaller tasks for the work done within the reaction generation for one family. This is also expected to decrease the data communication time and therefore improve the parallel efficiency for this case. Speed-up factors for the three cases and the suggested code improvements are reported in Tab. 4. The base cases are also presented in Fig. 2.

Monitoring the memory consumption is important because the forking algorithm limits the number of available parallel processes by the memory consumption of the base process

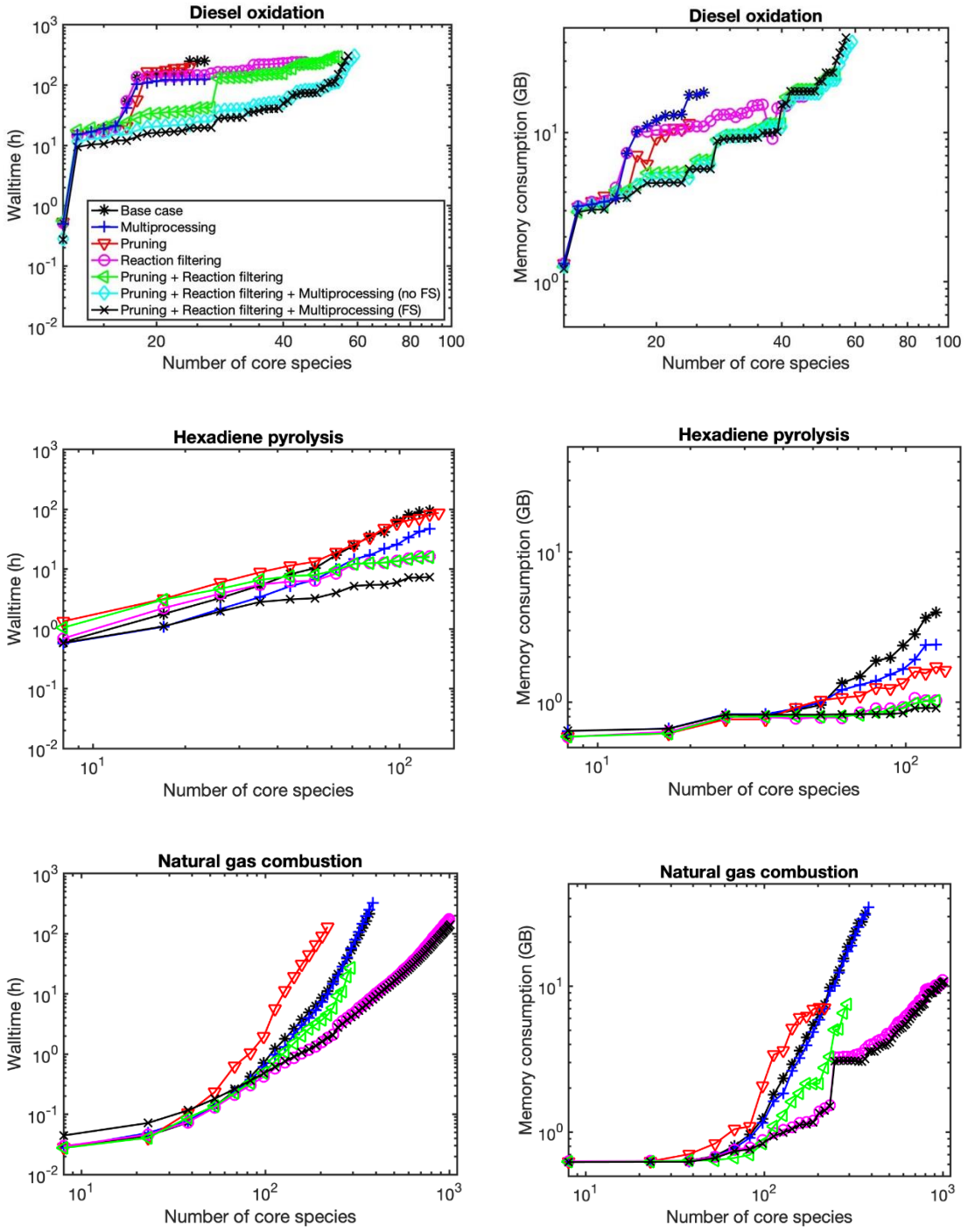


**Figure 12: Wall clock time and memory consumption as a function of the number of species added to the core employing the discussed scalability strategies either alone or combined for the cases outlined in Tab. 1.**

and the maximum available memory on the machine. Therefore, when using the parallel computing strategy, the memory consumption mimics the base case, limiting the number of allowed processes to below the available maximum of forty. Due to these restrictions for very memory intense cases, parallel computing might not be possible, causing the simulation to be carried out in serial. However, the combination of pruning and reaction filtering allows for a smaller memory footprint compared to the base case. Therefore, adding the multiprocessing option to these two heuristic strategies allows for better leveraging the parallel computation strategy. Similar observations can be made for the hexadiene pyrolysis and natural gas combustion cases presented in Fig. 12.b and c, respectively.

The speed-up between using algorithm heuristics alone and combining them with parallel computing is still about a factor of two. This seems to be a comparatively small

**Table 4: Speed-up factor compared with the base cases for the suggested changes in RMG. Evaluated at 24, 125, and 218 core species generated for diesel oxidation, hexadiene pyrolysis, and natural gas combustion, respectively. FS indicates splitting reaction families over several tasks for reactants containing more than ten non-hydrogen atoms. No FS indicates that all reaction family matches for one reactant or reactant pair are evaluated in a single task.**

| Speed-up | Base case | Multiprocessing | Pruning | Reaction filtering | Pruning + Reaction filtering | Pruning + Reaction filtering + Multiprocessing | |
|---|---|---|---|---|---|---|---|
| | | | | | | No FS | FS |
| Diesel oxidation | 1 | 2.1 | 1.3 | 1.7 | 6.7 | 11.3 | 13.3 |
| Hexadiene pyrolysis | 1 | 2.0 | 1.2 | 6.0 | 6.0 | - | 12.9 |
| Natural gas combustion | 1 | 1.1 | 0.1 | 6.0 | 2.5 | - | 6.0 |

improvement, however, as Amdahl's law (Amdahl, 1967) dictates, the maximum achievable speed-up in an enlargement iteration is governed by the time spent in the serial parts of the code.

# 5 Conclusions and perspectives

The complexity of modeling processes driven by chemical kinetics has been a long-standing challenge. With the concerted efforts outlined in this paper, we have identified critical bottlenecks and provided scalability solutions to make computer-aided reaction mechanism construction a tool with practical usefulness in the generation of predictive detailed reaction mechanisms. It was shown that bottlenecks for the overall performance of RMG simulations are dynamic in nature and depend on the state of the simulation and the methods used for the calculation of chemical data. A three-pronged approach consisting of code optimization, algorithm heuristics and parallel computing was employed to accelerate the essential phases of the rate-based enlargement procedure. Tests that measured the individual scalability characteristics have shown speed-up factors with respect to the use of a single process ranging between 6.6 and twenty-four for reaction generation and of about eighteen for thermodynamic property calculation. However, the parallel computing benchmarking also revealed issues that are of fundamental importance to automated mechanism generation tools. The goal of augmenting important entities with metadata competes with scalability and operation speed. Therefore, we believe that future efforts should be directed at devising novel data structures and approaches that fully take advantage of the vast computational resources available today, while creating information-rich models that enable analysis and scrutiny.

To conclude, this work demonstrated how the new version 2.4.0 of the Reaction Mechanism Generator opens up new opportunities for the construction of more comprehensive and accurate mechanisms of chemical processes. It also creates avenues for the modeling of real-world processes that were previously too complex to model. RMG is a free-to-use open-source code under the MIT X/11 license. The scalability improvements described in this paper are implemented in the latest version found at https://github.com/ReactionMechanismGenerator/RMG-Py. Expecting that the trend towards computer-assisted reaction mechanism generation continues, the identified hurdles and proposed solutions in this work might be applicable in a broader context outside the realm of RMG and may impact other adjacent computational approaches. More specifically, even though each of the available automatic mechanism generators are quite unique in their algorithms they are all expected to benefit from the described strategies when adapted accordingly. Parallelization of thermodynamic property calculation and reaction generation would be helpful in any software where the thermodynamic data or reactions are algorithmically generated.

# 6 Acknowledgements

The authors like to thank RMG developers, specifically Richard H. West, Joshua W. Allen and Murat Keceli for helpful discussions. Nick M. Vandewiele acknowledges financial support for a postdoctoral fellowship at MIT from the Belgian American Educational Foundation (BAEF). Agnes Jocher acknowledges financial support from the DFG Research Fellowship JO 1526/1-1. This research was supported by subcontract 7F-30180 to MIT from UC Chicago Argonne LLC. It is a component of the Exascale

Computing Project (ECP), Project Number 17-SC-20-SC, a collaborative effort of two DOE organizations, the Office of Science and the National Nuclear Security Administration, responsible for the planning and preparation of a capable exascale ecosystem including software, applications, hardware, advanced system engineering, and early test bed platforms to support the nation's exascale computing imperative

Declarations of interest: The authors declare they have no conflicts of interest.

# References

Allen, J. W., et al., 2012. Automatic estimation of pressure-dependent rate coefficients. Phys. Chem. Chem. Phys. 14, 1131–55.

Allen, J. W., et al., 2014. A coordinated investigation of the combustion chemistry of diisopropyl ketone, a prototype for biofuels produced by endophytic fungi. Combust. Flame 161, 711–724.

Amdahl, G. M., 1967. Validity of the single processor approach to achieving large scale computing capabilities. Proc. AFIPS Spring Jt. Comput. Conf. 30.

Behnel, S. et al., 2011. Cython: The Best of Both Worlds. Comput. Sci. Eng. 13, 31–39.

Benson, S. W., 1968. Thermochemical Kinetics. 1st, John Wiley & Sons Ltd.

Blurock, E. S., 1995. Reaction - System for modeling chemical reactions. J. Chem. Inf. Comput. Sci. 35.

Broadbelt, L. J., et al., 1996. Computer generated reaction modelling: Decomposition and encoding algorithms for determining species uniqueness. Comput. Chem. Eng. 20, 113–129.

Burke, M. P., 2016. Harnessing the Combined Power of Theoretical and Experimental Data through Multiscale Informatics. Int. J. Chem. Kinet. 48, 212–235.

Carr, A. G., et al., 2015. Supercritical Water Treatment of Crude Oil and Hexylbenzene: An Experimental and Mechanistic Study on Alkylbenzene Decomposition. Energy & Fuels 29, 5290–5302.

Class, C. A., et al., 2016. Automatic mechanism generation for pyrolysis of di-tert-butyl sulfide. Phys. Chem. Chem. Phys. 18, 21651–21658.

Cordella, L. P. P., et al., 2004. A (sub)graph isomorphism algorithm for matching large graphs. IEEE Trans. Pattern Anal. Mach. Intell. 26, 1367–1372.

Eyring, H., 1935. The Activated Complex in Chemical Reactions. J. Chem. Phys. 3, 107.

Foggia, P., et al., 2001. A Performance Comparison of Five Algorithms for Graph Isomorphism. Proceedings of the 3rd IAPR TC-15 Workshop on Graph-based Representations in Pattern Recognition. 188–199.

Gansner, E. R., North, S. C., 1999. An open graph visualization system and its applications to software engineering. Pr. Exper 0, 1–5.

Gao, C. W., et al., 2015. JP-10 combustion studied with shock tube experiments and modeled with automatic reaction mechanism generation. Combust. Flame 162, 3115–3129.

Gao, C. W., et al., 2016. Reaction Mechanism Generator: Automatic construction of chemical kinetic mechanisms. Comput. Phys. Commun. 203, 212–225.

Gao, C. W., 2016, Ph.D. dissertation, MIT. https://dspace.mit.edu/handle/1721.1/104205?show=full

Green, W. H., 2007. Predictive Kinetics: A New Approach for the 21st Century. Adv. Chem. Eng. 32, 1–50.

Gudiyella, S., et al., 2018. Modeling Study of High Temperature Pyrolysis of Natural Gas. Ind. Eng. Chem. Res. 57 (22), 7404-7420.

Han, K., et al., 2017. On-the-Fly Pruning for Rate-Based Reaction Mechanism Generation. Comput. Chem. Eng. 100. 1-8.

Hansen, N., et al., 2013. The Predictive Capability of an Automatically Generated Combustion Chemistry Mechanism; Chemical Structures of Premixed iso-Butanol Flames. 160. 2343-2351.

Jalan, A., et al., 2010. Predicting solvation energies for kinetic modeling. Annu. Reports Sect. 'C' Phys. Chem. 106, 211.

Jalan, A., et al., 2013. An Extensible Framework for Capturing Solvent Effects in Computer Generated Kinetic Models. J. Phys. Chem. B 117, 2955–2970.

Jones, E., Oliphant, T. SciPy Open Source Scientific Tools for Python. www.scipy.org. (accessed 18 February 2019).

Li, S., Petzold, L. R. 1999. Design of New DASPK for Sensitivity Analysis.

Magoon, G. R., Green, W. H., 2013. Design and implementation of a next-generation software interface for on-the-fly quantum and force field calculations in automated reaction mechanism generation. Comput. Chem. Eng. 52, 35–45.

Millett, L. I., Fuller, S. H., 2011. Computing Performance: Game Over or Next Level?, in Computer, vol. 44,no. , pp. 31-38. doi:10.1109/MC.2011.15

MOPAC2016, James J. P. Stewart, Stewart Computational Chemistry, Colorado Springs, CO, USA, HTTP://OpenMOPAC.net (accessed 18 February 2019).

Narayanaswamy, K., et al., 2016. A component library framework for deriving kinetic mechanisms for multi-component fuel surrogates: Application for jet fuel surrogates. Combust. Flame. 165, 288-309.

Prozument, K., et al., 2014. A Signature of Roaming Dynamics in the Thermal Decomposition of Ethyl Nitrite: Chirped-Pulse Rotational Spectroscopy and Kinetic Modeling. J. Phys. Chem. Lett. 5, 3641–8.

PyDAS. https://github.com/jwallen/PyDAS (accessed 18 February 2019).

Python 2.7.15 documentation. https://docs.python.org/2/ (accessed 01 February 2019).

Rangarajan, S., et al., 2012. Language-oriented rule-based reaction network generation and analysis: Description of RING. Comput. Chem. Eng. 45, 114–123.

Seyedzadeh Khanshan, F., West, R. H., 2016. Developing detailed kinetic models of syngas production from bio-oil gasification using Reaction Mechanism Generator (RMG). Fuel 163, 25–33.

Sharma, S., et al., 2007. Automated reaction mechanism generator and applications to model hexadiene doped methane flames. AIChE Annual Meeting, Conference Proceedings.

Sharma, S., et al., 2010. Modeling of 1-hexadiene, 2,4-hexadiene and 1,4-hexadiene doped methane flames: Flame modeling, Benzene and Styrene formation. Combust. Flame 157, 1331-1345.

Shi, Y., et al., 2011. Redesigning combustion modeling algorithms for the Graphics Processing Unit (GPU): Chemical kinetic rate evaluation and ordinary differential equation integration. Combust. Flame 158, 836–847.

Song, J., 2004. Building Robust Chemical Reaction Mechanisms: Next Generation of Automatic Model Construction Software. MIT.

Suleimanov, Y. V., Green, W. H., 2015. Automated Discovery of Elementary Chemical Reaction Steps Using Freezing

String and Berny Optimization Methods. J. Chem. Theory Comput. 11, 4248–4259.

Susnow, R. G., et al., 1997. Rate-based construction of kinetic models for complex systems. J. Phys. Chem. A 101, 3731–3740.

Van de Vijver, R., et al., 2015. Automatic Mechanism and Kinetic Model Generation for Gas- and Solution-Phase Processes: A Perspective on Best Practices, Recent Advances, and Future Challenges. Int. J. Chem. Kinet. 47, 199–231.

van der Walt, S., et al., 2011. The NumPy Array: A Structure for Efficient Numerical Computation. Comput. Sci. Eng. 13, 22–30.

Vandewiele, N. M., et al., 2012. Genesys: Kinetic model construction using chemo-informatics. Chem. Eng. J. 207, 526–538.

Vandewiele, N. M., et al., 2015. Kinetic Modeling of Jet Propellant-10 Pyrolysis. Energy Fuels 29, 1, 413-427.

Vermeire, F. H., et al., 2017. Experimental and kinetic modeling study of the pyrolysis and oxidation of 1,5-hexadiene: The reactivity of allylic radicals and their role in the formation of aromatics. Fuel 208, 779-790.

Warth, V., et al., 2000. Computer based generation of reaction mechanisms for gas-phase oxidation. Comput. Chem. 24, 541–560.

Zhang, P., et al., 2018. Modeling Study of the anti-knock tendency of substituted phenols as additives: An application of the Reaction Mechanism Generator (RMG). Phys. Chem. Chem. Phys. 20, 10637–10649.