A Parallelised High Performance Monte Carlo Simulation Approach for Complex Polymerisation Kinetics

Hugh Chaffey-Millar,[a] Don Stewart,[b]

Manuel M. T. Chakravarty,[b] Gabriele Keller,[b,*] Christopher Barner-Kowollik[a,*]

[a]*Center for Advanced Macromolecular Design, School of Chemical Sciences and Engineering, The University of New South Wales, Sydney, NSW 2052, Australia, c.barner-kowollik@unsw.edu.au*

[b]*Programming Languages and Systems, School of Computer Science and Engineering, The University of New South Wales, Sydney, NSW 2052, Australia, keller@cse.unsw.edu.au*

keywords: Monte Carlo simulation, kinetics (polym.), computer modeling, parallel computation

Abstract

A novel, parallelised approach to Monte-Carlo simulations for the computation of full molecular weight distributions arising from complex polymerization reactions is presented. The parallel Monte Carlo method constitutes perhaps the most comprehensive route to the simulation of full molecular weight distributions of multiple chain length polymer entities and can also provide detailed microstructural information. New fundamental insights have been developed with regard to the Monte-Carlo process in at least three key areas: (i) an insufficient system size is demonstrated to create inaccuracies via poor representation of the most improbable events and least numerous species; (ii) advanced algorithmic principles and compiler technology known to computer science have been used to provide speed improvements and (iii) the parallelisability of the algorithm has been explored and excellent scalability demonstrated. At present, the parallel Monte-Carlo method presented herein compares very favourably in speed with the latest developments in the

h-p Galerkin method based PREDICI software package while providing significantly more detailed microstructural information. It seems viable to fuse parallel Monte-Carlo methods with those based on the h-p Galerkin methods to achieve an optimum of information depths for the modelling of complex macromolecular kinetics and the resulting microstructural information.

1 Introduction

It is often useful to be able to predict the behavior of a chemical system by solution of the differential equations that describe rates of appearance or disappearance of the chemical constituents. For all but the simplest systems, no analytical solutions are known, and the physical chemist must resort to numerical methods to predict the outcome of more complex systems. Polymerising systems represent a particularly important class of chemistry, both for industry and academia. These systems have the feature that the important species have not only a concentration but a distribution of chain lengths. Consequently the systems of differential equations which describe these systems require considerable computational effort for their solution.

Many polymer chemists use kinetic modelling tools to help explain and predict the outcome of complex reactions[1–6] and a subset of the authors have, for some time, been interested in the kinetics of star polymerisations.[7–9] A difficulty frequently encountered with the kinetics of star polymerisations was the computational burden of their complex kinetic schemes which, using the program PREDICI® [1] provided by CiT (Computing in Technology, GmbH) , resulted in simulation times exceeding a day, in certain instances. In order to overcome this barrier to efficient research, methods have been developed to simulate complex systems in a more reasonable time-frame.

A number of solutions exist for the simulation of polymerisation reactions. The commercial software program PREDICI is one such example however researchers in this field frequently implement their own software in general purpose languages (such as C) via a Monte Carlo approach[2,4,5,10] or by solving the moment equations.[11–14] PREDICI is able to compute full molecular weight distributions (MWDs), as is a Monte Carlo approach. Solution of the moment equations cannot give full MWDs without additional assumptions regarding the shape of distributions.[1,12,14]

The Monte Carlo method is the focus of the current research. This method has been applied before to the simulation of chemical kinetics,[15,16] in particular the kinetics of polymerisation reactions.[2,4,5,10]

Additionally, Monte Carlo models are of significant importance in many fields spanning both the physical sciences, mathematics and financial modeling.[17,18]

With a view to fast simulation of complex polymerisation models, a Monte Carlo method, similar to that used previously[2,4,5,10] to simulate polymerisation kinetics, has been explored and further developed in the following manner. (1) The method has been adapted to operate on a parallel computer, providing significant speed increases and excellent scalability. (2) Various optimisations have been devised and implemented and their effect on the speed of simulation discussed. (3) Research has been conducted into the fundamental nature and performance of the method, yielding insights into where the method fails and excels. (4) The program is released as open source software (previous authors have kept their code in-house).

At the heart of the program is a Monte Carlo algorithm in which the stochastically governed reactions of a population of molecules are simulated, whilst the time evolved by the system is constantly updated. Although the reactions that actual molecules undergo are simulated explicitly at a microscopic level, the rates at which they react and the physical parameters governing these rates are associated with the differential equations known to physical chemistry as rate laws. Thus, the algorithm is essentially a numerical method for the solution of sets of coupled differential equations that arise in the description of chemical systems.

The purpose and structure of this publication is as follows. We first describe the algorithm from a numerical/mathematical point of view. Secondly, some caveats and properties of the system are discussed. Thirdly, the method is more closely examined with reference to actual implementation strategies of the Monte Carlo algorithm and how our implementation makes use of intelligent data structures for improved performance. Fourthly, methods of parallelisation are discussed. Finally, a performance analysis is performed yielding insights into the scalability of the program as well as how various optimisations have lead to speed improvements.

It is also hoped that the current document, through it's focus on implementation details, and the open source nature of the code which has been developed, will serve as a starting point for (polymer) chemists hoping to develop Monte Carlo methods for future research.

2 Theoretical background of the Monte Carlo method

2.1 Polymerisation kinetics

A simple polymerising system that one might wish to study consists of decomposition of initiator molecules to form free-radicals ($I_2 \rightarrow 2I\cdot$), reaction of those free radicals with monomer to form short polymer chains ($I\cdot + M \rightarrow P_1\cdot$), polymer chain growth/propagation ($P_n\cdot + M \rightarrow P_{n+1}\cdot$) and termination to form dead (i.e. non-free-radical bearing) polymer that undergoes no further growth ($P_n\cdot + P_m\cdot \rightarrow D_{n+m}$). In these reaction equations, P_i denotes a growing chain of length i (here, *length* is taken to mean the number of monomer units which have form the chain). To simulate how such a system behaves, one is required to solve a set of differential equations which, for the above reactions are given by equations 1–4.

$$R_{\text{decomposition}} = \frac{d[I_2]}{dt} = -k_d[I_2] \tag{1}$$

$$R_{\text{initiation}} = \frac{d[I\cdot]}{dt} = 2k_d[I_2] - k_i[I\cdot][M] \tag{2}$$

$$R_{\text{termination}} = \frac{d[D_n]}{dt} = \sum_{i=1}^{n-1} k_t(i, n-i)[P_i\cdot][P_{n-i}\cdot] \tag{3}$$

$$R_{\text{radical}} = \frac{d[P_n\cdot]}{dt} = k_p\left([P_{n-1}\cdot][M] - [P_n\cdot][M]\right) - [P_n\cdot]\sum_{i=1}^{n} k_t(i, n)[P_i\cdot] \tag{4}$$

$$n \in [1, \text{ max chain length in system}] \tag{5}$$

2.2 Solution by Monte Carlo methods

The algorithm, first described by Gillespie,[19] proceeds by choosing reaction events[i] at random, but according to probabilities dictated by the physical rates of the process responsible for each event. After each event, the system's time is advanced by a stochastically determined increment, based once again on the total rate at which events occur in the system (explained later). The probabilities of each reaction are calculated using chemical knowledge of each reaction's rate. More specifically, if there are r reactions, a reaction of index μ, where $1 \leq \mu \leq r$, is chosen if a random number γ_1 lies in the range

[i] Here, the word event is used to mean a reaction by a single or pairs of molecules.

$$\sum_{i=1}^{\mu-1} P_i < \gamma_1 \leq \sum_{i=1}^{\mu} P_i \tag{6}$$

where P_i is the probabilitie of some reaction i. These probabilities are calculated from

$$P_i = \frac{R_i}{\sum_{j=1}^{r} R_j} \tag{7}$$

where R_i is the "chemical" rate of some reaction i. For a bimolecular reaction between species l and m, the chemical rate is calculated from the rate law

$$R_i \quad = \quad \begin{cases} k_i X_l X_m & \text{if reactants are different} \\ k_i X_l (X_l - 1) & \text{if reactants are the same} \end{cases} \tag{8}$$

and for a first order reaction is

$$R_i = k_i X_l \tag{9}$$

In equations 8 and 9, $X_{l,m}$ are the number of particles of species l and m, k_i must be in units of s^{-1}. For reactions that have an order greater than unity the familiar chemical rate coefficient, k_i', must be modified and for a bimolecular reaction undergoes the transformation in Equation 10.

$$\begin{aligned} k_i \quad &= \quad \frac{k_i'}{N_A \cdot V} \quad && \text{for non-identical reacting species} \\ \text{or} \quad & \quad \frac{2k_i'}{N_A \cdot V} \quad && \text{for identical reacting species} \end{aligned} \tag{10}$$

where k_i' has units of $\mathrm{L/mol/K}$, V is the volume of the system and N_A is Avogadro's constant. The volume, V, of the system is determined by the number of particles, N, and the medium density, ρ. The choice of N is discussed below whilst ρ is an experimentally determined physical parameter.

A system at time t, after the occurrence of an event, must have its time incremented as $t \leftarrow t + \Delta t$, where

$$\Delta t = \frac{-\ln \gamma_3}{\sum_{j=1}^{r} R_j} \tag{11}$$

where γ_3 is a random number on the unit interval (0,1] and the denominator is, as in the calculation of reaction probabilities, the rate of all reaction events taking place in the system.

Some authors have used slight variations on the above method. Prescott gave each molecule its own time coordinate and only permitted reaction between species occurring with the same time[2] (within a tolerance) and He et al. employed a hybrid stochastic/analytical method[20] in which the reversible exchange reactions were not computed completely but the populations of the relevant molecules were adjusted periodically according to their known equilibrium constant, providing a speed improvement. This latter method might be applicable to the current simulation program, however it is beyond the scope of the current study and development goals.

3 Implementation/data structures

3.1 Overview

The classic kinetic Monte Carlo algorithm proceeds as follows.

1. **Initialisation.** System is initialised with the correct number of each type of particle.
2. **Reaction probabilities.** The cumulative probability of every reaction is (re)calculated using equation 7.
3. **Reaction Choice.** A reaction, R_i, is chosen using a random number, γ_1, and equation 6.
4. **Chain length Choice.** If any of the (two) reactants are polymeric, the chain length(s) are chosen using random numbers γ_{2a} and γ_{2b}.
5. **Product Creation.** Product molecule(s) are created based on chain length addition strategy appropriate to reaction R_i.
6. **Time Calculation.** The time coordinate for the system is advanced using equation 11, using random number γ_3.

7. Repeat from step 2 until finishing criteria (time, conversion of monomer, etc.) are satisfied.

3.2 Randomly choosing molecules

If all molecules were unimeric, choosing a molecule at random would be simple since all the molecules of a certain species do not have separate identities and can therefore be represented by a single variable which stores the number that exist. Given a set of molecules with chain lengths, the situation is slightly more complicated however and there are two main options:

Option (A) for representing these N molecules would be to store each one explicitly, requiring a list of the chain lengths of every molecule, for which choosing a molecule at random has $O(1)$ time complexity[ii] but $O(N)$ space complexity. Option (B) is to store a compressed representation, comprised of a list of chain lengths and the number of molecules with each chain length. The algorithm for choosing a molecule at random under option (B) can be realised with $O(\log n_{\text{max}})$ time complexity (the detailes of the algorithm are explained below) but only $O(n_{\text{max}})$ space complexity, where n_{max} is the maximum chain length existing for the species in question. Further, using the compressed representation, the entire system representation (for the systems that were the subject of the current research) is of the order of hundreds of kilobytes, meaning it is likely to fit into the cache of most modern processors.

On the one hand, option (A) uses a more efficient algorithm but requires a large amount of memory. On the other hand, since the cache memory is often roughly ten times faster than main memory, it was thought that option (B) with its less efficient $O(\log n_{\text{max}})$ algorithm might out-perform option (A). Additionally, since the system is stored in a more space efficient manner, option (B) might lead to a faster overall implementation if the distributed processes are required to communicate frequently. To determine which method is best, both have been implemented and tested, as described later.

3.3 Decision making based on probabilities

Each reaction event in the system results from three separate stochastic choices: (1) the choice of the timestep (Δt) (2) the choice of a particular reaction (μ) and (3) the choice of which of the many

[ii] Big "O" notation requires some description. Suppose the run time of a piece of code for an input of size N is denoted $g(N)$. The code is deemed to exhibit $O(f(N))$ time complexity if there is a function $f(N)$ such that $\alpha \cdot f(N) \geq g(N)$ for all $N > N_0$ and when α is a positive number. Explained differently, and from a more practical view point, this is similar to saying that $g(N) \propto f(N)$ as long as N is chosen to be non-trivial.

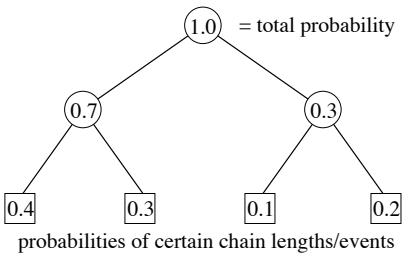


Figure 1: Sample probability tree.

molecules in the system actually undergo reaction.[iii] If there are r possible reactions, choice (1) of Δt is calculated straightforwardly with equation 11. A naive implementation of choice (2) might involve a linear scan through the probabilities, $p_0 \ldots p_{N-1}$, until we have the p_i of minimum index i such that $\sum_i p_i < \gamma_1$, for some random number γ_1. This would lead to $O(r)$ time complexity. However, an improved algorithm has been devised that makes a decision amongst r choices with $O(\log r)$ time complexity. This algorithm makes use of a binary tree data structure in which the leaves store the probabilities, $p_0 \ldots p_{r-1}$, of each choice, each internal node stores the summed probabilities of both of its subtrees and the root node contains the total probability. The number of leaves is always 2^k where $k = 0, 1, 2 \ldots$ and is as small as possible such that the data structure can accommodate the total number of choices that might be available, i.e. $2^k \geq r$. Any number of the leaves may however carry a value of zero. For choices with probabilities 0.1, 0.2, 0.3 and 0.4, an example tree, T, is given in Figure 1.

Given a random probability $P \in$(0,1), we can pick an event/chain length corresponding to this probability from tree T using the following algorithm represented as pseudo-code.

```
set t = root(T), p = P
while t is not a leaf
    if p <= p(t →left), then
        t := t →left
    else
        p := p − p(t →left)
        t := t →right
repeat
```

[iii] Note that choices 2 and 3 can in principal be considered as a single choice, however due to the current implementation it make sense here to consider them separately.

This tree based decision making data structure and algorithm has been used in implementation of reaction choice and chain length choice (although for the latter decision, this was not found to be the most appropriate algorithm).

The "probabilities" are not required to be in the range (0,$p_{\max} = 1$]; $p_{\max}$ can instead be any positive number as long as one can generate a random number in the same range (0,$p_{\max}$]. Thus, in our implementation, the root nodes/$p_{\max}$ for (i) the chain length choice tree and (ii) the reaction choice tree contain, respectively, (i) the total number of molecules of each particular polymeric species and (ii) the total rate of events in the system. The total number of each molecule (i, root of the chain length probability tree) is used to calculate the rate of the reactions that depend on that species and the total rate of events (ii, root of the reaction choice probability tree) is used to calculate the timestep via equation 11.

The use of a probability tree to select an individual reaction requires special consideration. After each reaction i occurs, the quantities of the molecules which are reactants and products of reaction i will change. This change in the quantities of molecules implies that some subset of the set of all reactions will require their probabilities to be re-calculated. Specifically, the reactions which have as their reactants any of the reactant or product species of the previous reaction will require their rates to be re-calculated.

We can therefore, for a set of reactions forming a model, define a dependency function $D(i) =$ [reactions whose rates depend on the outcome of i], so that if a reaction with index i occurs, the program is only required to update the rates of the smallest necessary subset of all the reactions such that the simulation is still correct. Further, since reaction rates are stored in the leaves, all paths between the leaves returned by $D(i)$ and the root node must be updated, and some of these will be common. This principal is explained with the help of Figure 2.

Assume a reaction with index 4 occurs and our dependency function returns $D(4) = [4, 5, 7, 9, 14]$ (note that when a reaction i occurs, at a very minimum the reaction probability for i must be updated since the numbers of the reactants participating in reaction i have just been depleted, i.e. it is always true that $i \in D(i)$). Figure 2 shows the path between reaction probability 4 to the root node as the most bold line and the paths for the remaining reactions in $D(4)$ are also shown as (slightly less) bold lines. It can be seen that in all of the paths requiring updates, some nodes are in common. To increase efficiency further, our software anticipates when update paths will pass over common nodes

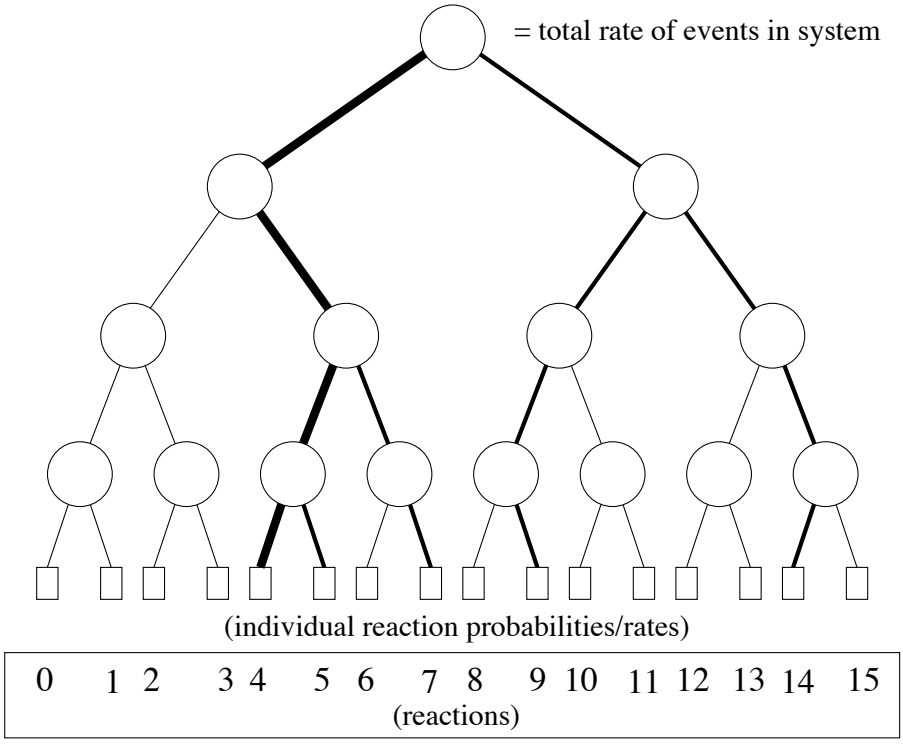


Figure 2: A schematic representation of the probability tree data structure, as used in the novel reaction choice algorithm. The leaves of the tree contain the rates of individual reactions and all internal nodes contain the sum of the numbers in their children; consequently, the root node contains the total rate of events in the system. The use of this data structure and the accompanying algorithm for reaction choice have lead to significant speed gain over the more obvious, linear scan, algorithm.

and ensures that no redundant tree arithmetic operations are performed.

3.4 On the representation of complex architecture molecules

The program is capable of simulating the kinetics of complex architecture molecules (CAMs), that is, molecules with two or more distinct chain lengths associated with them. The term CAMs has been used rather than "stars" since one can in principle simulate a star polymerisation with a single chain moiety. For example, when implementing an a arm star simulation, one could give the star species a propagation rate coefficient a times that of the single arm analogue – or, alternatively, a rate coefficient associated with radical acquisition which is a times that of the single arm species. Whilst this gives no information regarding the individual chain lengths making up the complex species, this information may not be required, especially as it is not obtainable from an experimental MWD. This method of using a unidimensional number to represent a species which does in fact posses more than one chain length has been employed by at least one other author.[14]

However, the above implementation cannot be used when species with more than a single chain length

can break apart to yield the constituent molecules since, in lumping the length parameters together as their sum, their individual information is lost. Such a situation exists in (at least) two polymerising systems. (1) In the reversible addition fragmentation chain transfer (RAFT) process,[21,22] where the radical intermediate P_n–RAFT(•)–P_m splits apart to yield $P^{\bullet}_{m,n}$ and $P_{n,m}$–RAFT. (2) In RAFT star polymerisations known as the Z-group approach,[23–25] the arms of the star grow as free, linear chains in solution and only become arms (in the true sense) when they undergo chain transfer at the core of the star – i.e. the arms are in dynamic and free exchange in and out of the CAM.

One simple method of representing CAMs with a chain lengths/arms would be to store explicitly the a-tuple representation of every such molecule, that is, for $a = 3$ molecules, every $(i,\ j,\ k)$ that occurs in the simulated system. Whilst this method is simple and clearly correct in that it does not introduce any assumptions, for certain systems it may require a large amount of memory to store all a-tuples accurately since if, for example, the range of chain lengths for each arm is 30 and there are six arms, the minimum number of molecules requiring representation is $30^6 \approx 7.3 \cdot 10^8$. As well as requiring a large amount of memory, in most simulations, a kinetically correct simulation might only require 10^8 – 10^{10} particles, and this complete n-tuple representation might force the use of more particles than is strictly necessary.

If we consider that, for our example 3-tuples $(i,\ j,\ k)$, the i, j and k will occur according to certain probability distributions, we can represent the 3-arm molecules as a collection of three MWDs. This probabilistic representation has the advantage of requiring fewer molecules in the simulation. However, each CAM no longer exists in the true sense but rather we can calculate the probability for the existence of a particular 3-tuple. There is however a situation in which this representation will yield inaccurate results. To illustrate this, consider a system containing *only* the two arm molecules: (1,1) and (5,5). If these are compressed into the above probabilistic representation, as well as (1,1) and (5,5) which exist, the set of all possible molecules that we can form includes also (1,5) and (5,1), both of which were not present in the original set. In fact a generalisation can be made at this point: in groups of CAMs in which there is a high degree of correlation between the arm lengths of a given molecule (e.g. the first arm being short implies that the remaining arms are likely to be short), the probabilistic representation will not produce strictly correct results since the probability-distribution-per-arm implementation assumes that the length of any arm is independent of the lengths of other arms. The method of storing each CAM's constituent chain lengths separately also incurs a speed

penalty, since the selection at random of a species with a arms will require a random numbers to be generated rather than a single random number per molecule.

This is unlikely to be a problem in the simulation of physical systems in which the arms are in constant dynamic equilibrium. The only situation in which it might be a problem is if a species is formed *irreversibly* from linear chains throughout a polymerisation, where the linear chains are short in the early stages of the polymerisation but increase in length during the polymerisation. However, if the CAMs are indeed formed *irreversibly*, there is no need to preserve the individual lengths i, j and k of the constituent chains and the CAMs can be consolidated into a single chain length species. For example, if a 3-arm species $C_{i,j,k}$ is formed from a 2-arm species $A_{i,j}$ and a linear species B_k, rather than implementing the reaction $A_{i,j} + B_k \rightarrow C_{i,j,k}$ one would instead implement $A_{i,j} + B_k \rightarrow D_{i+j+k}$.

To decide on the representation that is most appropriate it is important to consider the particular systems to which the Monte Carlo method is applied. The systems which are of interest in the current research are described later and appear in Figures 4 and 5. The CAMs, denoted as $\mathbf{Q}\ldots(i,\ j)$ in those figures, are not the most important or numerous species. Indeed, these species are intermediates in the kinetic models and in most cases only occur in numbers less than circa 10 for a 10^{10} particle system. They do, however, take part in very fast and hence frequently occuring reactions and it is therefore useful to make their simulation as efficient as possible. For these reasons an explicit representation of the CAMs has been chosen.

3.5 Haskell: Optimal C code generation

Haskell[26] is a lesser-known, high level, general purpose programming language developed with a view to writing code exhibiting brevity and clarity. Debugging time is minimised through the use of strict data types and the programmer does not implement an algorithm as a series of instructions, but rather writes a series of expressions which ultimately define the outcome of the algorithm, relying on the compiler to generate executables. Haskell, whilst used for prototyping in the current research, is too slow to be used in performance intensive calculations however. Its role in the final program was to take as input a reaction model and generate C code specialised for that particular problem. The compiler can then optimise this code to a far higher degree than it could generic C code since more data is known at compile time. Using code that is specific to a particular problem has given us a significant speed advantage. The method bears similarity to that he Fastest Fourier Transform in

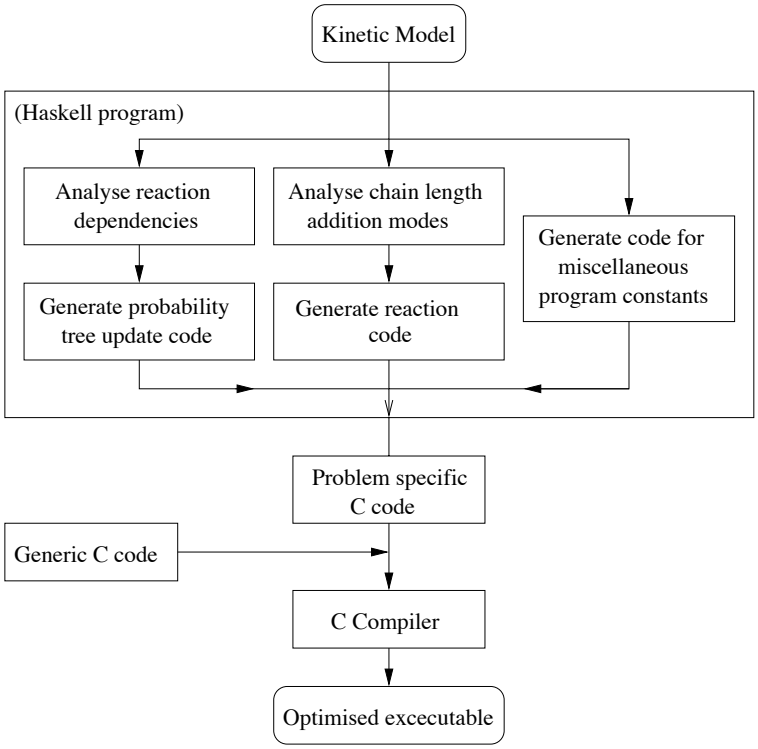


Figure 3: Flow chart depicting overall program architecture and the flow of information from the kinetic model to an optimised executable.

the West (FFTW) algorithm which achieves portability and speed by generating code for a specific architecture.[27]

Our code generation method as well as Haskell's role as a prototyping language is described in a companion publication,[28] however a brief summary will be given with the help of Figure 3. The kinetic model (reaction steps and rate coefficients) is specified by the researcher. The Haskell program then performs three tasks. (1) The reaction dependencies are analysed and a block of C code is generated for each possible reaction. This block of C code performs the minimum operations to update the reaction choice tree in light of the molecules whose numbers will have undergone changes do to each possible reaction. (2) For each possible reaction, a block of C code is generated – this performs the necessary chain length arithmetic depending on whether the reaction was propagation, combination, etc. (3) Miscellaneous program constants for the initial reagent quantities, rate coefficients, indices into various data structures, etc. are generated. The interested reader can find some examples of the generated C code in the supporting information section.

This specialised code is placed into a header file which is included into the rest of the simulation code, the conglomeration of which is compiled into a highly optimised executable which can be distributed across multiple compute nodes.

4 Experimental

4.1 Computer hardware and software

The simulation program has been implemented in ANSI C with system specific C code generated by a program written in Haskell.[26] The C code was compiled using the Intel C++ Compiler (ICC) 9.1.042. Haskell code was compiled using the Glasgow Haskell Compiler (GHC) 6.4.2. Inter-process communication was implemented using a Message Passing Interface (MPI2)[29] (MPICH2-1.0 implementation). The compiled binaries were run on either (a) a cluster comprised of eight compute nodes each with a single 3.2 GHz Intel Processor with Hyper-Threading Technology, connected via dedicated GigaBit Ethernet interfaces or (b) a shared memory, 8 processor Advanced Micro Devices (AMD) Opteron 870 machine.

PREDICI simulations were initially carried out with version 6.36.1 of the program, however, after correspondence with the authors (CiT, Computing in Technology, GmbH), an improved version (6.36.2) was provided. These were run on a Hyper-Threaded 3400 MHz Intel(R) Xeon(TM) machine. PREDICI runs as two threads, however, during simulation, these use only the execution rescources of a single processor and is a serial program.

For further details relating to code compilation and operating system/kernel versions see the supporting information section.

4.2 Test systems

Three kinetic models were used to generate the data and benchmarks that appears in this publication. The first was very close to the RAFT methyl acrylate (MA) model described by Drache et al.[4] and is very similar (in terms of the types of reactions) to the simple test system with the inclusion of a cross-termination reaction ($\mathbf{Q}(i,\, j) + P(k) \rightarrow D(i + j + k)$). The other test models are depicted in Figures 4 and 5 and respectively represent (a) a simple styrene RAFT polymerisation at 333 K and (b) a complex RAFT model of styrene at 353 K that has been used my some of the authors previously to simulate star polymerisations.[7] These models (a and b) are referred to in the rest of this document as the simple and complex test models. No chain length dependency of rate coefficients was implemented.

Decomposition/Initiation

$$\begin{array}{rcl} \mathrm{I} & \rightarrow & 2\mathrm{I}\bullet \qquad k_\mathrm{d} \\ \mathrm{I}\bullet + \mathrm{M} & \rightarrow & \mathrm{P}(1) \qquad k_\mathrm{i} \end{array}$$

Pre-equilibrium

$$\mathrm{RAFT\text{-}R} + \mathrm{P}(n) \rightarrow \mathrm{RAFT\text{-}P}(n) + \mathrm{R} \quad k_\mathrm{tr}$$

Propagation

$$\mathrm{P}(n) + \mathrm{M} \rightarrow \mathrm{P}(n+1) \qquad k_\mathrm{p}$$

Termination

$$\mathrm{P}(i) + \mathrm{P}(j) \rightarrow \mathrm{D}(i+j) \qquad k_\mathrm{t}$$

Main equilibrium

$$\begin{array}{rcccl} \mathrm{P}(i) + \mathrm{RAFT\text{-}P}(j) & \rightarrow & \mathbf{Q}(i,\,j) & & k_{-\beta} \\ \mathbf{Q}(i,\,j) & \rightarrow & \mathrm{P}(i) + \mathrm{RAFT\text{-}P}(j) & & k_\beta \\ \mathbf{Q}(i,\,j) & \rightarrow & \mathrm{P\text{-}RAFT}(i) + \mathrm{P}(j) & & k_\beta \end{array}$$

Figure 4: Simple test system corresponding to the kinetic model of a RAFT polymerisation. The model was parameterised for the polymerisation of bulk styrene at 333 K initiated with azobisisobutyronitrile. The right most column contains the rate coefficient; see the supporting information section for their values.

5 System behaviour and implementation discussion

5.1 Effect of the number of particles

The number of particles, N, must be chosen to be as small as possible such that the simulation is accurate, that is, there are sufficient particles to accurately represent the species which is lowest in concentration and keep it's time average concentration identical to a real chemical experiment. It is desirable to choose the lowest possible such N that meets these requirements, since the timestep Δt given in equation 11, upon substitution of equations 9, 8 and 10, yields

$$\Delta t \propto N^{-1}$$

Therefore, as we increase the number of particles, the timestep Δt decreases and a greater number of *Monte Carlo steps* are required to reach the desired system time/conversion. For example, by increasing the number of particles by a factor of 10, the speed of the simulation slows down by approximately the same factor. Typically, the number of particles in the system is between 10^8 and 10^{10} with the latter taking 100 times as long as the former to simulate. It is useful therefore to gain insight into how the number of particles affects the simulation output.

Decomposition/Initiation

$$
\begin{array}{lrcl}
0: & \mathrm{I} & \rightarrow & 2\mathrm{I}\bullet \qquad f \cdot k_\mathrm{d} \\
1: & \mathrm{I} & \rightarrow & 2\mathrm{Nothing} \qquad (1-f) \cdot k_\mathrm{d} \\
2: & \mathrm{I}\bullet + \mathrm{M} & \rightarrow & \mathrm{P}(1) \qquad k_\mathrm{i} \\
3: & \mathrm{R} + \mathrm{M} & \rightarrow & \mathrm{A}(1) \qquad k_\mathrm{i}
\end{array}
$$

Propagation

$$
\begin{array}{lrcl}
4: & \mathrm{P}(n) + \mathrm{M} & \rightarrow & \mathrm{P}(n+1) \qquad k_\mathrm{p} \\
5: & \mathrm{A}(n) + \mathrm{M} & \rightarrow & \mathrm{A}(n+1) \qquad k_\mathrm{p}
\end{array}
$$

Pre-equilibrium

$$
\begin{array}{lrcl}
6: & \text{RAFT-R} + \mathrm{P}(i) & \rightarrow & \text{QpreChain}(i) \qquad k_{\beta,1} \\
7: & \text{RAFT-R} + \mathrm{A}(i) & \rightarrow & \text{QpreArm}(i) \qquad k_{\beta,1} \\
8: & \text{QpreChain}(i) & \rightarrow & \text{RAFT-R} + \mathrm{P}(i) \qquad k_{\beta,1} \\
9: & \text{QpreChain}(i) & \rightarrow & \text{RAFT-P}(i) + \mathrm{R} \qquad k_{\beta,1} \\
10: & \text{QpreArm}(i) & \rightarrow & \text{RAFT-R} + \mathrm{P}(i) \qquad k_{\beta,1} \\
11: & \text{QpreArm}(i) & \rightarrow & \text{RAFT-A}(i) + \mathrm{R} \qquad k_{\beta,1}
\end{array}
$$

Termination

$$
\begin{array}{lrcl}
12: & \mathrm{A}(i) + \mathrm{A}(j) & \rightarrow & \text{A-coupled}(i+j) \qquad k_{\mathrm{t,A,A}} \\
13: & \mathrm{P}(i) + \mathrm{A}(j) & \rightarrow & \text{Adead}(i+j) \qquad k_{\mathrm{t,A}} \\
14: & \mathrm{P}(i) + \mathrm{P}(j) & \rightarrow & \mathrm{D}(i+j) \qquad k_\mathrm{t}
\end{array}
$$

Main RAFT equilibrium

$$
\begin{array}{lrcl}
15: & \mathrm{P}(i) + \text{RAFT-P}(j) & \rightarrow & \mathbf{Q}(i,\,j) \qquad k_{-\beta} \\
16: & \mathbf{Q}(i,\,j) & \rightarrow & \mathrm{P}(i) + \text{RAFT-P}(j) \qquad k_\beta \\
17: & \mathbf{Q}(i,\,j) & \rightarrow & \text{P-RAFT}(i) + \mathrm{P}(j) \qquad k_\beta \\
 & & & \\
18: & \text{RAFT-P}(i) + \mathrm{A}(j) & \rightarrow & \mathbf{Qstar}(i,\,j) \qquad k_{-\beta} \\
19: & \mathrm{P}(i) + \text{RAFT-A}(j) & \rightarrow & \mathbf{Qstar}(i,\,j) \qquad k_{-\beta} \\
20: & \mathbf{Qstar}(i,\,j) & \rightarrow & \mathrm{P}(i) + \text{RAFT-A}(j) \qquad k_\beta \\
21: & \mathbf{Qstar}(i,\,j) & \rightarrow & \text{RAFT-P}(i) + \mathrm{A}(j) \qquad k_\beta \\
 & & & \\
22: & \mathrm{A}(i) + \text{RAFT-A}(j) & \rightarrow & \mathbf{QstarStar}(i,\,j) \qquad k_{-\beta} \\
23: & \mathbf{QstarStar}(i,\,j) & \rightarrow & \mathrm{A}(i) + \text{RAFT-A}(j) \qquad k_\beta \\
24: & \mathbf{QstarStar}(i,\,j) & \rightarrow & \text{RAFT-A}(i) + \mathrm{A}(j) \qquad k_\beta
\end{array}
$$

Figure 5: Complex test system corresponding to the kinetic model of a RAFT polymerisation with explicitly modelled R-group bearing polymer. The left most column contains a reaction index and the right most column the rate coefficient. The model was parameterised for the bulk polymerisation of styrene at 353 K initiated with azobisisobutyronitrile. See the supporting information section for all rate coefficients. This model has been used by the authors in the simulation of complex architecture polymerisation kinetics.[9]

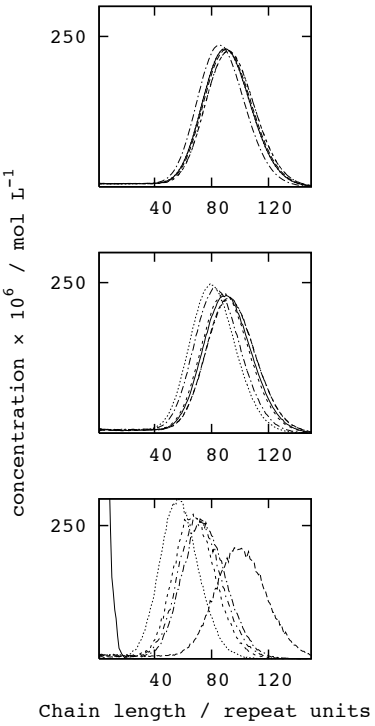


Figure 6: Effect of the number of parameters on the output from simulation of the simple test system at 5×10^4 s (from top to bottom 10^{10}, 10^9 and 10^8 particles). The primary effect of a reduction in the number of particles is a reduction in the reproducibility for different series of random numbers.

Early in the current research, it was noticed that when running simulations to constant time, a reduction in the number of particles, N, in the simple test system (Figure 4) reduced slightly the quality of the resulting MWDs, although this was *not* the most significant effect on the accuracy. Rather, as the number of particles was decreased, the largest effect was, for different sequences of random numbers, for a fixed simulated polymerisation time, to introduce statistical scatter into the position of the main MWD peak and also into the conversion of monomer. More specifically, different sequences of random numbers lead to different results of the simulation. Figure 6 depicts the simulation output for simulations with values of N of 10^8, 10^9 and 10^{10}. By examining this figure it can be seen that $N = 10^{10}$ particles gives almost perfectly reproducible behaviour, 10^9 leads to some scatter and for 10^8 particles there is significant deterioration in the reproducibility.

For some systems, a number of particles as low as 10^8 might be sufficient to produce a sufficiently accurate results in terms of the smoothness of the distribution. Conversely, a small number of particles might lead to noisy MWDs, but be able to accurately represent the kinetics of the system as far concentration and average molecular weights. For example the complex test system, when simulated to 35% conversion using 10^9 particles, produces almost identical output to a simulation using 10^{10}

particles if a smoothing function is applied to the simulation output, or, the system size is scaled by a factor of 10 towards the end point of a simulation. It is important to consider, therefore, why a smaller number of particles creates differences in the position of the MWD in the simple test system and how it might otherwise cause mis-representation of the kinetics.

When looking at the system time, t, and how it advances as a function of the number of reaction steps, it was discovered that in systems with a small number of particles, t is advanced by one of two possible processes: either there is (a) significant activity in the system due to the presence of a radical, a fast rate of events and a small timestep or (b) no radical present, the only event is decomposition of an initiator molecule, a slow rate of events and therefore a large timestep. In systems with a small number of particles, it is this latter event that is mostly responsible for the evolution of the time of the system. More specifically, the time average number of radicals in a system with a small number of particles is actually less that unity. Therefore, since nearly all events require the presence of a radical in the system, systems with small numbers of particles can reach a state where the only possible event is initiator decomposition. This creates a scenario where a relatively small number of random timestep increments carries a large contribution to the time evolution.

We here define the term large timestep event (LTSE) to denote an event occurring in a system which, at the time of the event, has a small rate. Initiator decomposition events, when there are no other possible reaction pathways, is an example of a LTSE. We can further define the quantity *fraction of time evolution due to LTSEs*. Figure 7 displays the fraction of time evolved by large time-step events (LTSEs) for the simple test system (Figure 6) for both 10^8 and 10^9 particles and it can be seen that compared to the 10^9 particle system, the 10^8 particle system has a significant lack of reproducibility in its fraction of time evolution due to LTSEs, which fluctuates in the range 0.007 to nearly 1 for 40 simulation runs.

It seems likely that the least correctly represented subset of an entire system are those events that are least probable and those species which are lowest in concentration, since it is these events that are sampled most rarely and therefore experience the greatest statistical scatter. The least probable event in the simple test system is initiator decomposition and the least numerous species is the propagating radical, and these together are the cause of irreproducibility when the number of particles is reduced.

Further, recall that the timestep is calculated from $\Delta t = -\log_e(\gamma)/\text{rate}$ with the random number $\gamma \in (0, 1)$. The function $y = -\log_e(x)$, approaches infinity as x approaches 0, meaning that if the

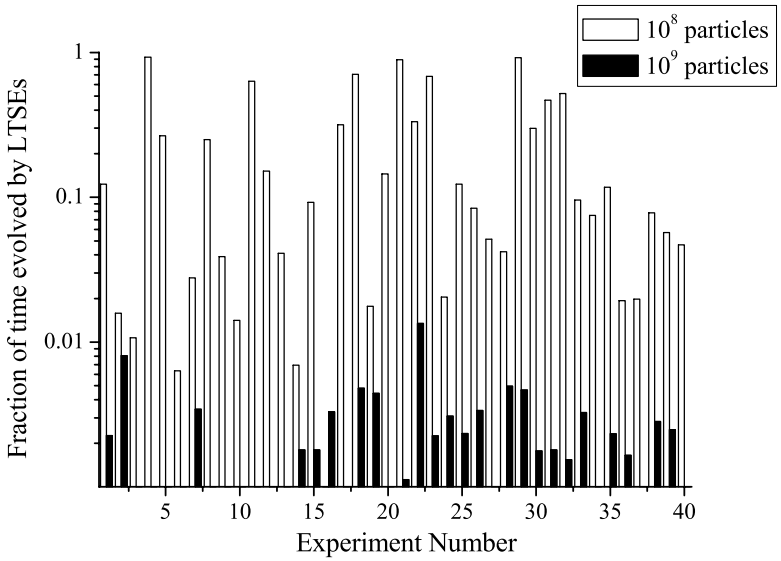


Figure 7: Fraction of time evolved by large time-step events (LTSEs) for the simple test system (Figure 6) in 40 simulation runs for both 10^8 and 10^9 particles.

denominator (the rate) is small, a sum over only a small number of Δt's will not lead to consistent times. It is worth noting that there would be no complications in calculating the timestep this way if there were so many events in the lifetime of the simulated system that there were no statistical variation in the occurrence of the least probable event.

If we are interested only in the extent of conversion of monomer as our stopping criterion, a much smaller number of particles can be used, since conversion, being due to a large number of events, is a far better represented quantity than the time. The trade-off however is that whilst the shape of some output MWDs might be more reproducible, the amount of initiator decomposed at a certain conversion will be less reproducible, as will be the timestep calculation and MWDs which are directly affected by initiator decomposition.

Reproducibility and MWD quality are not however the only measure of accuracy that is affected by the number of particles. The rate of initiator decomposition as a function of conversion is also significantly affected by the number of particles. Turning our attention now to the complex test system (but to emphasise certain phenomena, using half the initial initiator concentration specified), the quantity of initiator remaining in the system has been plotted as a function of the conversion in Figure 8, displaying large, systematic differences in the initiator decomposition characteristics for varying numbers of particles. It can be seen how two systems with $N = 10^8$ and 10^9, both at

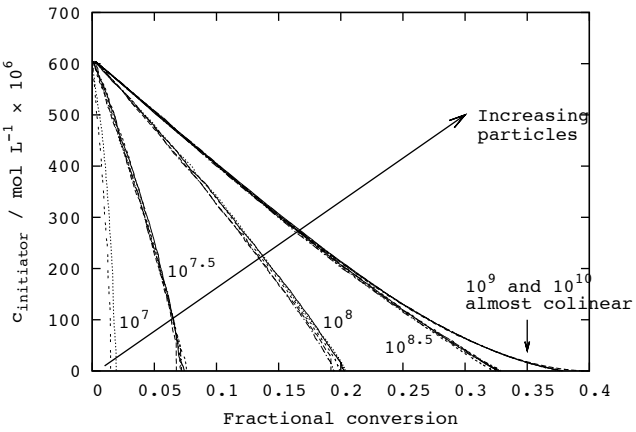


Figure 8: Initiator decomposition as a function of conversion for numbers of particles, N, in the complex test system: for the groups of curves proceeding from left to right $N = 10^7$, $10^{7.5}$, 10^8, $10^{8.5}$, 10^9 and 10^{10} (curve corresponding to 10^9 and 10^{10} are almost indistinguishable). The different curves within each group correspond to different simulations and have incomplete collinearity due to statistical scatter between runs.

10% conversion, despite both having similar MWDs (not depicted), can give significantly different behaviour. The former will run out of initiator molecules at circa 20% conversion and also will be biased by having more short, initiator fragment bearing chains in the system. It is important to note however, that the decomposition of initiator with respect to time is always identical regardless of the number of particles (data not depicted).

It is possible to gather more fine-grained information regarding the affect of the number of particles on system behaviour. Figure 9 contains the summed and normalised reaction probabilities for each reaction in the complex test model, averaged over the first 1% of conversion for various system sizes between $10^{7.5}$ and 10^{11}. As for the decomposition of initiator with respect to conversion, varying the number of particles creates large systematic changes in the reaction probabilities. First note that the reactions whose probabilities undergo the largest changes are all radical species. This is in agreement with the notion postulated above that the most rare species are the most poorly represented.

Further, as the number of particles decreases, the normalised probabilities of reactions of these rare species increase indicating that the affect of a very small system size is to force the rare species to be more numerous than they ought to be. Note in particular reaction probability 14, corresponding to the radical sink reaction P(i) + P(j) $\rightarrow$D($i + j$), which undergoes the largest change with varying N. This is the reason why the decomposition of initiator with respect to conversion (Figure 8) is too fast for small systems: radicals are removed too quickly and therefore must be re-introduced by initiator decomposition to support continual conversion of monomer to polymer.

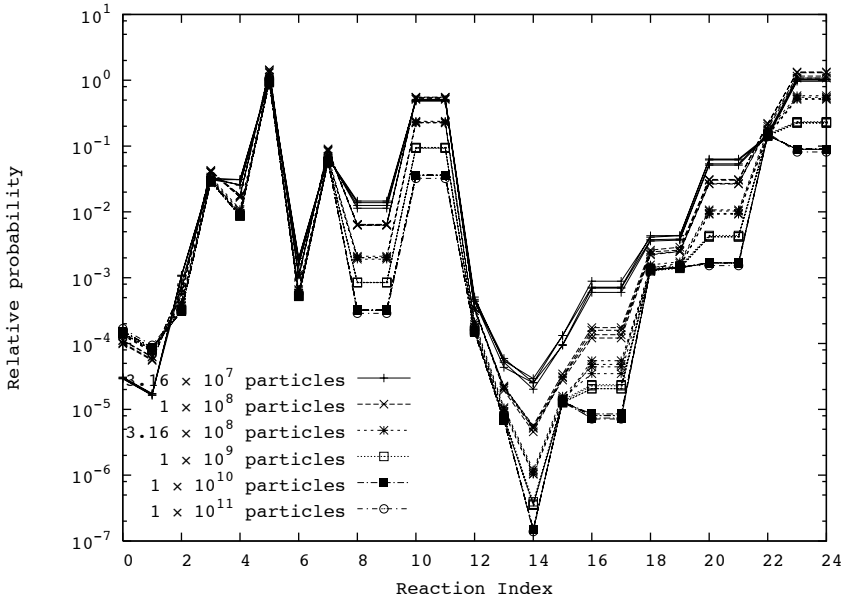


Figure 9: Relative reaction probabilities for complex test system (Figure 5) averaged over the first 1% of conversion. Data is for systems of different sizes, normalised to achieve equivalence of the probabilities of arm propagation reaction ($\mathrm{A}(n) + \mathrm{M} \rightarrow \mathrm{A}(n+1)$, reaction index 5).

5.2 Time calculation

The timestep calculation requires the generation of a random number on (0, 1), computation of its logarithm and a division operation. Depending on the complexity of each event (i.e. whether simple, polymeric or complex species are involved), this can be a significant fraction of the total simulation time.

Instead of calculating the timestep, Δt, microscopically, that is after each reaction event, the current system time, t, could be calculated macroscopically if one had a function $f : (\text{system state}) \rightarrow t$ that delivered the system time by inspection of the system state. With this method it would only be necessary to calculate the time when one wished to know it and, assuming this would only be every million or so events, would incur almost no computational burden.

Assuming the system has sufficient particles that the simulation has a high degree of accuracy, the simplest method is to use the number of particles of any species for which there exists an analytical solution. In the current systems, initiator, which undergoes exponential decay, is an obvious choice ($t = f(n_{\text{initiator}}) = -\log_e(n_{\text{initiator}}/n_{\text{initiator, 0}})/k_{\text{d}}$). If the decay of initiator is poorly represented, it is possible to solve the moments equations using an ODE solver and use the time dependent concentration of any species whose concentration is monotonic increasing/decreasing.

Both the above method has been used to gain a speed improvement of up to 15% for simulation of the

simple test system, however due to the lack of generality, this optimisation has not been used in the benchmarking section of the manuscript.

5.3 Parallelisation

If the computer system over which the program is to be distributed consists of p compute nodes, the parallelisation is performed by dividing the N particles approximately evenly amongst the p nodes. The mini-system allocated to each node is then reacted in isolation until a synchronisation step occurs, during which the nodes communicate their state, then recommence computation, each with its new local state coming from the average of the global state. This method of dividing the system into parts has the physical analogy of a chemical system, which is (i) divided into smaller parts, each of which are (ii) reacted in isolation for a brief period prior to (iii) stirring to achieve homogeneity before returning to step (i).

There are several design choices to be made relating to the synchronisation. These are (a) how often synchronisation occurs and (b) whether synchronisation occurs when the nodes reach (i) the same simulated time, (ii) the same number of reaction events or (iii) some other equivalence criterion. Methods (i) and (ii) are the only two that have been explored in the current research.

During communication of the local system state, the nodes are unable to continue simulating and a certain amount of non-simulatory computation is required to perform a communication step. It is desirable, therefore, that the communication step occurs as infrequently as possible such that accuracy is still maintained.

Synchronising nodes when they reach the same time has the appeal of realism, since in real life, sub-parts of a larger physical system must experience the same speed of passage of time. However, if different sub-systems require a different number of reaction steps (and hence computing time) to reach a certain physical time, there will be periods where some nodes will be waiting for others to finish, leading to less than maximum utilisation of the computing power available. This will especially be the case if the simulated time is calculated microscopically, since there will be statistical scatter in the number of events required by different systems to reach a certain time.

Synchronising when each node has performed a pre-determined number of reaction events has the advantage that the times taken by each of the compute nodes will be essentially identical (assuming

that each node has the same computing power), meaning that the chance of some nodes having to wait for others to finish will be small. However this method lacks a degree of realism since we are in effect forcing each system to undergo the same number of events whereas in a physical system, not all sub-parts would necessarily experience the same rate of reaction events.

The communication was implemented using a message passing interface (MPI).[29] After each node reaches the synchronisation point, it packages its state data (MWDs, concentrations, times and any other information such as error checks) into a contiguous piece of memory (see supporting information for structure) and then communicates this using the `MPI_All_reduce()` interface function. This function, when called by every compute node in the system, transmits a packet of data (in this case state information) to all other nodes, performs some operation (in this case concentration averaging) on all received packets then returns, at which point, it is guaranteed that all nodes have undergone an identical reduce operation and have identical data.

Thus, after the nodes have communicated and added their states together, each node has an identical copy of the global state. Each of the p nodes then takes $\frac{1}{p}$th of each species and rebuilds all MWD and reaction probability tree data structures. The times of the nodes are also averaged, if using a microscopic time-step calculation, or updated from a macroscopic time calculation at this point. Also at this point, the system conducts a monomer audit, ensuring that the number of remaining molecules and the number of polymerised molecules is equal to the initial number of monomer molecules in the system. This is performed both within individual compute nodes as well as for the global system. Whilst not a requirement as such, it is mentioned since it provides a rigorous test of the correctness of the implementation and has been extremely useful for debugging the simulation itself and the communication code.

6 Performance and benchmarking

6.1 Correctness and implementation validation

Figure 10 contains a number of simulation results for 10^9 and 10^{10} particles for various synchronisation intervals for 4, 8, 12 and 16 compute nodes.[iv] Each graph also contains points which depict the

[iv] These simulations were run on the networked cluster which had only 8 compute nodes and therefore, running > 8 processes results in at least some of these processes not having the full computing power of that node available. However,

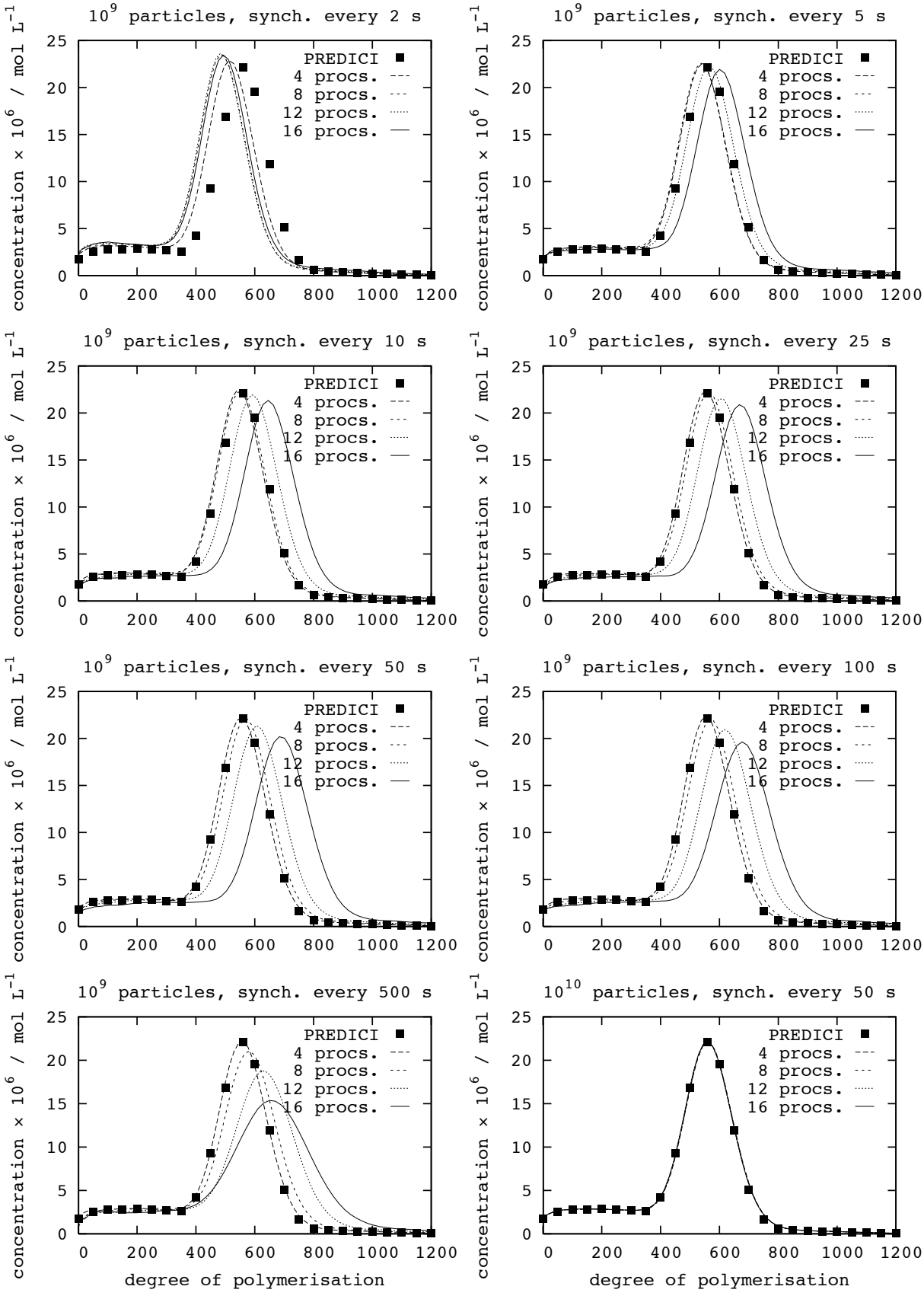


Figure 10: Simulated molecular weight distributions resulting from the complex test system at 10000 s, ≈35% conversion, distributed into 4, 8, 12 and 16 compute nodes that underwent synchronisation after a fixed numbers of seconds (solid lines) and PREDICI output for the same test system (points). Proceeding through the figure as one would read a document, the first 7 graphs correspond to simulations using 10^9 particles with the 8th graph corresponding to 10^{10} particles. See graph titles for the number of seconds between synchronisation.

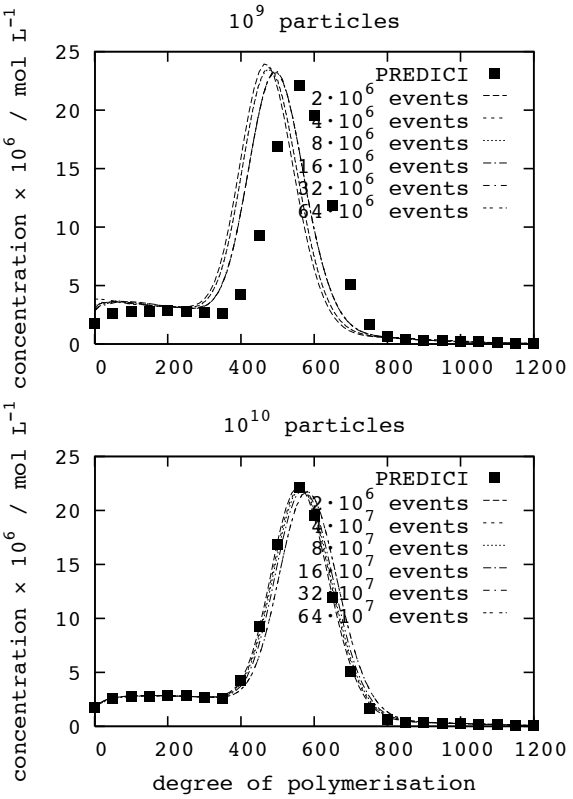


Figure 11: Simulated molecular weight distributions resulting from the complex test system at 10000 s, ≈35% conversion, distributed over 8 compute nodes that underwent synchronisation after a fixed number of events (solid lines) and PREDICI® output for the same test system (points).

result of a PREDICI simulation of the same system – these can be considered to be correct.[v]

Proceeding through the figure as one would read a document, the first 7 graphs correspond to 10^9 particles with synchronisations every 2, 5, 10, 25, 50, 100 and 500 seconds and the 8th graph corresponds to 10^{10} particles with synchronisations every 50 seconds. When there are long gaps between synchronisation, the simulations with only 10^9 particles display divergence from the correct result for large numbers of compute nodes (usually greater than 8). Reducing the time between synchronisations for these 10^9 particle systems brings about a greater degree of concordance between simulation from small and large number of compute nodes, however, for these frequently synchronised systems, the position of the MWDs shift away from the correct result. The single 10^{10} particle simulation graph in figure 10 demonstrates agreement with the PREDICI output for all numbers of processors.

Figure 11 contains simulation results of both 10^9 and 10^{10} particle simulations where synchronisation

since we are concerned here only with accuracy and not speed, this does not present a problem.

[v]PREDICI is a deterministic differential equation solver and has inbuilt estimation of integration error and step-size control based on a user specified accuracy setting. By performing successive simulations with increasing accuracy, observing convergence to a common result, and ensuring agreement between species concentrations from a full molecular weight distribution simulation and a moments simulation, it can be ensured that the simulation output is correct.

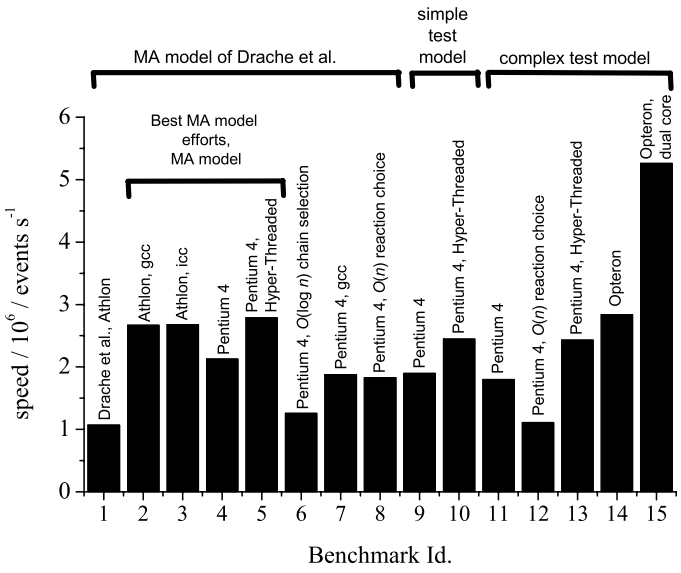


Figure 12: Speeds of various Monte Carlo implementations/versions; a higher bar indicates a better performance. The figure is annotated with a number of crucial parameters but Table 1 and the discussion in the manuscript must be consulted for complete understanding. The Pentium 4 and Athlon systems were nominally of the same speed. All Pentium 4 benchmarks used the Intel C++ Compiler (icc) unless otherwise specified.

has occurred after a fixed number of events rather than after a fixed period of time. The 10^9 particle simulations produced significant divergence from the correct result whilst the 10^{10} particle simulations only slight divergence. Both showed reproducibility for different sequences of random numbers.

In general systems with large numbers of particles are extremely robust with regard to changes in the parameters of parallelisation such as number of compute nodes, frequency of synchronisation or synchronisation criteria.

6.2 Speed and efficiency

Table 1 contains the benchmark results of various simulation codes and polymerisation models running on a variety of architectures and Figure 12 contains a graphical summary of this data. The speeds of various implementations are now discussed with the aid of the information in this figure and table. The absolute speeds reported are in events per second, since it is felt that this is the most general measure of speed. The other possible measure of speed – computer time taken to simulate a certain polymerisation time – depends on the number of particles deemed necessary for the simulation. Note

Table 1: Benchmark results for single processor operation. Figure 12 contains graphical depictions of these benchmarks.

#	1[6]	2	3	4	5	6	7
polymerisation model[1]	MA, Drache	MA, Drache	MA, Drache	MA, Drache	MA, Drache	MA, Drache	MA, Drache
architecture[2]	Athlon 64 3200+	Athlon XP 3200+	Athlon XP 3200+	P4 3200	P4 3200 HT	P4 3200	P4 3200
compiler[3]	?	gcc	icc	icc	icc	icc	gcc
reaction choice[4]	?	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
chain choice[5]	$O(1)$?	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(\log c_{\max})$	$O(1)$
speed / 10^6·events s^{-1}	1.07	2.67	2.68	2.13	2.79	1.26	1.88

#	8	9	10	11	12	13	14	15
polymerisation model	MA, Drache	simple	simple	complex	complex	complex	complex	complex
architecture	P4 3200	P4 3200	P4 3200 HT	P4 3200	P4 3200	P4 3200 HT	Opteron	Opteron DC
compiler	icc	icc	icc	icc	icc	icc	gcc	gcc
reaction choice	$O(n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
chain choice	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
speed / 10^6·events s^{-1}	1.83	1.90	2.45	1.80	1.11	2.43	2.84	5.26

[1] *MA, Drache* implies the model reported by Drache et al.[4] and is very similar to that depicted in Figure 4, *simple* implies the model depicted in Figure 4 and complex implies the model depicted in Figure 5. [2] *Athlon* and *Opteron* refer to microprocessor series produced by Advanced Micro Devices (AMD); *P4* implies Intel Pentium 4; *HT* implies that Hyper-Threading Technology was utilised; *DC* implies that both cores of the microprocessor were utilised. [3] *gcc* and *icc* refer respectively to the GNU Compiler Collection and the Intel C++ Compiler. [4] $O(\log n)$ and $O(n)$ imply respectively the binary tree and linear scan reaction election algorithms. [5] $O(1)$ implies an explicit representation of all molecules allowing a molecule to be picked at random; $O(\log c_{\max})$ implies a compressed representation of polymeric molecules, requiring a binary tree algorithm for selection of a particular molecule at random. [6] The information in this column comes from the research by Drache et al.[4] It is not possible to infer their reaction choice algorithm or the compiler used, however it seems likely, due to the large memory usage they report and their description, that an explicit state representation and an $O(1)$ algorithm were employed.

that it was not desired to compare the performance of different microprocessors, only the behaviour of different implementation strategies.

For comparison purposes, we have, to the best of our knowledge, reproduced the methyl acrylate (MA) model reported by Drache et al.[4] and benchmark 1 corresponds to the running statistics reported in that manuscript. Benchmark 2 corresponds to the current implementation running on an almost identical system and it can be seen that the current implementation runs approximately 2.5 times as fast. Whilst the exact details of the hardware that the other authors' code was run on is unavailable, it is felt that the difference in performance and the similarity of the hardware are sufficient to demonstrate that our optimisations have had a favourable affect on performance.

Comparing 2 with 3 (the GNU Compiler Collection [GCC] v.s. the Intel C++ Compiler [ICC]), it can be seen that for this particular AMD microprocessor, there is a negligible speed improvement brought about by the use of the Intel compiler. Comparing 3 and 4 (AMD v.s. Intel), it can be seen that for single threaded operation, the AMD system performs 26% faster than the nominally equivalent Intel system. However, running two threads on the Intel processor (benchmark 5) reveals a significant speed improvement over single threaded operation, clearly making use of the Hyper-Threaded Technology features of the microprocessor.

Comparing benchmarks 4 and 6 in which the chain selection changes from the explicit, large memory usage, $O(1)$ time complexity algorithm to the binary tree, small memory, $O(\log c_{\max})$ time complexity algorithm it can be seen than the latter induces a significant degradation in performance. Comparing benchmarks 4 and 7 (ICC v.s. GCC) it can be seen that for code run on an Intel processor, use of the Intel compiler leads to a significant performance gain over the GNU compiler. Note that this is in contrast to what was observed for the code run on an AMD processor (benchmarks 2 and 3, discussed above), where the choice of compiler made only a negligible difference to the speed.

Benchmarks 4 and 8 demonstrate the difference for the MA model between a tree based, $O(\log n)$ and a linear, $O(n)$ reaction selection algorithms. The former is seen to lead to an overall increase in performance of 16%. Consider now benchmarks 11 and 12 which correspond to the complex test system but which differ in the same way as benchmarks 4 and 8 with respect to their reaction choice algorithm. For this complex model, the performance gain for the use of the $O(\log n)$ algorithm is far more dramatic, with a 62% speed improvement.

Four sets of benchmarks compare the performance of dual- versus single-thread execution for both

Hyper-Threaded and Dual Core systems. Looking at the comparisons relating to Hyper-Threading – benchmarks (4,5), (9,10) and (11,13) – there are performance gains of 30 – 35%. The speed increase resulting from dual core operation is, as expected, more favourable, with utilisation of the second core resulting in an 85% speed increase. The reason for there not being a 100% speed increase is likely to be caused by the fact that (a) both cores share the same cache and/or (b) both cores share the same access pathway to the main memory and there is insufficient bandwidth to support the needs of both processes.

It is surprising how much the speed of the code benefits from running two threads in parallel on a single Hyper-Threading Pentium 4 microprocessor (30 – 35%), since this is just above the expected maximum of 30% claimed by Intel.[30] The important feature of a Hyper-Threading (formally referred to by Intel as their trademark Hyper-Threading Technology) microprocessor is that it contains the necessary resources to hold the running state information (or execution context) of two processes/threads simultaneously, and thus appears to the operating system as if two processors are in fact present. There are not, however, two real processors present: the Hyper-Threading achieves a speed increase by allowing one process/thread to continue to use the free execution resources in the processor while the other process/thread is stalled or not maximising its use of the execution resources. This might be because, for example, it is waiting for data from the main memory (termed a cache miss). As an aside, it was noted that running three threads in parallel on a Hyper-Threaded processor did not provide further speed increase.

Benchmarks 4, 9 and 11 correspond to the MA model, the simple test model and the complex test model, respectively – all other parameters are held constant. The speed of these three benchmarks differ because the complexity of/time required by individual events depends on (a) the size of the reaction tree and (b) the complexity of the reactions that occur in a particular system. To explain point (b) further, a combination reaction is a computationally burdensome event, since it involves choosing two molecules at random, adding their chain lengths, then adding the product molecule back into the system. An initiator decomposition event, in contrast, requires very little computational effort since lacking chain any chain length information, the initiator molecule count is simply decremented and the product populations incremented.

As was seen in the previous section (in particular Figure 11), using the number of events rather than the time as the synchronisation criterion leads to inaccuracies, especially for the 10^9 particle

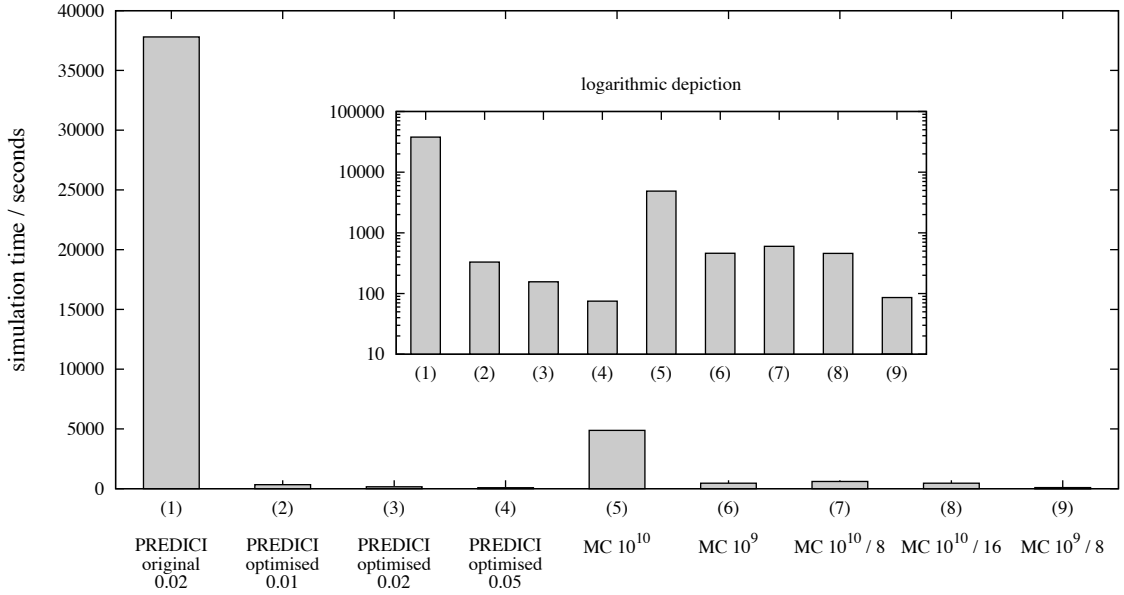


Figure 13: Speed comparison of a number of simulations of the complex test system to 10000 s. The time taken for each simulation to run is given, therefore lower is better. From left to right: (1) the original PREDICI version, prior to optimisation by the authors at CiT (Computing in Technology, GmbH); (2) – (4) optimised PREDICI with accuracies of 0.01, 0.02, 0.05; (5) – (9) all Monte Carlo benchmarks: (5) 10^{10} particles, (6) 10^{9} particles, (7) 10^{10} particles over 8 processors, (8) 10^{10} particles over 8 processors with 16 simulation threads utilising Hyper-Threading, (9) 10^{9} particles on 8 processors.

systems. Therefore, in the remainder of the tests conducted, parallel simulations used time as the synchronisation criterion, with the systems being synchronised every 50 s.

Figure 13 contains the run-times resulting from a number of simulations of the complex test system to 10000 s in both PREDICI and the current Monte Carlo implementation. Probably the most important parameter in PREDICI's numerical options for controlling the trade-off between speed and correctness is the "accuracy" parameter. This relates to the tolerable error in each integration step. A lower value of the accuracy parameter leads to a higher accuracy but often slower simulation. More information on the PREDICI benchmarking setup can be found in the supporting information section.

At the commencement of the research, the only available version of PREDICI resulted in simulation times for the complex test model of more than 8 hours. Very close to the time of publication, we entered into a collaboration with the company that produces PREDICI (CiT - Computing in Technology, GmbH) and the complex test system used in this research revealed in PREDICI's implementation of the hp-Galerkin numerical method a subtle yet important assumption which manifested itself dramatically for certain types of systems. CiT was able to make a slight yet significant modification to the algorithm implemented in PREDICI and achieve much faster simulation times with PREDICI than

were originally observed by us. It can be seen in Figure 13 that even for the 10^{10} particle Monte Carlo simulation, by distribution into 16 sub-systems over 8 processors, an almost two orders of magnitude speed increase is achieved over the original PREDICI simulation, however, the best Monte Carlo efforts are now similar to the best PREDICI efforts.

Despite the fact that a single processor PREDICI simulation now produces similar speeds to an 8 processor Monte Carlo simulation, the Monte Carlo method is still the more advantageous method in certain instances, such as when one wants to understand copolymer composition and reactions of polymer molecules with more than a single chain length associated with them. This is not possible under PREDICI / the hp-Galerkin method. A combined PREDICI/parallel Monte-Carlo approach will combine the individual strengths of each method.

It is interesting to examine how efficiently a parallel computing resource is utilised when the Monte Carlo algorithm is distributed. One measure of the efficiency is the relative speed increase (RSI). If complete efficiency were obtained, the relative speed increase would be RSI$(p) = p$ where p is the number of compute nodes.

Figure 14 contains RSI(p) for four test scenarios: 10^9 particles with network communication, 10^{10} particles with network communication, 10^9 particles with shared memory communication and 10^{10} particles with shared memory communication. Tests using network communication were run on a cluster of computers and those using shared memory communication were run on a single, 8 processor machine.

It can be seen that for simulations using 10^9 particles, the simulation is unable to utilise the total computing power available. In contrast, the 10^{10} particle simulations enjoy almost complete utilisation of the available computing power (even though the 10^9 particle system requires less time for simulation). The reason for the loss of efficiency in both systems is that there is statistical scatter in the number of events each sub-system is required to simulate in order to reach a particular polymerisation time, meaning that there are periods where some nodes are idle and waiting for the remaining nodes to finish. By simulating with a greater number of particles, the time required for each sub-system to reach a given time is more uniform (there is less statistical scatter) and hence more complete utilisation of the parallel computing resource. The use of shared memory as opposed to network communication does not lead to an improvement of the RSI because the bottleneck is not the communication latency or bandwidth but rather an intrinsic property of the algorithm. The reason why the relative speed

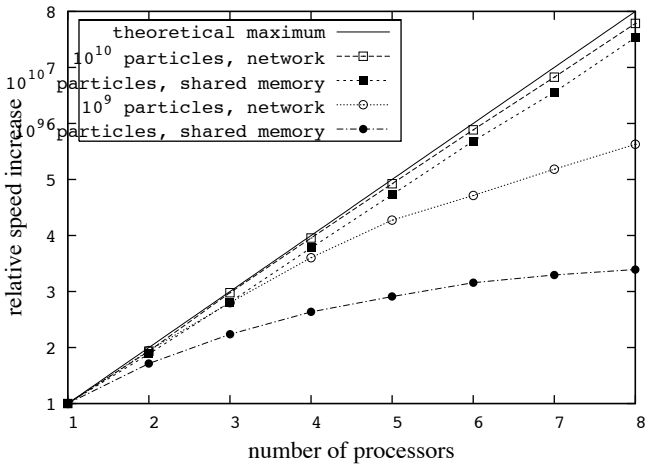


Figure 14: Relative speed increase as a function of the number of processors for simulation of the complex test system with a number of configurations: 10^9 and 10^{10} particles for both network and shared memory communication. The synchronisation criterion was the time of the simulated system with synchronisations occurring every 50 s.

increases for shared memory benchmark is poorer than that for the network benchmark is because the speed of each individual microprocessor in the shared memory machine is much greater than in the networked cluster (compare benchmark 14 with 11 in Figure 12) – it would appear that relatively speaking, the communication overhead in the system with faster processors incurs a larger overall performance cost.

In summary, with regard to distribution of the Monte Carlo algorithm over a parallel computer it can be said:

- Simulations with a smaller number of particles (circa 10^9):
  - may be accurate enough for comparison with experiments at lower conversions,
  - experience a further reduction in accuracy upon parallelisation,
  - in terms of correctness, have a reduced upward scalability for parallelisation due to reduced accuracy,
  - when synchronised after a fixed number of events (rather than simulated seconds), experience a reduction in the accuracy, and
  - experience a relatively poor utilisation of the computing power available on distributed systems.
- Simulations with a large number of particles (circa 10^{10}):

  - produce results identical to the PREDICI output over a wide range of parallelisation conditions, and
  - make excellent use of the total computing power available on distributed systems.

- In general, significant speedups were obtainable by distribution of the computation over a parallel computer.
- The less than 100% utilisation of the total computing power available when the algorithm is distributed does not arise from communication bandwidth or latency but from an intrinsic property of the way the Monte Carlo method has been paralellised.

7 Future developments

The number of possibilities for further optimisation/improvement of the current code is enormous and the following list is not exhaustive.

The parallelisation model explored in the current research is straightforward and there may be others which could be successful. For example, instead of assigning each processing unit its own mini-system to simulate, it might be possible to have 2 or more processing units share a system representation and perform reactions on the same set of molecules. Such a parallelisation model would almost certainly be best suited to hardware on which fine-grain parallelisation is possible, such as on the cores of a single multi-core processor.

Less than 100% utilisation of the total computing power available was observed when the algorithm was distributed over a parallel computer, especially for smaller numbers of particles, and attempts to improve this might be worthwhile.

Whilst the affect of the number of particles, N, on various system properties has been explored, no method exists for calculating the most appropriate value of N, and this parameter must still be chosen explicitly by the researcher who utilises the Monte Carlo approach. Since the speed of simulation is proportional to N it would be valuable to be able to determine on-line, whether the current value of N is appropriate, and then scale the system size accordingly. A method to do this might involve performing a small number of reaction events (say 10^5) for a several system sizes and testing for the value of N at which convergence of reaction probabilities occurs.

It would be interesting to apply the Monte Carlo methods developed to a greater range of systems, especially those with chain branching, copolymerisations and complex architecture molecules (CAMs) with several constituent moieties. Also, the concentration (throughout the compute cluster) of low concentration reactive species such as radicals have not been monitored as part of the study but it will be important at some point to understand how this concept influences the accuracy and system behaviour.

Finally, it seems possible that there could be benefit from using the Galerkin h-p method that is implemented in PREDICI in conjunction with Monte Carlo methods, where the error control and global molecular weight distributions come from the Galerkin h-p method and the detailed microstructural information the Monte Carlo method.

8 Conclusions

The current research has indeed lead to the desired outcome. Simulations which previously required many hours (using the commercial program package PREDICI) can now be performed in minutes. The high speed has been made possible by (a) the application of advanced methods in computer science, (b) the distribution of computing burden over multiple compute nodes and, it is conjectured, (c) the fundamental suitability of a Monte Carlo simulation method to the kinetics of the complex polymer systems which are of interest.

Acknowledgments H.C.-M. is grateful for funding from the University of New South Wales (UNSW, Sydney, Australia) and additional funding from the Faculty of Engineering, UNSW. C.B.-K. acknowledges receipt of a Discovery Grant as well as an Australian Professorial Fellowship (from the Australian Research Council). All the authors acknowledge the Faculty of Engineering, UNSW for support. We would also like to thank Dr. Michael Wulkow from CiT GmbH for the stimulating and interesting discussions that have benefited both the PREDICI and parallel Monte Carlo approach.

Supporting information The supporting information contains a number of further details relating to the research. These relate to the implementation, compilation, kinetic parameters, PREDICI

implementation details and raw profiling output.

Program code can be downloaded from `http://www.camd.unsw.edu.au/parapolysim`

References

[1] Wulkow, M. *Macromol. Theory Simul.* **1996,** *5,* 393.

[2] Prescott, S. W. *Macromolecules* **2003,** *36,* 9608.

[3] Schmidt, C. U.; Busch, M.; Lilge, D.; Wulkow, M. *Macromol. Mater. Eng.* **2005,** *290,* 404.

[4] Drache, M.; Schmidt-Naake, G.; Buback, M.; Vana, P. *Polymer* **2004,** *46,* 8483.

[5] He, J.; Zhang, H.; Yang, Y. *Macromol. Theory Simul.* **1995,** *4,* 811.

[6] Vana, P.; Davis, T. P.; Barner-Kowollik, C. *Macromol. Theory Simul.* **2002,** *11,* 823.

[7] Chaffey-Millar, H.; Busch, M.; Davis, T. P.; Stenzel, M. H.; Barner-Kowollik, C. *Macromol. Theory Simul.* **2005,** *14,* 143.

[8] Barner-Kowollik, C.; Stenzel, M. H.; Davis, T. P.; Chaffey-Millar, H. *ACS Polymer Pre-prints* **2005,** *230,* 4162.

[9] Chaffey-Millar, H.; Stenzel, M. H.; Davis, T. P.; Coote, M. L.; Barner-Kowollik, C. *Macromolecules* **2006,** *39,* 6404.

[10] Lu, J.; Zhang, H.; Yang, Y. *Macromol. Theory Simul.* **1993,** *2,* 747.

[11] Bamford, C. H.; Tompa, H. *Trans. Faraday. Soc.* **1954,** *50,* 1097.

[12] Wang, A. R.; Zhu, S. *J. Polym. Sci., Part A: Polym. Chem.* **2003,** *41,* 1553.

[13] Monteiro, M. J. *J. Polym. Sci., Part A: Polym. Chem.* **2005,** *43,* 5643.

[14] Goh, Y.-K.; Monteiro, M. J. *Macromolecules* **2006,** *39,* 4966.

[15] Battaile, C. C.; Srolocitz, D. J. *J. Appl. Phys.* **1997,** *82,* 6293.

[16] Fichthorn, K. A.; Weinberg, W. H. *J. Chem. Phys.* **1991,** *95,* 1090.

[17] Landau, D. P.; Binder, K. *A Guide to Monte Carlo Simulations in Statistical Physics;* Cambridge University Press: 2000.

[18] Jäckel, P. *Monte Carlo Methods in Finance;* John Wiley and Sons: 2002.

[19] Gillespie, D. T. *J. Phys. Chem.* **1977,** *81,* 2340.

[20] He, J.; Zhang, H.; Chen, J.; Yang, Y. *Macromolecules* **1997,** *30,* 8010.

[21] Le, T. P.; Moad, G.; Rizzardo, E.; Thang, S. H. **1998,** WO Patent 9801478.

[22] Moad, G.; Rizzardo, E.; Thang, S. H. *Aust. J. Chem.* **2006,** *59,* 669.

[23] Rizzardo, E.; Chiefari, J.; Chong, B. Y. K.; Ercole, F.; Krstina, J.; Jeffery, J.; Le, T. P. T.; Mayadunne, R. T. A.; Meijs, G. F.; Moad, C. L.; Moad, G.; Thang, S. H. *Macromol. Symp.* **1999,** *143,* 291.

[24] Stenzel, M. H.; Davis, T. P. *J. Polym. Sci., Part A: Polym. Chem.* **2002,** *40,* 4498.

[25] Barner, L.; Davis, T. P.; Stenzel, M. H.; Barner-Kowollik, C. *Macromol. Rapid. Commun.* **2007,** *28,* 539.

[26] Peyton Jones, S. *et al. J. Funct. Programming* **2003,** *13,* 0.

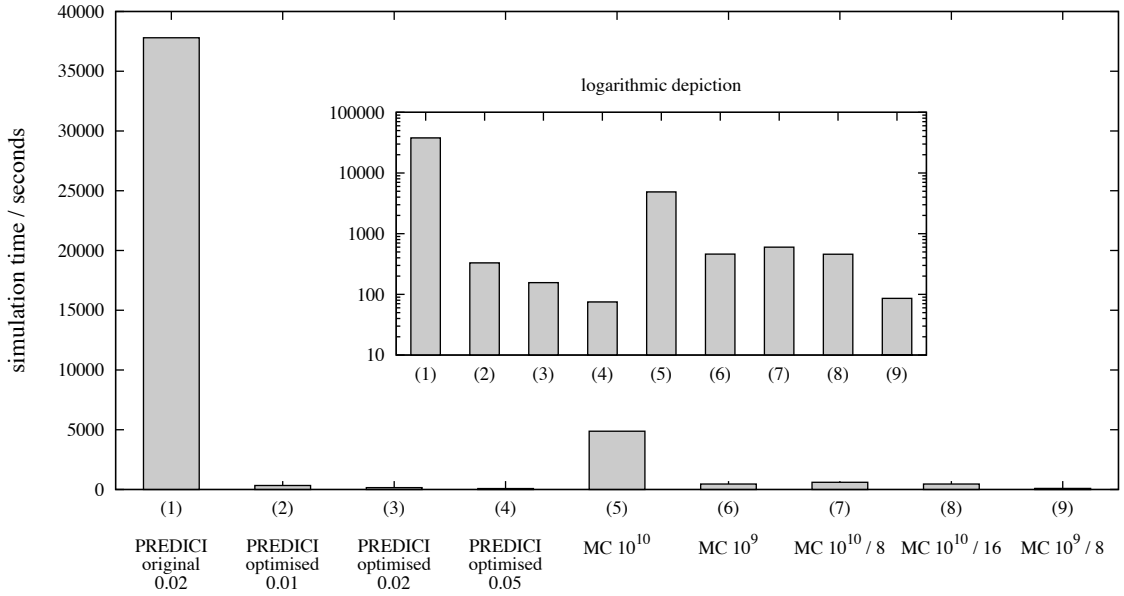
[27] Frigo, M.; Johnson, S. G. *Proceedings of the IEEE* **2005,** *93,* 216 special issue on "Program Generation, Optimization, and Platform Adaptation".

[28] Stewart, D.; Chaffey-Millar, H.; Keller, G.; Chakravarthy, M. M. T.; Barner-Kowollik, C. *Proceedings of the International Conference on Functional Programming '07* **2007,** submitted, draft available via http://www.cse.unsw.edu.au/ dons/papers/SCKCB07.html.

[29] Gropp, W.; Huss-Lederman, S.; Lumsdaine, A.; Lusk, E.; Nitzberg, B.; Saphir, W.; Snir, M. *MPI: The Complete Reference (Vol. 2);* The MIT Press: 1998.

[30] Marr, D. T.; Binns, F.; Hill, D. L.; Hinton, G.; Koufaty, D. A.; Miller, J. A.; Upton, M. *Intel Technology Journal* **2002,** *6,* 4.

Graphical table of contents entry



Speed comparison of a number of simulations of the complex test system to 10000 s. The time taken for each simulation to run is given, therefore lower is better. From left to right: (1) the original PREDICI version, prior to optimisation by the authors at CiT (Computing in Technology, GmbH); (2) – (4) optimised PREDICI with accuracies of 0.01, 0.02, 0.05; (5) – (9) all Monte Carlo benchmarks: (5) 10^{10} particles, (6) 10^{9} particles, (7) 10^{10} particles over 8 processors, (8) 10^{10} particles over 8 processors with 16 simulation threads utilising Hyper-Threading, (9) 10^{9} particles on 8 processors.