Journal of Computational Physics 363 (2018) 128–139

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

# Simulation of ring polymer melts with GPU acceleration

R.D. Schram [a,*], G.T. Barkema [b]

[a] Laboratoire de Physique et Centre Blaise Pascal, École Normale Supérieure de Lyon, CNRS UMR5672, 46 allée d'Italie, 69364 Lyon, France
[b] Department of Information and Computing Sciences, Utrecht University, Princetonplein 5, 3584 CC Utrecht, The Netherlands

ARTICLE INFO

*Article history:*
Received 22 June 2016
Received in revised form 14 September 2017
Accepted 13 February 2018
Available online 15 February 2018

*Keywords:*
GPU
Ring polymers
Monte Carlo simulations
Chromatin organization

ABSTRACT

We implemented the elastic lattice polymer model on the GPU (Graphics Processing Unit), and show that the GPU is very efficient for polymer simulations of dense polymer melts. The implementation is able to perform up to $4.1 \cdot 10^9$ Monte Carlo moves per second. Compared to our standard CPU implementation, we find an effective speed-up of a factor 92. Using this GPU implementation we studied the equilibrium properties and the dynamics of non-concatenated ring polymers in a melt of such polymers, using Rouse modes. With increasing polymer length, we found a very slow transition to compactness with a growth exponent $\nu \approx 1/3$. Numerically we find that the longest internal time scale of the polymer scales as $N^{3.1}$, with $N$ the molecular weight of the ring polymer.

## 1. Introduction

Polymers have been studied for decades, both numerically and analytically. However, there is still much to learn. Due to the nature of polymer systems, which involve large length and time scales, relatively crude models, such as Monte Carlo lattice polymer models are still an important tool. In this paper, we present a new implementation that uses the GPU (Graphics Processing Unit) to significantly shorten the time of simulations. The underlying simulation model is the elastic lattice polymer model, which has already proven its efficiency in a wide range of applications [1–3].

GPUs have been used in many other areas of statistical physics [4–6], for their efficiency in parallel computation. For instance, they have been used for research in Molecular Dynamics simulations [7,8] and Monte Carlo simulations [9]. The latter is an implementation of the bond fluctuation model (BFM). The elastic lattice polymer model has a number of advantages over the BFM, including improved ergodicity and more efficient dynamics at high density.

The algorithm presented in this paper is the most efficient for high density polymer melts. Melts of linear and ring polymers or a mix thereof are supported. The GPU implementation is many times faster than algorithms that use a CPU (Central Processing Unit). The growth of raw parallel GPU processing power far outpaces the single core processing power of CPUs. Thus, as long as we are able to sufficiently parallelize the computation, GPU computation will be increasingly important in the future.

The GPU algorithm is implemented in both the CUDA [10] and OpenCL [11] frameworks. In our description of the algorithm, we use the OpenCL terminology. For those that are familiar with GPU architecture or GPU programming in general, it should be easy to understand what the terms mean. Otherwise, the glossary of the OpenCL specification document provides an explanation of the terminology. OpenCL terms are presented in italic font.

* Corresponding author.
*E-mail addresses:* raoul.schram@ens-lyon.fr (R.D. Schram), g.t.barkema@uu.nl (G.T. Barkema).

https://doi.org/10.1016/j.jcp.2018.02.027
0021-9991/© 2018 Elsevier Inc. All rights reserved.

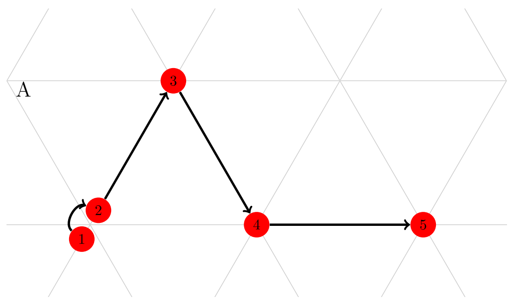

**Fig. 1.** A 2-dimensional version of the elastic lattice polymer model. Monomers 1 and 2 are on the same lattice site, and their bond is a unit of stored length. Monomer 1 is able to move to site A as an extension move. In the same manner, monomer 5 can retract to the site of monomer 4 to perform a retraction move. This move is accepted with probability 1/8, in order to maintain detailed balance. Stored length moves along the backbone of the polymer using a diffusive move, for example moving monomer 2 towards monomer 3. Monomer 2 can also move to the empty site A in a forward transverse move. The movement of monomer 3 to the site of monomer 4 constitutes a backward transverse move. In our GPU implementation, monomer 3 cannot move towards monomers 1 and 2, because a maximum of two monomers are allowed at the same site.

This paper is organized as follows. The next section introduces the elastic lattice polymer model, and notes some of the modifications that were made for the GPU implementation. Sections 3 and 4 together describe the implementation of this model on the GPU. The following section shows the performance of the GPU implementation and compares it to a singlethreaded CPU implementation.

Section 6 presents the results of a set of simulations on ring polymer melts. Part of the interest in melts of ring polymers is due to a perceived connection with large scale chromatin organization [12]. If chromosomes in the interphase would behave as ordinary linear polymers, they would mix. Experimental observations show, however, that they are not mixed: a much more realistic description is that each chromosome has its own territory. It is straightforward to find biological reasons why mixing is unwanted: cell division would otherwise become messy. At this moment, it is, however, unclear which physics/chemistry prevents the chromosomes from mixing. This might be because mixing increases the free energy, or because the mixing process is simply too slow to be occurring during the cell cycle. One possible scenario is that the chromosomes are internally crosslinked, and a crude model description for this is to describe them as non-concatenated ring polymers (linear polymers that are crosslinked to themselves at the ends). This is the approach taken in this paper.

## 2. Elastic lattice polymer model

The elastic lattice polymer model is based on the repton model by Rubinstein [13], which allows for very efficient implementations, see for example [14]. An ingredient of the elastic lattice polymer model is the notion of “stored length”. Normally, monomers are disallowed from sitting at the same lattice site; the excluded volume principle. Here, there is one exception; adjacent monomers along the polymer chain are allowed to be on top of each other. For technical reasons, in our GPU implementations the number of monomers is limited to two per site. This allows for more efficient data structures. Bonds that have zero length as a result are called units of “stored length”. In effect, this makes the chain shorter than the number of monomers in the polymer chain, and provides elasticity to polymers. The sequence of sites visited by the polymer is the backbone of the polymer. A simple move is the displacement of stored length along the backbone. In combination with a move that extends and retracts the tails of the monomer, the model provides reptation explicitly.

The lattice of choice for our model is the face centered cubic (FCC) lattice. A way to construct an FCC lattice is to take a Cartesian lattice $(x, y, z)$, and remove all sites for which $x + y + z$ is odd, then divide all distances by $\sqrt{2}$. There are 12 unit vectors in total that determine the neighbors of a site on the FCC lattice. We use four of those as basic unit vectors $\mathbf{t}$, $\mathbf{u}$, $\mathbf{v}$, and $\mathbf{w}$:

$$\hat{\mathbf{t}} = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\1\\0\end{bmatrix}, \hat{\mathbf{u}} = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\-1\\0\end{bmatrix}, \hat{\mathbf{v}} = \frac{1}{\sqrt{2}}\begin{bmatrix}-1\\0\\1\end{bmatrix}, \hat{\mathbf{w}} = \frac{1}{\sqrt{2}}\begin{bmatrix}-1\\0\\-1\end{bmatrix}. \tag{1}$$

All other unit vectors can be constructed as a sum of basic unit vectors, where we add one or zero times each basic unit vector. An advantage of using these basic unit vectors instead of Cartesian coordinates is that only 4 bits are necessary to store a unit vector (as opposed to 5 or 6 in the Cartesian system). Four out of the 16 combinations do not have length 1: $\mathbf{t} + \mathbf{u}$, $\mathbf{v} + \mathbf{w}$ and $\mathbf{t} + \mathbf{u} + \mathbf{v} + \mathbf{w} = 0$. The remaining 12 describe edges on the FCC lattice. The polymers are confined to a non-cubic box of size $L$, with periodic boundary conditions. The edges of the slanted box are parallel to the $\mathbf{t}$, $\mathbf{u}$ and $\mathbf{v}$ unit vectors. Thus, each lattice site can be represented by coordinates $(t, u, v)$, with $0 \le t < L$, $0 \le u < L$, $0 \le v < L$. In total there are $L^3$ unique lattice sites in the box.

First, monomer $i$ from polymer $j$ is selected as a candidate to move. The selection process depends on the specifics of the implementation and details will be given in their respective sections. The moves are shown in Fig. 1.

The diffusive move is the simplest move, in which stored length is moved along the backbone. The bonds $(i, i + 1)$ and $(i + 1, i + 2)$ are swapped if either of them contains stored length.

The transverse move allows the polymer to move sideways, as well as extend or retract the tail monomer. First, consider the case where monomer $i$ is not at the tail end of the polymer. There are two moves that are inverse to each other. The first is the forward transverse move, in which monomer $i$ presently co-occupies a lattice site with its neighbor (stored-length), and is subsequently moved to a neighboring empty site. The other is the backward transverse move, in which monomer $i$ moves to a site of its neighbor, leaving behind an empty site.

A forward transverse move is possible if exactly one of the bonds $(i-1, i)$ and $(i, i+1)$ is stored length. Let $\mathbf{r}$ be the non-stored-length bond unit vector. Then there are 4 combinations of unit vectors $(\mathbf{p}, \mathbf{q})$ such that $\mathbf{p}+\mathbf{q}=\mathbf{r}$, forming a triangle. Thus, instead of the current "short-cut" $\mathbf{r}$ that the polymer currently takes, we can reroute it along the unit vectors $\mathbf{p}$ and $\mathbf{q}$. One of the 4 possibilities is randomly chosen. If the new site is empty, the move is accepted. Otherwise the move is rejected.

The backward transverse move is triggered when monomer $i$ is not at the end of the polymer and has non-stored-length bonds on both sides $(\mathbf{p}, \mathbf{q})$, with the bonds defined in the same direction (from tail to head or the reverse). Then, if $\mathbf{r}=\mathbf{p}+\mathbf{q}$ is a unit vector, either bond is randomly chosen, and an attempt is made to move the monomer to that side. Since only two monomers are allowed on the same site at once, the move is rejected if the destination site already has two monomers.

In case a monomer is selected such that it is at the very tail end, an extension or retraction move is attempted. If the bond attached to the monomer $i$ is stored length an extension move is attempted. We select a neighboring site, each with probability $1/16$. Since there are only 12 neighboring sites on the FCC lattice, the move is guaranteed to be rejected with probability of at least $4/16$. If the site is empty, the monomer is moved there.

In the reverse case, where we select an end that is not connected to a stored length bond, we attempt a retraction move. Since there is only one possible way to retract a monomer, it is necessary to reject the move with probability $7/8$, in order to satisfy detailed balance.

## 3. GPU programming

The GPU as a compute device is particularly adapted towards executing many identical instructions on different data in parallel. This is due to the specific architecture of modern GPUs. At the highest level a GPU encompasses one or more *compute units*. A high-end GPU has several dozens of these. The *compute units* themselves are comprised of several sub-units, some of them graphics specific (e.g. geometry shaders, ROPs), others can be used for general purpose computing (e.g. *processing elements*, load/store units, *local memory*). The computation is done on the *processing elements* (*PE*). An array of *PE*'s in a single *compute unit* is called a SIMD array (Single Instruction Multiple Data). *PE*'s in the same compute unit cannot execute different instructions, but they can execute the same instruction on different data. Branching is done by "disabling" individual *PE*'s using a predicate bit. Effectively, if a branch is executed in one of the *PE's*, all other *PE*'s are waiting for this branch to be finished. Thus, to build an efficient GPU program, we need to ensure that many identical instructions can be executed in parallel and branching is preferably avoided.

The most simple polymer MC simulation does not satisfy this requirement. Take for example the following structure for each step:

- Select a random monomer in the system.
- Attempt a Monte Carlo move for that monomer.
- If the move is valid, accept the move, otherwise keep the old configuration.

In this structure there is very little inherent parallelism. The obvious solution is to allow multiple monomers to be moved at once. We have to be careful not to move two monomers that are close to each other simultaneously. Otherwise it might happen that two monomers are moved to the same place, or that two monomers next to each other along the chain become separated. To prevent this, it is required that two monomers moving at the same time are at least a distance of 3 bonds apart (on the lattice). Under the condition that Monte Carlo moves only affect neighboring sites, the integrity of the system is guaranteed. All moves as described in section 2 do satisfy this condition.

## 4. GPU implementation

As established in the previous section, the most natural way to enable parallelism is to divide work spatially over the lattice. Each *work-item* is assigned its own small part of the lattice. These parts are assigned along the $t, u, v$ axes, and we'll denote them as "cubes" although the edges of the boxes are not orthogonal. To allow for more parallelism, the size of the *work-item* cubes should be small. Then, a large number of *work-items* can work together on a single system. On the other hand, the size cannot be too small either, because then the moves of the *work-items* would interfere with each other.

To find the optimal size of a *work-item* cube, we first determine how the polymers are stored in the simulation. Data is stored per lattice site, to make sure that local moves are also local modifications in memory. The data for each lattice sites is stored in the *local memory* of the GPU, which is on-chip memory that is about an order of magnitude faster than the *global memory*, and even greater benefits in terms of latency. At each Monte Carlo time step the data is loaded from the *global memory* into the *local memory*.

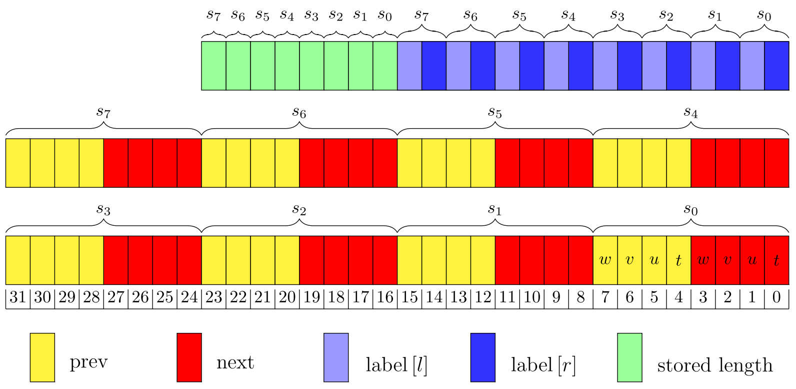

**Fig. 2.** The storage of data for a single $2 \times 2 \times 2$ block (numbered $s_0$ through $s_7$). On the horizontal axis are the bit numbers, with 0 the least significant bit. The "prev" and "next" vectors show in which direction the polymer continues. The label consists of two parts per site, because there are two possible monomers on a single site. If there is an extra monomer on the site, the corresponding stored length bit is set to "1".

Each time a site is selected to attempt a move, the following information is needed to determine the outcome of a move:

1. Is there a monomer at the site?
2. At which site is the next monomer along the chain?
3. At which site is the previous monomer along the chain?
4. Is stored length present at the site?
5. Is the destination site empty?

Additionally, it is important to be able to determine which polymer is which, for observables that follow individual polymers or monomers over a longer period of time such as the center-of-mass diffusion. This information is irrelevant while performing Monte Carlo moves, but is necessary for subsequent analysis. To preserve memory bandwidth as well as the memory capacity, it is important to keep the data structure as compact as possible. This is especially true in our case, because we use the low capacity, but high bandwidth *local memory* for temporary storage. The implementation needs a total of 11 bits per lattice site. Of these, 8 are used to denote where the neighboring monomers along the chain are (relative to the current site), using two unit vectors ("prev" and "next"). Obviously, only one of the two is sufficient to uniquely define the polymers, but using both saves computing time and relieves (bandwidth) pressure on the *local memory*. In the case of an empty site, both these vectors are set to 0. If the polymer is a linear polymer, and the monomer at the site is the last monomer, the "next" unit vector will be 0. The same goes for the "prev" vector of the first monomer. One bit is used to signify whether the site has stored length. Thus, saving *local memory* space is the motivation behind restricting the amount of stored length to one per site. The last two bits are for storing a label, which keeps track of which polymer is which. All along the polymer, a label used for identification is coded as a unique string of bits, one per monomer, $N$ per polymer. Consequently, in case of a linear polymer, the number of polymers that we can track uniquely is limited to $2^N$. Unless our polymers are very small, this number is much greater than the number of polymers in our system. In case of ring polymers, we need to be careful that the label loops around, which causes some ids to loose their uniqueness. Preventing this, the label of each polymer starts with the same sequence of 1´s and 0´s from the the first monomer. It is ensured that this starting sequence does not occur in any other place. Even then, in a homopolymer melt, the minimum ring polymer length is 21 (at $L = 96$, monomer density 1), which is quite small. Each monomer encodes a single bit of the label , while there are at most two monomers occupying the same site. Therefore, two bits per site are required to encode the labels for all polymers. During the Monte Carlo moves the label bits have to remain linked to the same monomer, and as such they are moved around in tandem.

The amount of 11 bits per site is not practical for storage. Of course, there is the possibility to store it as single 32-bit or 16-bit word, but this is somewhat wasteful. We group together $2 \times 2 \times 2$ lattice sites and store their data in the same part of memory. Two 32-bit words are used for the prev/next vectors, while one word contains both the stored length bits as well as the label. The layout is shown in detail in Fig. 2.

Since groups of $2 \times 2 \times 2$ blocks are stored together, we must ensure that neighboring *work-items* do not modify or read sites in the same group at the same step (otherwise atomic instructions are needed, which are much slower). This is achieved by assigning each *work-item* a block of size $4 \times 4 \times 4$. Each time a site is selected for a MC step, all *work-items* choose the same site relative to the $4 \times 4 \times 4$ block, with the appropriate offset. For example in Fig. 3, all *work-items* could select the bottom left sites of all the colored *work-item* blocks. Careful examination shows that this indeed prevents situations in which *work-items* access/modify the same memory address at the same time.

Most modern day GPU's are able to assign at least 48 KB of local memory to a single *work-group*. Thus, we use the largest possible *work-group* of $8 \times 8 \times 8 = 512$ *work-items*. The reason to take a *work-group* size as large as possible is that at the edges of the *work-group* block bonds connecting to other *work-group* blocks must be "frozen", i.e. they must not be

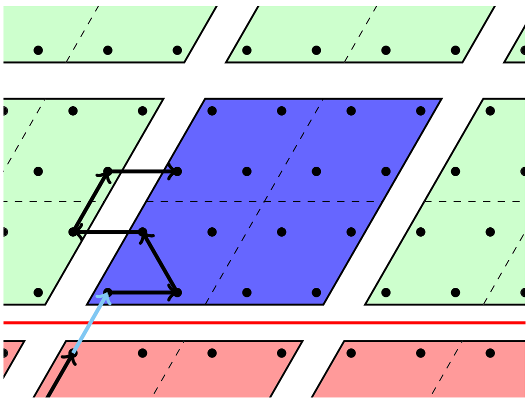

**Fig. 3.** The distribution of data among different *work-items*. The lattice is three dimensional, but for clarity a two-dimensional cross-section is shown. Each colored parallelogram denotes the collection of sites that can be selected by a single *work-item*. Each black dot denotes a lattice site. Indicated by the dashed lines are $2 \times 2 \times 2$ blocks of sites that are stored together (see Fig. 2). The red blocks belong to *work-items* of another *work-group*, and as a result information cannot be exchanged quickly across these borders. A polymer is indicated by the connected arrows. The blue arrow indicates a "frozen" bond, which cannot move during this time step, because it would change the "next" vector of a red lattice site. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

moved to maintain the integrity of the system (see Fig. 3 for an example). Having more *work-items* in a *work-group*, results in a larger *work-group* block, having a lower fraction of frozen bonds. This improves the efficiency, and decreases the chance of artifacts due to frozen bonds.

It is clear that the boundaries between *work-groups* should not remain at the same place on the lattice, because certain bonds would be frozen indefinitely. This is prevented by randomly assigning the position of the boundaries after each MC time step ($t = 0, t = 1, \ldots$). The entire lattice is stored in the *global memory*, is read by the *work-items* at the start of each time step, and is written back after each time step. In between time steps the *work-groups* are synchronized.

The number of *work-groups* is a variable that is adjustable depending on the requirements of the simulation. In the given implementation, the lattice dimensions are a direct result of the number (and arrangement) of *work-groups*. Larger lattices have the upside that wider GPUs (with more compute units) are more easily kept busy. For the results presented here, we have settled on a lattice of size $128^3$.

We have made several variants of the above sketched algorithm. One variant where the basic blocks have a size $3 \times 3 \times 3$, but with a different local memory layout to prevent race conditions. Another variant does not utilize or store the "prev" data, at the cost of being limited to simulating ring polymers. Additionally, the moves ratio (diffusion:backward:forward) is chosen as 1:1:2. The results presented are from the latter variant, but differences between the variants are relatively small, and have a big advantage over the CPU.

At the start of the program, the polymer configuration is either read from a file or constructed from scratch, and loaded into the GPU buffers. After the initialization, the flow of the GPU program is as follows:

- Choose an offset $(t, u, v)$ that determines the boundaries of the *work-groups*.
- Give control to the GPU with the given offset.
- The *work-groups* read their part of the lattice as determined by the offset and their *work-group* id, into the *local memory*.
- Each *work-item* attempts $4 \times 4 \times 4$ Monte Carlo moves (one diffusion and one transverse step each).
- The updated lattice is written back to the global memory of the GPU.
- The GPU as a whole is synchronized and a new offset is chosen.
- After a chosen number of time steps (up to $10^7$ for long polymers), the GPU buffers are read into the system memory and the polymer configuration is written to a file.

## 5. Benchmarks

### 5.1. Test setup and methodology

Benchmarks are done on two different platforms. The first is a MacBook Pro Retina (2012), the other is the local cluster in the CBP. The specifications are given in Table 1.

The performance is measured by performing a run with identical starting conditions: a melt with monomer density 1.2 of ringpolymers with length 1000. Due to stored length, an average of 28% of the sites are empty.

**Table 1**
Detailed specifications of the two systems used for benchmarking the polymer melt algorithms.

| | MacBook Pro | CBP |
|---|---|---|
| CPU Type | Intel Core i7 | Intel Xeon |
| CPU Architecture | Ivy Bridge | Sandy Bridge |
| CPU Speed | 2.3 GHz | 2.0 GHz |
| # Cores/Threads | 4/8 | 6/12 |
| # CPU | 1 | 1 |
| GPU | Nvidia 650M | Nvidia GTX 980 |
| GPU Architecture | Kepler | Maxwell |
| # GPU | 1 | 1 |
| Cuda version | 6.5.51 | 6.5.15 |
| Compiler | LLVM 6.1.0 | gcc 4.9.2 |

**Table 2**
This table shows the performance of the different test systems. The simulation is done using a dense ring polymer melt. Detailed descriptions of the systems are given in Table 1. Performance of melts which include linear polymers is slightly worse for the GPU program.

| System | Compute type | moves per second |
|---|---|---|
| Intel Core i7-3615QM 2.3 GHz | CPU (Barkema [15]) | $19.9 \cdot 10^6$ |
| Intel Xeon E5-2670 2.6 GHz | CPU (Barkema [15]) | $20.1 \cdot 10^6$ |
| Intel Core i7-3615QM 2.3 GHz | CPU | $30 \cdot 10^6$ |
| NVidia Tesla K40m | GPU/Cuda | $0.9 \cdot 10^9$ |
| NVidia Geforce GTX 970 | GPU/Cuda | $1.7 \cdot 10^9$ |
| NVidia Geforce GTX 980 | GPU/Cuda | $2.6 \cdot 10^9$ |
| NVidia Geforce GTX 980Ti | GPU/Cuda | $2.7 \cdot 10^9$ |
| NVidia Geforce GTX 1080 | GPU/Cuda | $4.1 \cdot 10^9$ |

## 5.2. Performance

Performance is measured in the number of Monte Carlo moves (1 diffusion move + 1 transverse move). Performance of the different systems is shown in Table 2. It compares the performance of the CPU programs and the GPU implementation. An alternative single CPU core implementation [15] is included for comparison. One has to be careful to simply compare the number of moves per second of different algorithms, because the (effective) density, entanglement length, and move acceptance probabilities are all different.

In the context of dense ring polymer melts, a more interesting measure for the efficiency is: how much real time does it take to displace a polymer of size $cN_e$ over its own radius of gyration.

$$r_e = N_e^{-3.2} t_{\text{fac}}^{-1} \text{ moves/s}, \tag{2}$$

where $t_{\text{fac}}$ is a measure for how many Monte Carlo steps it takes for a ring polymer of size $N_e$ to diffuse over its own radius of gyration. An estimate for $N_e$ was obtained for the original version of the GPU program using the method described in [16], and using the superposition of Rouse modes, an estimate for the remaining density settings was determined. For the CPU algorithm, $N_e$ ranges from 120 to 260 (which includes stored length). For the GPU algorithm the range is tighter at between 115–125.

The efficiency of the CPU and GPU programs for several density and move ratio settings has been plotted in Fig. 4. Between the best performing CPU and GPU programs lies a factor of 100 in favor of the GPU.

## 5.3. Simulation runs

Table 3 summarizes the runs that were done for this paper. The CPU simulations were done with the following compiler flags: “-O2 -mavx”, which performed the best.

## 6. Application: a melt of ring polymers

As motivated in the introduction, ring polymers likely hold a part of the key in our understanding of large scale chromatin organization. The algorithm described in this paper is very suitable for the system of ring polymers in a melt, because the desired density of the melt is high. The section is structured as follows: section 6.1 gives the definition of the Rouse modes, and introduces its formalism. Section 6.2 shows the simulation results on the static properties of the melt. The dynamics of the system are treated in section 6.3.

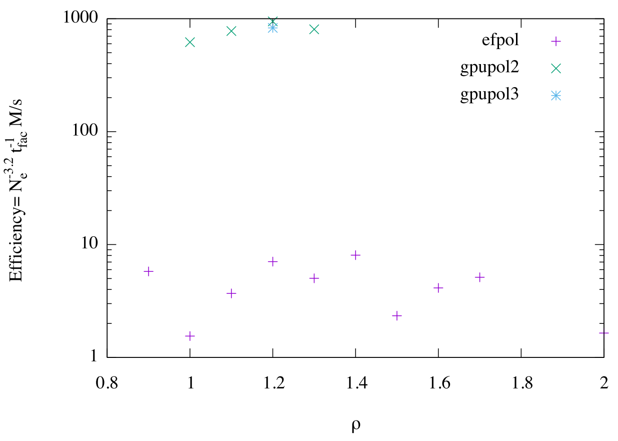

**Fig. 4.** The efficiency of different variants of the GPU (gpupol) and CPU (efpol) algorithms for different densities and move ratios. The efficiency is defined in Eq. (2). The purple symbols give the CPU algorithm performance numbers, while the blue and green symbols denote the GPU results. The GPU algorithms are about a factor 100 more efficient then the CPU ones.

**Table 3**
A summary of all MC simulations done for this paper. All simulations have a lattice size of $L^3 = 128^3$. The first column shows the number of monomers of the ring polymers in the melt. The simulation time in steps is shown in the second column. “Eq. time” is the amount of MC time steps that are done before measurements are done. The longest run with $N = 4000$ takes around 440 GPU hours to finish on a NVidia Geforce GTX 1080. In terms of entanglement length it corresponds to about $32N_e$.

| $N$ | MC timesteps | Eq. time |
|---|---|---|
| 30 | $999 \cdot 10^2$ | $150 \cdot 10^2$ |
| 60 | $520 \cdot 10^3$ | $15 \cdot 10^3$ |
| 100 | $1003 \cdot 10^3$ | $33 \cdot 10^3$ |
| 150 | $303 \cdot 10^4$ | $8 \cdot 10^4$ |
| 200 | $503 \cdot 10^4$ | $15 \cdot 10^4$ |
| 300 | $300 \cdot 10^5$ | $3 \cdot 10^5$ |
| 375 | $500 \cdot 10^5$ | $6 \cdot 10^5$ |
| 500 | $700 \cdot 10^5$ | $11 \cdot 10^5$ |
| 700 | $2000 \cdot 10^5$ | $25 \cdot 10^5$ |
| 1000 | $300 \cdot 10^6$ | $6 \cdot 10^6$ |
| 1500 | $500 \cdot 10^6$ | $22 \cdot 10^6$ |
| 2000 | $1031 \cdot 10^6$ | $55 \cdot 10^6$ |
| 3000 | $2000 \cdot 10^6$ | $194 \cdot 10^6$ |
| 4000 | $300 \cdot 10^7$ | $47 \cdot 10^7$ |

### 6.1. Rouse modes

Rouse modes are a very useful tool, because the formalism is well established and studying the dynamics of polymers becomes more intuitive (see for example [17] as a reference). In addition, the effect of closing the polymer chain induces effects in e.g. the mean squared Euclidean distance $\langle r(g)^2 \rangle$ as a function of the genomic distance $g$, which is defined as the number of bonds between the two monomers. This effect is seen to a much smaller extent in the Rouse modes themselves, but shows itself again in the mathematical transformation from Rouse modes to Euclidean space.

Given a polymer with monomeric positions $\{\mathbf{r_0}, \dots, \mathbf{r_{N-1}}\}$, we define the Rouse modes $\mathbf{C}_p$ and $\mathbf{S}_p$ as ($N$ is even):

$$\mathbf{C}_p = \sqrt{2/N} \sum_{n=0}^{N-1} \cos\left(\frac{2\pi p(n+1/4)}{N}\right) \mathbf{r_n}, \quad p = 1, 2, \dots N/2 \tag{3}$$

and

$$\mathbf{S}_p = \sqrt{2/N} \sum_{n=0}^{N-1} \sin\left(\frac{2\pi p(n+1/4)}{N}\right) \mathbf{r_n}, \quad p = 1, 2, \dots N/2 - 1 \tag{4}$$

and the center of mass mode as

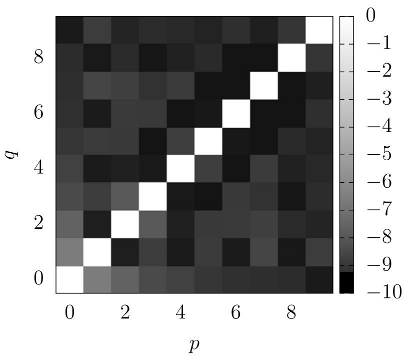

**Fig. 5.** Correlation map of the natural logarithm of the correlation coefficients $\log|c_{p,q}|$, between the Rouse modes $p$ and $q$. Values with $|c_{p,q}| < 10^{-4}$ are adjusted upward to $10^{-4}$ to have a better range of colors. By definition the correlation coefficients on the diagonal are 1. The figure shows that the off-diagonal elements are much smaller and close to 0, within error margins. Thus, we conclude that the Rouse mode amplitudes are approximately independent. The length of the ringpolymers is 3000. The largest off-diagonal element with $p, q$ is $c_{0,1}$, for which we found $|c_{0,1}| = 0.006$.

$$\mathbf{C_0} = \sqrt{1/N} \sum_{n=0}^{N-1} \mathbf{r_n}. \tag{5}$$

By changing the numbering along the ring polymer, we can transform $\mathbf{S}_p$ into $\mathbf{C}_p$ or reverse. Thus, it is easy to see that we must have $\langle |\mathbf{S}_p|^2 \rangle = \langle |\mathbf{C}_p|^2 \rangle$. For brevity, we will leave out the $\mathbf{C}_{N/2}$ term in the upcoming formula's. In case $N$ is odd, the largest term is $\lfloor N/2 \rfloor$. When $N$ is even, there is one additional term with $\mathbf{C}_{N/2}$ that has its prefactor divided by 2 (because $\mathbf{S}_{N/2}$ does not exist). By using basic algebra and using the definitions we find the mean squared displacement as a function of the Rouse amplitudes:

$$\left\langle |\mathbf{r}_n - \mathbf{r}_{n+g}|^2 \right\rangle = \frac{8}{N} \sum_{p=1}^{N/2-1} \sin^2\left(\frac{\pi p g}{N}\right) |\mathbf{C}_p|^2. \tag{6}$$

## 6.2. Static properties

First we check whether the Rouse modes are independent of each other. To this effect, we plot the correlation $c_{p,q}$ between the Rouse modes:

$$c_{p,q} = \frac{\langle \mathbf{C}_p \cdot \mathbf{C}_q \rangle}{|\mathbf{C}_p||\mathbf{C}_q|} \tag{7}$$

The correlation map is shown in Fig. 5, for $p, q < 10$. There are no clear patterns visible, except the diagonal itself. Plots for smaller $N$ give similar results. Thus, we can assume in our calculations that the correlations are negligible and the Rouse modes are to a large extent independent:

$$c_{p,q} \approx \delta_{p,q}, \text{ for all } \ p, q. \tag{8}$$

Next, we want to probe whether the Rouse modes depend only on the length scale $\ell = N/p$, and not directly on the molecular weight $N$ or the mode number $p$. For clarity, we define a Rouse amplitude that depends on the length scale: $\mathbf{X}_\ell = \mathbf{C}_{N/p}$. We expect in that case that the Rouse amplitudes $|\mathbf{X}_\ell|$ scale as:

$$|\mathbf{X}_\ell|^2 \sim \ell^{1+2\nu}. \tag{9}$$

Here, $\nu$ is the growth exponent, which is 1/2 in the case of a linear polymer in a melt, and 1/3 in the case of a compact fractal. For better comparison between different simulations we also use a version of $\ell$ that is rescaled with the entanglement length:

$$\ell_z = \ell/N_e \tag{10}$$

Fig. 6 shows $|\mathbf{X}_{\ell_z}|^2$ as a function of $\ell_z$. It features a very slow crossover from supposedly SAW/Gaussian behavior at the scale of single bonds to the fractal regime ($\nu = 1/3$) at $\ell_z \approx 10$. An intermediate regime, where excluded volume is screened, but the ring polymer is not yet becoming compact might be expected [18]. However, the shape of the crossover makes a determination of this intermediate exponent inconclusive at best. Modes with low $p$ (specifically $p = 1$) do not exactly collapse, which is remarkable, because this deviation does not occur for either Gaussian rings or self-avoiding rings in a melt (with concatenation).

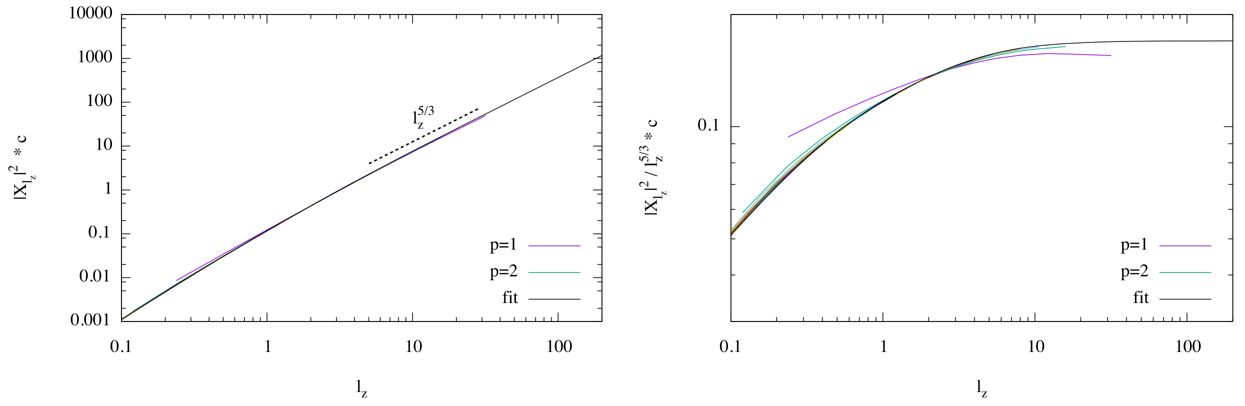

**Fig. 6.** The left hand figure shows the collapse of the Rouse amplitude $|\mathbf{X}_{\ell_z}|^2$, with $\ell_z = \ell/N_e$. The fit is a very slow crossover function towards $|\mathbf{X}_{\ell_z}|^2 \sim \ell_z^{5/3}$. Asymptotically, this would mean a compact object, i.e. $\nu = 1/3$. The lengths of the polymers are given in Table 3. The right hand figure shows the same collapse, but is divided by $\ell_z^{5/3}$, thus a horizontal line denotes an asymptotically compact object. This magnifies slowly emerging disparity between low mode numbers $p$ at long lengths. It is clearly significant and is also observed in other (unpublished) work. Its implications for the asymptotic behavior, however, are hard to judge, because we lack data on even longer polymers.

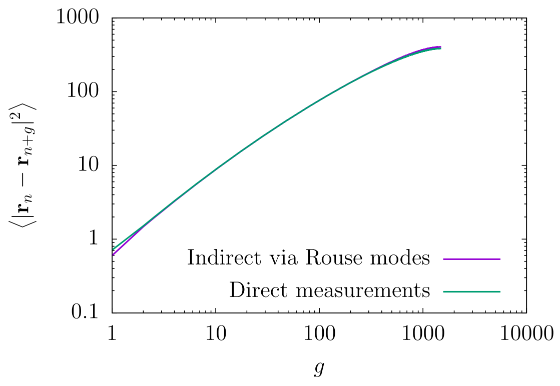

**Fig. 7.** Squared Euclidean distance $\langle r^2 \rangle$ versus genomic distance $g$. One curve is derived from the approximated Rouse amplitudes, using Eq. (6), the other is direct measurement from the simulation. The curves agree very well. The length of the ring polymers is $N = 3000$. The deviation at long lengths is due to the difference in the $p = 1$ mode. As can be seen in Fig. 6, the $p = 1$ mode at large length scales has a smaller amplitude $\mathbf{X}_\ell$, which causes the squared distance to be overestimated by using the fit.

Numerically and theoretically a great deal of work has been done that suggests $\nu = 1/3$ [12,18–21], and our work clearly points in the same direction.

By combination of the fit in Fig. 9 and eq. (6), we obtain the average squared distance along the chain $r(g)^2$. The resulting curve is plotted in Fig. 7, together with direct measurements of this quantity. As expected, the long-scale behavior is captured quite well, although a small deviation exists, because the $p = 1$ Rouse mode deviates from the other modes in the collapse.

In the context of chromatin organization, the contact probability is a sought after indicator, because it can be very accurately measured by Hi-C experiments. The contact probability for a Gaussian chain as a function of the genomic distance $g = |i - j|$ scales as $g^{-3/2}$. The contact probability for mammalian/eukaryotic DNA at the very large scale is mostly measured around $p_c \sim g^{-1.1}$. Melts of rings, being candidates in this context, are often compared in this regard, and generally fall in the range of $p_c \sim g^{-1.2}$. However, it is somewhat unclear whether this observed behavior is asymptotic, not in the least given the slow crossover of the Rouse mode amplitudes. A direct difficulty pertaining to its asymptotic behavior is that the looping of the ring affects the contact probability long before the halfway point of the polymer. To alleviate this issue, we propose measuring the contact probability, not as a function of genomic distance $g$, but as a function of the average squared distance between two monomers $\langle r^2 \rangle$. Translational invariance dictates that two monomers $g$ monomers apart have the same average squared distance between them. Since the average squared distance increases monotonically, the relation between the two must be bijective.

The simulation data in Fig. 8 clearly indicates that $p_c \sim r^{-3}$, as far as it reaches. However, this indicates a problem with the hope that this is asymptotic behavior, since combining this with $r \sim g^{1/3}$ results in:

$$p_c \sim r^3 \rightarrow p_c \sim g^{-1}, \tag{11}$$

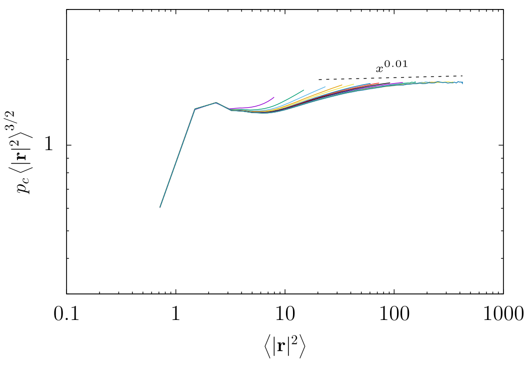

**Fig. 8.** The contact probability times the averaged squared distance to the power 3/2 as a function of the average squared distance. The different colored lines indicate different polymer sizes. Asymptotically, it seems to converge towards a constant, indicating $p_c \sim r^{-3}$.

which is technically impossible ($p_c$ integrated over $g$ should be limited by the density times the contact surface, i.e. a constant). While in principal this can be overcome by lowering the exponent by an infinitesimal amount, imagined conformations with exponents close to $-1$ seem unlikely, which raises the question whether this is really the asymptotic regime.

## 6.3. Dynamical properties

In the description of a single polymer in solution by the Rouse model, the Rouse modes are the exact dynamical eigenmodes, and consequently their decay is strictly exponential. With excluded-volume interactions between the beads of a single linear polymer in solution, the Rouse modes are no longer the exact dynamical eigenmodes, but still to a very good approximation, and are observed to decay exponentially, be it with times that scale differently from those in the Rouse model [22]. The Rouse modes of the individual polymers provide a useful tool to characterize the dynamics of linear polymers in dense solutions [23]. However, apart from the very short times and very long times, their decay changes from exponential to stretched-exponential [23], with stretched exponent $\alpha = 1/2$. Given that in the case at hand the solution is dense as well, we expect stretched-exponential decay of the Rouse modes, possibly with a different stretched exponent:

$$\langle \mathbf{X}_\ell(t) \cdot \mathbf{X}_\ell(t+dt) \rangle \sim \exp(-(dt/\tau_\ell)^\alpha). \tag{12}$$

Here, $\tau_\ell = A\ell^\gamma$ is the characteristic time at the length scale $\ell$. To find a collapse, we use the following:

$$\log(-\log((\langle \mathbf{X}_\ell(t) \cdot \mathbf{X}_\ell(t+dt) \rangle)/\left\langle \mathbf{X}_\ell(t)^2 \right\rangle = \alpha(\log dt - \gamma \log \ell - \log A). \tag{13}$$

Thus, with the addition of $\alpha\gamma \log \ell$ to the left term, we get an expression independent of $\ell$, and the curves with different $N$ and $p$ should collapse. We obtain $\alpha$ from plotting that expression against $\log dt$ and finding the slope, which in turn determines the most interesting exponent $\gamma$. This procedure is shown in Fig. 9, from which we obtain $\beta \approx 1.88$, $\alpha \approx 0.61$, $\gamma \approx 3.1$, asymptotically. Initially for small time scales the following is measured: $\beta \approx 1.88$, $\alpha \approx 0.85$, $\gamma \approx 2.2$.

Interestingly, the collapse of curves with equal modes is near perfect, but at long times curves with different modes start to spread out. On the other hand, curves with the same mode numbers $p$ do seem to coincide, which indicates that the size of the polymer $N$ directly affects the dynamics at lower length scales $\ell$. Thus, for example, a quarter of a ring polymer behaves differently from half of a ring that is half the size. From collapsing Fig. 9, we find that approximately:

$$\tau_{e,p} \approx \tau_{e,\text{ring}} p^{-2.2} \tag{14}$$

Thus, instead of perhaps a more expected single transition/entanglement time $\tau_{e,\text{ring}}$, it appears to be a more gradual transition where higher mode numbers are transitioning first. The actual theoretical meaning of this currently eludes us, but one idea is to look at the external pseudo-entanglement between nearby ring polymers, to explain why the first mode is "special". However, correlation between center of mass displacements of nearby ring polymers was too small to be observed, which puts the importance of inter-polymer entanglement on the overall ring dynamics in question.

The zeroth Rouse mode is the center of mass displacement. This mode behaves very differently from the other Rouse modes. Instead of stretched-exponential decay, it is expected that for long enough time the polymer exhibits Brownian diffusive motion: $\langle r_{cm}^2 \rangle = 6Dt$, with the diffusion constant $D$. For unrestricted movement (that is to say without any entanglement barriers), we expect the diffusion constant to be proportional to $1/N$, which follows directly from the total mass scaling as $N$, combined with independence of the thermal kicks whose number also scales as $N$, but which combine to a total force vector scaling as $\sqrt{N}$. Fig. 10 shows the mean squared displacement of the center of mass as a function of time. It is clear that the diffusion coefficient is not proportional to $1/N$, as the data would have collapsed in that scenario.

From the internal dynamics of the polymer chain we see no direct unambiguous way to derive the diffusion constant, without additional arguments or assumptions. One reasonable assumption is that the time required for the polymer to

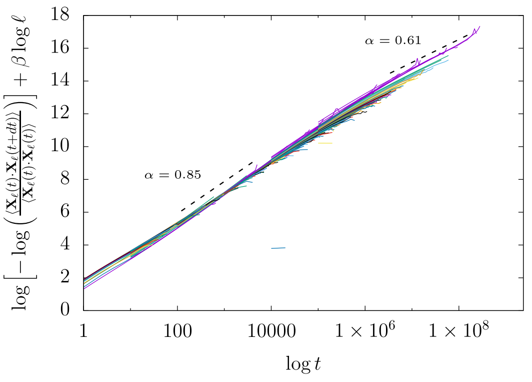

**Fig. 9.** Collapse of the auto correlation of the Rouse modes $|X_\ell|$. The left-side plot uses a double logarithmic scale on the y-axis, and a single logarithmic scale for the x-axis. Up to the $\tau_{e,\text{ring}}$ the only relevant parameter is the length scale $\ell$ along the chain. In case of a purely stretched exponential, the lines should collapse in a straight line. We have used $\beta = 1.88$ to obtain the collapse in the figure. From this we find that $\tau$ transitions from about $\ell^{2.2}$ (Rouse time for unentangled SAW polymers) to $\ell^{3.1}$, though it is hard to tell from our data, whether this is asymptotic behavior. The exponent $\beta$ is a derived exponent: $\beta = \alpha\gamma$, which is explained in the main text. The plot contains ring lengths shown in Table 3, and logarithmically chosen Rouse modes (each subsequent value for $p$ is at most 1.2 times higher).

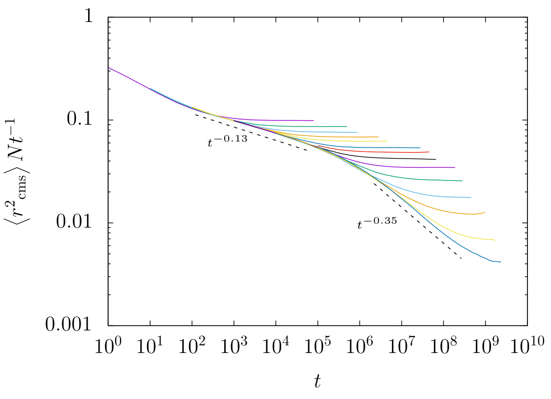

**Fig. 10.** Mean squared displacement of the center of mass of the ring polymers, multiplied by the length of the polymer, and divided by the time as a function of time. The length of polymers is given in Table 3 and ranges from $N = 30$ to $N = 4000$. For a Rouse chain, these curves will collapse onto a single straight line with slope 0.

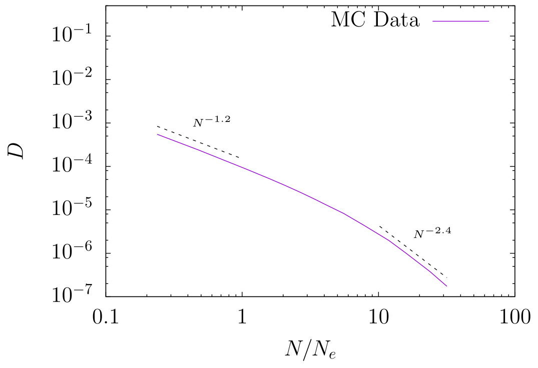

**Fig. 11.** The diffusion constant $D$ as a function of the molecular weight of the ring polymers $N$. The green and purple lines are used to estimate the asymptotic behavior of the diffusion coefficient.

displace over its own size coincides with the timescale of the largest internal mode (a crossover between $t \sim N^{2.2}$ and $t \sim N^{3.1}$) up to a constant factor. Using this assumption we find a diffusion coefficient of $D \sim r^2/t = N^{2\nu-\gamma}$, which is expected to show a crossover from $D \sim N^{-1.2}$ to $D \sim N^{-2.4}$. To test this hypothesis, we plotted the diffusion coefficient found in Fig. 10 as a function of the length of the polymer $N$ in Fig. 11. The plot shows exponents that are close to the range that we predicted. The paper by Halverson et al. [24] claims to find $D \sim N^{-2.3}$. This is definitely within statistical error of our data, especially considering finite size effects.

## 7. Conclusion and discussion

This paper shows that with the right implementation, GPUs are very efficient at performing polymer simulations. We implemented the elastic lattice polymer model, which has been used before and has proven its efficiency. Our implementation on the GPU is about two orders of magnitude faster than any previous works on the CPU. The scaling of the algorithm with faster GPUs is promising as well. With manufacturing and design limits on how fast a single CPU core can be, the wide parallel nature of GPUs makes it likely that the performance discrepancy between the algorithms will only grow with time. Profiling the GPU program also indicated that the computation is ALU (Arithmetic Logic Unit) bound, which further strengthens our confidence in its scalability.

We find that in a melt of ring polymers there is a slow crossover to the compact/fractal regime. Rouse modes allow for a clean characterization of this crossover. It shows that ring polymers are compact objects in the limit where the molecular weight $N$ goes to infinity. From the scaling of the Rouse mode amplitudes we derived scaling of the Euclidean distance versus genomic distance.

The internal dynamics of ring polymers is faster than that of linear chains in a melt of linear chains. We found that the dynamical behavior of polymers shows complex behavior. Asymptotically, we find that the longest time scales are scaling as $\tau \sim \ell^{3.1}$. In conjunction, the diffusion constant scales as $D \sim N^{-2.4}$, where the relation between the two entities is governed by the size of the polymer over which it has to diffuse before equilibrating.

Finally, there is the question of wider application of the algorithm to more complex problems. Our approach is extensible on the condition that there are only short range interactions between polymers.

In case the problem demands the storage of more than 1 bit extra per site, the amount of local memory assigned per site has to be increased, and consequently the number of *work-items* per *work-group* has to be decreased.

The code for the GPU simulations is available on request by emailing the first author of this paper.

## Acknowledgements

Computing time on the "Cartesius" supercomputer, which is funded by the Dutch agency NWO, is acknowledged. This work is part of the research programme of the Foundation for Fundamental Research on Matter (FOM), which is financially supported by the Netherlands Organisation for Scientific Research (NWO) (11PR2854). This work was supported by Agence National de la Recherche (ANR-15-CE12-00006-EpiDevoMath). Acknowledged is Helmut Schiessel for reading the paper and suggestions for improvement. We also acknowledge Ralf Everaers for discussions about the entanglement length.

## References

[1] J.K. Wolterink, G.T. Barkema, M.A.C. Stuart, Diffusion and exchange of adsorbed polymers studied by Monte Carlo simulations, Macromolecules 38 (2005) 2009–2014.
[2] J.K. Wolterink, G.T. Barkema, D. Panja, Passage times for unbiased polymer translocation through a narrow pore, Phys. Rev. Lett. 96 (2006) 208301.
[3] M. Baiesi, G.T. Barkema, E. Carlon, D. Panja, Unwinding dynamics of double-stranded polymers, J. Chem. Phys. 133 (2010) 154907.
[4] Y. Zhao, Lattice Boltzmann based PDE solver on the GPU, Vis. Comput. 24 (5) (2007) 323–333.
[5] R.D. Schram, Reaction-diffusion model Monte Carlo simulations on the GPU, J. Comput. Phys. 241 (2013) 95–103.
[6] T. Preis, P. Virnau, W. Paul, J.J. Schneider, GPU accelerated Monte Carlo simulation of the 2d and 3d Ising model, J. Comput. Phys. 228 (12) (2009) 4468–4477.
[7] J.A. van Meel, A. Arnold, D. Frenkel, S.F.P. Zwart, R.G. Belleman, Harvesting graphics power for MD simulations, Mol. Simul. 34 (3) (2008) 259–266.
[8] J.A. Anderson, C.D. Lorenz, A. Travesset, General purpose molecular dynamics simulations fully implemented on graphics units, J. Comput. Phys. 227 (10) (2007) 5342–5359.
[9] S. Nedelcu, M. Werner, M. lang, J.-U. Sommer, Implementations of the bond fluctuation model, J. Comput. Phys. 231 (7) (2011) 2811–2824.
[10] NVidia Corp., Cuda programming guide, 2015.
[11] Khronos Group, OpenCL specifications, 2015.
[12] J.D. Halverson, W.B. Lee, G.S. Grest, A.Y. Grossberg, K. Kremer, Molecular dynamics simulation study of nonconcatenated ring polymers in a melt. I. Statics, J. Chem. Phys. 134 (2011) 204904.
[13] M. Rubinstein, Discretized model of entangled-polymer dynamics, Phys. Rev. Lett. 59 (1987) 1946.
[14] G. Barkema, J. Marko, B. Widom, Electrophoresis of charged polymers: simulation and scaling in a lattice model of reptation, Phys. Rev. E 49 (1994) 5303–5309.
[15] A. van Heukelum, G.T. Barkema, Reaching large lengths and long times in polymer dynamics simulations, J. Chem. Phys. 119 (2003) 8197–8202.
[16] N. Uchida, G. Grest, R. Everaers, Viscoelasticity and primitive path analysis of entangled polymer liquids: from f-actin to polyethylene, J. Chem. Phys. (2008) 044902.
[17] M. Doi, S.F. Edwards, The Theory of Polymer Dynamics, Oxford University Press, 1990.
[18] T. Sakaue, Ring polymers in melts and solutions: scaling and crossover, Phys. Rev. Lett. 106 (2011) 167802.
[19] J. Suzuki, A. Takano, Y. Matsushita, Topological effect in ring polymers investigated with Monte Carlo simulation, J. Chem. Phys. 129 (2008) 034903.
[20] T. Vettorel, A.Y. Grosberg, K. Kremer, Statistics of polymer rings in the melt: a numerical simulation study, Phys. Biol. 6 (2009) 025013.
[21] A.Y. Grosberg, Annealed lattice animal model and flory theory for the melt of non-concatenated rings: towards the physics of crumpling, Soft Matter 10 (2014) 560–565.
[22] D. Panja, G.T. Barkema, Rouse modes of self-avoiding flexible polymers, J. Chem. Phys. 131 (2009) 154903.
[23] G.T. Barkema, D. Panja, J.M.J. van Leeuwen, Structural modes of a polymer in the repton model, J. Chem. Phys. 134 (2011) 154901.
[24] J.D. Halverson, W.B. Lee, G.S. Grest, A.Y. Grossberg, K. Kremer, Molecular dynamics simulation study of nonconcatenated ring polymers in a melt. II. Dynamics, J. Chem. Phys. 134 (2011) 204905.