Available online at www.sciencedirect.com

ScienceDirect

Physics Procedia 15 (2011) 29–32

Physics Procedia

A GPU approach to parallel replica-exchange polymer simulations

Jonathan Gross[a,b], Wolfhard Janke[c], Michael Bachmann[a,b]

[a] *Soft Matter Systems Research Group, Institute for Complex Systems (ICS-2) and Institute for Advanced Simulation (IAS-2), Forschungszentrum Jülich, D-52425 Jülich, Germany*

[b] *Center for Simulational Physics, University of Georgia, Athens, Georgia 30602, USA*

[c] *Institut für Theoretische Physik and Centre for Theoretical Sciences (NTZ), Universität Leipzig, Postfach 100920, D-04009 Leipzig, Germany*

Abstract

We investigate new programming techniques for parallel tempering Monte Carlo simulations of an elementary bead-spring homopolymer model using graphics processing units (GPUs). For a precise estimation of statistical quantities, like the peak structure of the specific heat, a large number of conformations with substantial statistical data is needed. Therefore the advantage of gathering this data faster by improving the performance of Monte Carlo simulations cannot be overrated. With the huge computational capability of the large number of cores on GPUs, that can be exploited by means of multithreaded programming, we find significant increases in efficiency compared to CPU-only simulations.

Keywords:
GPU computing, CUDA, structural transitions, generalized ensemble methods, polymers

1. Introduction

Computer simulations in physics have become very important to understand the behavior of complex systems, like in biophysics and polymer physics, that cannot be investigated by purely analytical means. The accurate calculation of many-body interactions is a huge computational challenge. But even for simplified models computing time can increase dramatically, when a large amount of statistical data is needed.

Although the computing power of CPUs has evolved over the years most computational tasks can not be tackled by simulations on single CPUs in an acceptable time. The need for more advanced algorithms and programming paradigms has grown. Various approaches have been applied to speed up simulations, parallel computing on large scale clusters using *message passing* or *multithreading* on recent multicore CPUs.

In recent years graphics processing units (GPUs), driven by the computer gaming industry, have become very powerful computing devices. They possess a parallel architecture similar to multicore CPUs, but to a greater extent. With the availability of convenient programming interfaces to these devices, like OpenCL[1] and NVIDIA's CUDA[2], it has become simpler to exploit the computing capabilities of GPUs at a mainstream level. GPUs find their applications in many fields, like astronomy [1], medicine [2], molecular dynamics simulations [3] and Monte Carlo simulations of spin models [4, 5]. We are particularly interested in the thermodynamical behavior of polymer models on lattices [6, 7] and off-lattice [8, 9].

[1] `http://www.khronos.org/opencl/`
[2] `http://www.nvidia.com/cuda`

1875-3892
doi:10.1016/j.phpro.2011.05.055

2. Thermodynamic properties of coarse-grained polymers

We examined a coarse-grained model for an elastic polymer. The interaction between the monomers is described by a Lennard-Jones-like potential $E_{\mathrm{LJ}}(r_{ij}) = 4\epsilon\left[\left(\sigma/r_{ij}\right)^{12} - \left(\sigma/r_{ij}\right)^{6}\right]$, where r_{ij} is the distance between monomers i and j. The bonds are modeled with the finitely extensible nonlinear elastic (FENE) anharmonic potential, which is given by $E_{\mathrm{FENE}}(r_{ii+1}) = -\frac{K}{2}R^2 \log\left[1 - \left(\frac{r_{ii+1}-r_0}{R}\right)^2\right]$. It has its minimum at r_0 and diverges for $r \to r_0 \pm R$. The precise parametrization can be found in [9].

Previous similar studies off-lattice [9, 10] and on lattices [11] showed that these polymer models undergo complex structural transitions which depend on temperature and chain-length [9]. Above the Θ-temperature the polymer forms

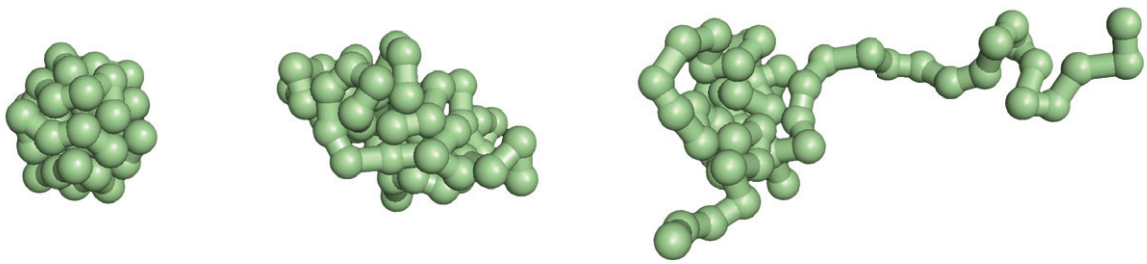

Figure 1: Conformations of a 55mer. On the right is a typical random structured coil, which can be found at high temperatures. The structure in the middle shows a collapsed polymer below the Θ-point. At even lower temperatures a "liquid-solid"-transition occurs. The polymer now has a crystalline form as shown on the left.

random coils. At the Θ-point the monomer-monomer attraction is at balance with the repulsion by volume exclusion. The polymer collapses from the coil to globular structures. In this "liquid phase", there is no internal structure. When going to even lower temperatures a freezing transition occurs. Representative structures in these phases are illustrated in Fig. 1.

Whereas the Θ-transition is very well examined and understood, the liquid-solid transition at low temperatures still is to be investigated in more detail. This low temperature regime is virtually impossible to be simulated by means of simple Metropolis [12] Monte Carlo methods. More sophisticated algorithms help to avoid the trapping of conformations in local minima of the free energy and thus the slowing down of the sampling of phase space.

Successful examples are histogram reweighting algorithms like multicanonical sampling [13] or the Wang-Landau method [14]. Another technique is parallel tempering, also known as replica-exchange [15]. In parallel tempering a number of copies or replica of the same system are simulated at different temperatures. After a certain number of Monte Carlo steps an exchange of conformations among neighboring temperatures is proposed. A swap is accepted with the probability $p = \min\left(1, \exp\left[(E_i - E_j)(\beta_i - \beta_j)\right]\right)$. The acceptance of such an exchange therefore depends on the difference in energy and inverse temperature $\beta = 1/k_B T$ of the two copies. This can be identified by the overlap of the energy histograms of each copy at that specific temperature. The bigger the overlap, the higher the probability of a swap of conformations. With the application of this algorithms, each replica of the system performs a random walk in temperature space and thus effectively heating up and cooling down the system.

In this work parallel tempering was chosen, because it can be parallelized very easily. The replica of the system are simulated via Metropolis importance sampling. As Monte Carlo update proposal a simple displacement of coordinates of a randomly selected monomer of the chain was chosen.

3. Parallelizing code for GPUs

With the embarrassingly parallel nature of the replica-exchange method it is very straightforward to develop a parallel code for the simulation. Since each replica is simulated completely independent from all others, for a given number of Monte Carlo "time steps", one can run a separate program thread for each copy. This is valid for programming of CPUs and GPUs alike. As depicted on the left of Fig. 2 each of the N replica is assigned to a work thread. These threads are then processed by the available number of cores in the CPU or the GPU, respectively.

CPUs typically have 4–8 cores nowadays, GPUs have hundreds of compute cores (512 for NVIDIA's latest Fermi-cards[3]). One can basically use the same code for both architectures. Only little additional programming effort is necessary to utilize the GPU. The programmer has to allocate memory on the GPU and to explicitly copy the data to the device. That means all memory allocation and initialization of configurations is done on the CPU side and only the calculation is performed by the GPU. After copying the needed information the GPU memory, the so-called *kernel*, is called. The kernel is the main function of the GPU code. It is a normal C function with a special keyword in front of its declaration to indicate that this function is meant to be executed by the GPU. When all calculations on the GPU device finish, one has to copy the results back to main memory for further processing or evaluation. The NVIDIA CUDA toolkit provides convenient functions for memory management, making it easy for programmers to port their existing multithreaded code to GPUs.

Figure 2: Schematic representation of multithreaded parallelization on CPUs on the left and twofold parallelization on GPUs on the right.

Up to this point there is not much difference between multithreading on CPUs or GPUs. But even with simply porting a multithreaded program to run on a GPU can bring speed-ups of factors of about 10 compared to single CPU implementations, see Ref. [16]. The speed-up depends on the number of cores available on each device and of course the number of replicas. In our tests [16] we find optimal numbers of threads to gain maximum speed-ups that conform with the number of processors on the GPUs. Graphics cards have a scheduler that distributes threads to the core in a very intelligent way. Hence it is possible and even recommended to executed more threads than there are cores available on the GPU. Otherwise the GPU might not be used to capacity. Since GPUs have different layers of memory with different latencies, with a vast number of running threads, the scheduler is able to hide memory latency by postponing threads that are waiting for data and running threads that already have all needed data for execution. With this knowledge we improved the implementation. Instead of running one thread per replica, each copy of the systems was delegated to a block of M threads, see Fig. 2 on the right. These M threads were used to calculate the energy function, the most demanding calculation of the simulation, in a parallel way. More than ten thousand threads were running in our simulations. A unimaginable large number when talking about multithreading on CPUs. These adaptations to the program introduced even bigger speed-up factors up to 130 with a single graphics card compared to a single CPU. For a more detailed discussion, see Ref. [16].

4. Conclusions

The use of graphic cards as accelerators in scientific simulations promises huge and easy achievable speedups. With faster and cost-efficient computing power it is possible to get better results faster. Also, with algorithms that scale well it is possible to investigate larger systems.

We acknowledge support by the German-Israeli program "Umbrella" under Grant No. HPC_2, the German Research Foundation (DFG) under Grant Nos. JA 483/24-2/3, the Leipzig Graduate School of Excellence "BuildMoNa", the German-French DFH-UFA PhD College under Grant No. CDFA-08-07, and by the Forschungszentrum Jülich for supercomputing time grants jiff39 and jiff43.

References

[1] E. B. Ford, New Astronomy **14** (2009) 406–412.

[3] Fermi-based cards can have up to 16 processors with 32 cores each.

[2] X. Gu, D. Choi, C. Men, H. Pan, A. Majumdar, S. B. Jiang, Phys. Med. Biol. **54** (2009) 6287–6297.
[3] J. A. van Meel, A. Arnold, D. Frenkel, S. F. Portegies Zwart, R. G. Belleman, Molecular Simulation **34** (2008) 259–266.
[4] M. Weigel, Comput. Phys. Commun., in press (2011).
[5] J. Yin, D. P. Landau, Phys. Rev. E **80** (2009) 051117-1–8.
[6] M. Bachmann, W. Janke, Phys. Rev. Lett. **91** (2003) 208105-1–4.
[7] M. Bachmann, W. Janke, J. Chem. Phys. **120** (2004) 6779–6791.
[8] M. Bachmann, H. Arkın, W. Janke, Phys. Rev. E **71** (2005) 031906-1–11.
[9] S. Schnabel, T. Vogel, M. Bachmann, W. Janke, Chem. Phys. Lett. **476** (2009) 201–204.
[10] D. T. Seaton, T. Wüst, D. P. Landau, Phys. Rev. E **81** (2010) 011802-1–10.
[11] T. Vogel, M. Bachmann, W. Janke, Phys. Rev. E **76** (2007) 061803-1–11.
[12] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, E. Teller, J. Chem. Phys. **21** (1953) 1087–1092.
[13] B. A. Berg, T. Neuhaus, Phys. Lett. B **267** (1991) 249.
[14] F. Wang, D. P. Landau, Phys. Rev. Lett. **86** (2001) 2050.
[15] K. Hukushima, K. Nemoto, J. Phys. Soc. Jpn., **65** (1996) 1604-1–4.
[16] J. Gross, W. Janke, M. Bachmann, Comput. Phys. Commun., in press (2011).