Parallel kinetic Monte Carlo simulation of Coulomb glasses
E. E. Ferrero, A. B. Kolton, and M. Palassini

Citation: AIP Conference Proceedings 1610, 71 (2014); doi: 10.1063/1.4893513
View online: http://dx.doi.org/10.1063/1.4893513
View Table of Contents: http://scitation.aip.org/content/aip/proceeding/aipcp/1610?ver=pdfcov
Published by the AIP Publishing

Articles you may be interested in
Parallel kinetic Monte Carlo simulation framework incorporating accurate models of adsorbate lateral interactions
J. Chem. Phys. 139, 224706 (2013); 10.1063/1.4840395

Non-equilibrium dynamics of Coulomb glass: Kinetic Monte Carlo simulation

Parallel Markov chain Monte Carlo simulations

Comparison of kinetic Monte Carlo and molecular dynamics simulations of diffusion in a model glass former
J. Chem. Phys. 120, 8134 (2004); 10.1063/1.1690241

Parallel Monte Carlo simulations of epitaxial growth
Parallel kinetic Monte Carlo simulation of Coulomb glasses

E.E. Ferrero∗†, A.B. Kolton∗ and M. Palassini∗∗‡

∗CONICET, Centro Atómico Bariloche, 8400 Bariloche, Argentina
†Current address: LIPhy, Université Joseph Fourier, UMR 5588, F-38402 Saint Martin d’Hères, France
∗∗Departament de Física Fonamental, Universitat de Barcelona, 08028 Barcelona, Spain
‡The Abdus Salam International Centre for Theoretical Physics (ICTP), 34014 Trieste, Italy

Abstract. We develop a parallel rejection algorithm to tackle the problem of low acceptance in Monte Carlo methods, and apply it to the simulation of the hopping conduction in Coulomb glasses using Graphics Processing Units, for which we also parallelize the update of local energies. In two dimensions, our parallel code achieves speedups of up to two orders of magnitude in computing time over an equivalent serial code. We find numerical evidence of a scaling relation for the relaxation of the conductivity at different temperatures.

Keywords: Monte Carlo algorithms, Hopping transport
PACS: 02.70.Tt, 71.23.Cq, 72.20.Ee

INTRODUCTION

A frequent limitation of Markov-chain Monte Carlo (MC) methods is low acceptance. In equilibrium MC this problem can sometimes be circumvented with a clever choice of the MC moves, which can be chosen with a certain freedom as long as the Markov chain converges to the probability distribution of interest. Such freedom is not allowed in kinetic Monte Carlo (KMC), in which the MC moves are dictated by the physical dynamics to be simulated.

In this work we present a general “parallel rejection” (PR) algorithm to address the problem of low acceptance, which is especially suitable for implementation on Graphics Processing Units (GPUs), easily available and inexpensive platforms for massively parallel computing. We apply PR to the KMC simulation of the hopping conduction in Coulomb glasses, which is notoriously plagued by very low acceptance in the variable-range hopping (VRH) regime. For this particular application, PR can be seen as a parallelization of the “mixed” algorithm of Refs. [1, 2], which can be further accelerated in GPUs by efficiently parallelizing the N local energy updates that, due to the long-range interaction, are required after each elementary MC move in a system with N sites.

We implemented a GPU code using CUDA [3], exploiting both sources of parallelism. In the next section we illustrate the PR idea in general, and later we apply it to the hopping dynamics of the Coulomb glass. We then present our results for the lattice model in two dimensions, notably on the short-time relaxation of the conductivity. Finally, we compare the performance of the GPU code with a serial implementation of the mixed algorithm.

PARALLEL REJECTION ALGORITHM

Let us consider a generic Markov chain specified by a proposal matrix $Q_{αβ}$ and an acceptance matrix $P_{αβ}$ between the configurations $α, β$ of a certain system. The standard (serial) rejection algorithm to simulate such a chain consists in iterating the following two steps: 1. Propose a move $α → β$, where $α$ is the current configuration and $β$ is chosen with probability $Q_{αβ}$; 2. Accept the move with probability $P_{αβ}$ and, if this is accepted, update the configuration to $β$.

The PR algorithm (see Ref.[4] for a similar idea) runs simultaneously on an ordered array of $M$ parallel “threads” (for example, GPU threads), iterating the following steps ($k = 1, \ldots, M$ is the thread label):

1. Propose, independently from the other threads, a move $α → β_k$, where $α$ is the current configuration (common to all threads) and $β_k$ is chosen with probability $Q_{αβ_k}$;
2. Accept the move $α → β_k$ with probability $P_{αβ_k}$, independently from the other threads (without updating the configuration);
3. If at least one thread has accepted a move, update the configuration to $β_q$, where $q$ is the lowest label among the threads that have accepted a move.
This article is copyrighted as indicated in the article. Reuse of AIP content is subject to the terms at: http://scitation.aip.org/termsconditions. Downloaded to IP: 31.38.137.35 On: Sun, 17 Aug 2014 16:24:47

While the two algorithms above are mathematically equivalent (see Fig. 1(a) for an illustration), the parallel version is increasingly faster as the acceptance rate, \( A = \langle P_{ab} \rangle \), decreases. If \( m \) and \( p \) are the number of iterations of the serial and parallel algorithms, respectively, until a move is accepted, we can estimate the speedup of PR as \( \langle m \rangle / \langle p \rangle \) (neglecting parallelization overheads and differences in the computing time for one iteration in the parallel and serial implementations). Since \( \langle m \rangle = A^{-1} \) and \( \langle p \rangle = 1/[1 - (1 - A)^{m}] \), for \( A \ll 1 \) we have \( \langle p \rangle \sim (MA)^{-1} \ll \langle m \rangle \), and thus \( \langle m \rangle / \langle p \rangle \sim \min(A^{-1}, M) \). In principle, the optimal choice for \( M \) is thus \( M^* = \min(A^{-1}, C) \), where \( C \) is the maximum number of threads that can run simultaneously. In practice, in a GPU implementation it is possible (and recommended) to use \( M \) larger than \( C \), in which case the virtual parallel execution is efficiently handled by the hardware scheduling system. The computing time can then remain sublinear in \( M \) even for \( M \) as large as \( 10^4 \). Thus, for many applications the speedup of the PR algorithm on GPUs is essentially dictated by \( A^{-1} \), and can be very significant.

A popular alternative to the rejection algorithm in low-acceptance situations is the rejection-free BKL or Gillespie algorithm [5]. This requires a computing time proportional to the average number \( z = \langle \sum_{\beta} \theta(Q_{ab}P_{a\beta}) \rangle \) of configurations that are accessible from a given configuration \( \alpha \). Thus, BKL is faster than serial rejection when \( z \ll A^{-1} \), but slower than PR when \( (MA)^{-1} \ll z \). Below we apply PR to the rejection part of the mixed algorithm of Ref. [1], which combines the BKL and rejection algorithms for the simulation of phonon-assisted hopping conduction.

**HOPPING CONDUCTION IN THE COULOMB GLASS**

We consider the standard Coulomb glass model [6] described by the dimensionless Hamiltonian

\[
H = \sum \phi_i n_i + \frac{1}{2} \sum_{i \neq j} \frac{(n_j - \nu)(n_j - \nu)}{r_{ij}} - E \sum n_i x_i, \tag{1}
\]

where \( n_i = 0, 1 \) (\( i = 1, \ldots, N \)), with \( \sum n_i = \nu N \), are the electron occupation numbers of \( N \) sites in a \( D \)-dimensional volume, \( \nu \) is the filling factor, \( r_{ij} \) is the Euclidean distance between sites \( i \) and \( j \) in units of the average spacing \( a = (\text{Volume}/N)^{1/D} \), and \( E \) is an external electric field in the (negative) \( x \) direction. The random potentials \( \phi_i \) are sampled from the uniform distribution \([−W, W]\). We consider only phonon-assisted single-electron hops, with transition rates that can be approximated by

\[
\Gamma_{ij} = \tau_0^{-1} \theta_{ij} e^{-2r_{ij}/\xi} \min[1, e^{-\Delta H_{ij}/k_B T}] \tag{2}
\]

where \( T \) is the temperature, \( k_B \) is the Boltzmann constant, \( \theta_{ij} = \delta_{n_i, 1} \delta_{n_j, 0} \), \( \xi \) is the electron localization length, and \( \tau_0 \) is a microscopic time of order \( 10^{-12} \) s, which will be our unit of time. The energy change after a hop \( i \to j \) is
\[ \Delta H_{ij} = \epsilon_j - \epsilon_i - 1/r_{ij} - E \Delta x_{ij}, \] where \( \epsilon_i = \phi + \sum_j (n_j - v) / r_{ij} \) are the single-particle energies and \( \Delta x_{ij} = x_j - x_i \) is the length of the hop along the field. The instantaneous conductivity is given by \( \sigma(t) = \frac{1}{N E} \frac{dP(t)}{dt} \) (3)

where \( P = \sum n_i x_i \) is the electric dipole moment, provided \( n = \epsilon / T \) is small enough to ensure a linear response.

In the naive serial KMC algorithm for simulating the dynamics in Eq.(2) a hop \( i \rightarrow j \) is proposed by choosing \( i \) and \( j \) uniformly at random among the \( N \) sites, and is accepted with probability \( \tau_0 \Gamma_{ij} \) (thus, \( Q_{ij} = N^{-1} (N-1)^{-1} \sum_k \Theta_{jk} \) and \( P_{ij} = \sum_k \tau_k \Theta_{jk} \Gamma_{ij} \), where \( \Theta_{jk} = 1 \) if \( \alpha \) and \( \beta \) differ by the hop \( i \rightarrow j \) and \( \Theta_{jk} = 0 \) otherwise). The time is incremented by \( \Delta = 1 \) after \( N(N-1) \) proposals.

The above algorithm suffers from extremely low acceptance due to both the tunneling factor \( \Gamma_{ij} = e^{-2r_{ij}/\xi} \) and the thermal activation factor \( \Gamma_{ij} = \tau_0 \theta_{ij} \min[1, \exp(-\Delta H_{ij} / T)] \). The mixed algorithm [2] exploits the factorization \( \tau_0 \Gamma_{ij} = \Gamma_{ij} \Gamma_{ij}^{\dagger} \) and the fact that \( \Gamma_{ij} \) is configuration independent. The proposal matrix is now \( Q_{ij} = \sum_{\alpha\beta} \Theta_{ij} \Theta_{\alpha\beta} \Gamma_{ij}^{\dagger} / \Gamma \), where \( \Gamma = \sum_{\alpha\beta} \sum_{ij} \Theta_{ij} \) and can be sampled without rejection (for example, using the “tower sampling” method [7]). The acceptance matrix is \( P_{ij} = \sum_{\alpha\beta} \Theta_{ij} \Theta_{\alpha\beta} \Gamma_{ij}^{\dagger} \). After each proposal, \( t \) is incremented by a random \( \Delta \) sampled from \( p(\Delta) = \Gamma \exp(-\Gamma \Delta) \) \([5]\). Since the rejection now only comes from \( \Gamma_{ij}^{\dagger} \), the acceptance rate is increased by a factor \( \sim N_\xi^{-D} \). Nevertheless, deep in the VRH regime, where \( T \) is only a few percent of the Coulomb energy, the acceptance is still quite low (for the \( D = 2 \) lattice model we find \( A \approx 0.03 \) for \( T \leq 0.2 \)). It becomes then advantageous to use the PR strategy. This results in the following mixed PR algorithm running on \( M \) threads \( (k = 1, \ldots, M) \), which is mathematically equivalent to the serial mixed algorithm and, therefore, to the naive KMC:

1. Propose (indendently from the other threads) a hop \( i_k \rightarrow j_k \) by choosing \( (i_k, j_k) \) with probability \( \Gamma_{i_kj_k} / \Gamma \) by rejection-free sampling.
2. Accept the hop with probability \( \Gamma_{i_kj_k} \) (even if accepted, do not execute the hop).
3. If at least one thread has accepted a hop then:
   a. Execute the hop \( l_q \rightarrow j_q \), where \( q \) is the lowest label among the threads that have accepted a hop. Any accepted hop in the other threads is discarded.
   b. Update the dipole moment as \( \Delta P = \Delta x_{i_kq} \) and the local energies by adding \( 1/r_{i_kj_k} \) to \( \epsilon_{i_k} \), and \( -1/r_{i_kl} \) to \( \epsilon_{j_k} \) and \( 1/r_{lq} - 1/r_{lq} \) to \( \epsilon_l \) for \( l \neq i_k, l \neq j_q \).
   c. Increment the time by a random \( \Delta \) sampled from a Gaussian distribution of mean \( m/\Gamma \) and standard deviation \( \sqrt{m/\Gamma} \) for at least one thread has accepted a hop.

We implemented the above algorithm in CUDA [3] for the case in which the sites belong to a lattice with toroidal boundary conditions. In this case \( \Gamma_{ij} \) only depends on \( r_{ij} \) (thus \( \Gamma = N \sum \Gamma_{ij} \)) hence the “tower” is much smaller than

\[ |g(\varepsilon, T)| \]

\[ pM + p \]

where \( p \) is the number of iterations since the previous executed hop.

\[ \frac{\sigma(t)}{N E} \frac{dP(t)}{dt} \]
for the model with random sites [2]. At each iteration we need 3M random numbers (to choose $i_{jk}$, $r_{i_{jk}}$, and to accept the hops), which requires a random number generator (RNG) able to generate a large number of uncorrelated random sequences in parallel. The code uses functions of our own and the open-source libraries Thrust [8] (for generic parallel transformations) and Philox from Random123 [9] (a GPU-suited RNG). The code also exploits the “embarrassing parallelism” of the local energy update: we distribute the update on $N$ parallel threads, where the $k$-th thread updates the local energy on lattice site $k$.

RESULTS FOR THE LATTICE MODEL IN TWO DIMENSIONS

We simulated two-dimensional square lattices with $N = L^2$ sites, for $L = 64, 128, 256, 512, 1024, 2048$, setting $v = 1/2$, $\xi = 1$, $W = 1$, and $E = T/10$. We use toroidal boundary conditions in both directions, and adopt the minimal image convention for $r_{ij}$ and $\Delta x_{ij}$. We do not allow hops larger than $L/2$ (in our simulations, the typical hopping length is much shorter than $L/2$ anyway). To recover cgs units from the numerical data below, the dimensionless quantities $\{H, T, \epsilon, P, \sigma\}$ must be multiplied respectively by $\{(\epsilon a^2)/(\kappa a)^2, \epsilon a^2/(\kappa a)^2, \sigma_0, e/(\kappa a^2), e a, a^3/\kappa, T_0\}$, where $e$ is the electron charge and $\kappa$ is the dielectric constant of the lattice.

To validate the code, we start by analyzing the single-particle density of states, defined as the normalized histogram $g(\epsilon, T) = \sum_\epsilon \delta_{N\epsilon}(\epsilon - \epsilon_\epsilon)$, for a given binning size $\Delta \epsilon$. In Fig.2(a) we show $g(\epsilon, T)$ in the steady state for different temperatures and $L = 512$. As shown in the inset, in the Coulomb gap region the data can be rescaled as $g(\epsilon, T) = T f(\epsilon/T)$, and are well fitted by $g(\epsilon, T) = c|\epsilon|$ for small $|\epsilon|$, with $c = 0.62$. The prefactor is consistent with the theoretical estimate [6] $c = 2/\pi$ but is larger than the estimate $c = 0.40$ obtained with the parallel tempering MC algorithm [10], which allows many-electron rearrangements and thus can reach lower energies than the single-particle KMC, even if the latter seems to have reached a steady state, as shown in Fig.2(b).

Next, we analyze the conductivity. The inset of Fig.3(a) shows how, starting from a random configuration, after a transient time the polarization grows linearly in time, and a stationary conductivity can be estimated as $\sigma_0 = P/(N E T)$ for large $t$. Further relaxation of the conductivity at larger times cannot be discarded, but should be negligible for time scales a few times larger than the transient time. Our data for $\sigma_0(T)$, shown in Fig.3(b), agree with Ref.[1] up to a factor two. The data are well fitted by the Efros-Shklovskii law $\sigma_0 = b T^{-\lambda} \exp[-(T_0/T)^{1/2}]$ assuming $\lambda = 1$, which gives $T_0 = 6.2, b = 2.7$. However, the choice $\lambda = 2$ fits equally well the data for $T < 0.25$, giving $T_0 = 9.3, b = 1.7$. As shown in the main figure, the time evolution of $P(t)$ at different temperatures can be collapsed very well onto a single curve by rescaling the time with a characteristic transient time $\tau \sim (\sigma_0 T^2)^{-1}$. This suggests a scaling relation

$$\sigma(t, T) = \sigma_0(T) f(t/\sigma_0(T) T^2).$$

(4)
In this section we compare the performance of our parallel GPU code with that of a serial CPU implementation of the mixed algorithm. In both cases, the wall-clock computing time required to execute a hop is the sum of two main parts: the rejection time, \( W_R \), spent proposing and rejecting hops until one is accepted, and the update time, \( W_U \), spent updating the \( N \) local energies after a hop is executed. As discussed earlier, aside from hardware-related corrections we expect \( W_R \propto A^{-1} \log L \) for the CPU code and \( W_R \propto (AM)^{-1} \log L \) for the GPU code running on \( M \) threads, where the acceptance \( A \) increases with the temperature and the log \( L \) factor comes from the tower-sampling. \( W_U \) is independent of the temperature and the configuration, and we expect \( W_U \propto L^2 \). In the parallel implementation, nevertheless, this will hold only at large enough \( L \) (when the hardware occupancy saturates), while at smaller sizes the scaling will be sublinear in \( L^2 \), and even constant for very small sizes. In the Coulomb glass, most excitations active at low \( T \) are dipoles (short electron-hole pairs). In 2D the density of states of dipoles at low energy is constant, which implies \( A \propto T \) in the steady state (indeed we find \( A \approx 0.03 \, T \)). Hence, the overall computing time will be dominated by \( W_R \) for temperatures below a certain threshold that decreases with \( L \) as \( \log L / L^2 \), and by \( W_U \) above the threshold.

To see how well the above estimates hold in practice, in Fig.4(a) we show the temperature dependence of \( W_R \) for the CPU and GPU codes in the steady state for \( L = 256 \) (the dependence on \( L \) is very weak). For the CPU code, we see roughly \( W_R \propto 1 / T \), as expected. For the GPU code we used a number of threads close to the optimal value \( M^* \approx A^{-1} \). Hence, at moderate temperatures where \( M^* \) is not too large, \( W_R \propto (M^* \, A)^{-1} \) is almost independent of \( T \), as expected. At very low \( T \), \( M^* \) is large enough to saturate the maximum number of concurrent threads and thus \( W_R \) increases with \( 1 / T \), although less than linearly since the GPU can still save time by hiding memory latency. Therefore, the relative performance of the GPU vs the CPU codes, shown in the inset, is consistent with the expected linear behavior in \( 1 / T \).

In Fig.4(b) we compare the size dependence of \( W_U \) for the GPU and CPU implementations. As expected, for the CPU we observe \( W_U \propto L^2 \) while for the GPU, using \( L^2 \) threads, \( W_U \) is almost constant for small \( L \) and grows for large \( L \). Interestingly, the increase is still sublinear even when \( L^2 \) is several times larger than the number of physical cores, due to the efficient internal thread scheduling. Consequently the speedup, which is already substantial even for...
relatively small sizes, increases with \( L \) and exceeds two orders of magnitude at \( L = 2048 \), without quite saturating yet.

The overall computing time per executed hop of the GPU code is basically the sum of \( W_R \) and \( W_U \). For example, for the platform CPU3+GPU3 (see Fig.4 for details) the overall time ranges from 0.1 ms (\( L = 64 \)) to 1.04 ms (\( L = 2048 \)) at \( T = 0.05 \), representing speedups of 1.7x to 179x respect to the serial code, and 0.27 ms (\( L = 64 \)) to 1.23 ms (\( L = 2048 \)) at \( T = 0.001 \), representing speedups of 15x to 157x. Nevertheless, note that these speedup factors depend on the particular serial implementation and hardware.

In order to see in what regime of \( T \) and \( L \) the GPU code may be useful in practice, it is illustrative to estimate the physical time we can simulate in, say, a ten-day run. While this time is independent of \( T \) for the CPU code, according to the previous discussion it scales as \( 1/T \) for the GPU code for low enough \( T \). This is confirmed in Fig.1(b), where we also show the transient time \( \tau = 1/(\sigma T^2) \) for the establishment of a steady-state conductivity. Clearly, since \( \tau \) grows much faster than the speedup as \( T \) decreases, even the GPU code cannot reach the steady state at very low \( T \). At the temperatures at which we can reach the steady state, the typical hopping length \( r \sim (\xi/4)(T_0/T)^{1/2} \) is less than ten lattice spacings. Hence, for the purpose of measuring the conductivity it is preferable to average over many samples at intermediate \( L \), rather than a few samples at large \( L \). Large samples, for which the GPU code is significantly advantageous, might be useful for studying the large-scale geometry of the conducting paths, for instance.

It is worth noting that the GPU code should be significantly more advantageous in 3D than in 2D, because (i) \( N \) increases more rapidly with \( L \), and (ii) both \( T_0 \) and the dipole density of states of dipoles are smaller in 3D, so in the VRH regime the acceptance rate is even lower. It is also straightforward to incorporate multiple-electron hops in our GPU code. Since these hops have even lower acceptance rate, we expect the speedup to be substantial.

**CONCLUSIONS**

We have presented a novel parallel KMC technique for simulating the Coulomb glass hopping conduction. It allows to simulate larger systems and longer times than its serial counterpart, with speedups over 100x for relatively large system sizes in two dimensions. This might be helpful for studying features involving larger length-scales. In 2D, we find that the short-time relaxation of the conductivity at different temperatures is well described by a single scaling curve. Finally, our current implementation can be easily extended to higher dimensions, multiple occupation, and multi-electron hops. For these extensions we can expect an even larger speedup with respect to the mathematically equivalent serial implementation. The code is available to download, modify and use under GNU GPL 3.0 at [13].

**ACKNOWLEDGMENTS**

We acknowledge support from the “Acción Integrada Argentina-España” MICYT-MINECO (Ref. PRI-AIBAR-2011-1206). MP acknowledges support from MINECO (FIS2012-38266-C02-02) and AGAUR (2012 BE 00850). We thank FaMAF-UNC (Argentina), and LIPhy-UJF (France) for computational resources for benchmarking and simulations. MP thanks the CMSP section of ICTP for hospitality.

**REFERENCES**