Large-scale lattice gas Monte Carlo simulations for the generalized Ising model

Tobias C. Kerscher and Stefan Müller
Institut für Keramische Hochleistungswerkstoffe
Technische Universität Hamburg-Harburg
21073 Hamburg, Germany

Quinn O. Snell and Gus L. W. Hart
Brigham Young University
Provo, Utah 84602

Abstract—We present an efficient parallel algorithm for lattice gas Monte Carlo simulations in the framework of an Ising model that allows arbitrary interaction on any lattice, a model often called a cluster expansion. Thermodynamic Monte Carlo simulations strive for the equilibrium properties of a system by exchanging atoms over a long range, while preserving detailed balance. This long-range exchange of atoms renders other frequent parallelization techniques, like domain decomposition, unfavorable due to excessive communication cost. Our ansatz, based on the Metropolis algorithm, minimizes communication between parallel processes. We present this new “partial sequence preserving” (PSP) algorithm, as well as benchmark data for a physical alloy system (NiAl) comprised of one billion atoms.

I. LATTICE GASES

There are many models in science that use a lattice as the domain of the problem: Fluid dynamics ([1], [2], [3], [4]) catalysis simulations ([5], [6]), diffusion modeling ([7], [8]) and many others use a lattice as a framework for computation ([9], [10], [11]). The lattice may be a coarse-grained approach to a continuum situation (as in fluid dynamics simulations) or it may represent individual atomic sites as in a kinetic simulation of diffusion in a solid.

Often the lattice gas simulation consists of particles (atoms or molecules or discrete amounts of a fluid) that move on the lattice. The motion of the particles may be determined by update rules characterized by the occupation of neighboring lattice sites (as in many fluid dynamics simulations). In kinetic and thermodynamic simulations of catalysis or ordering in solids, the configuration of particles on the lattice sites is dictated by physically-motivated interactions. These simulations attempt to determine the time evolution of a system or a steady-state situation, or they may be used to find the minimum free energy configuration.

In this paper, we discuss a parallel algorithm for thermodynamic Monte Carlo (MC) modeling on a lattice. The physical model that we address is a generalized Ising model, called a cluster expansion, although the concepts essential to parallelization are not limited to this case. The specific case that we focus on are scientific problems that require a large lattice (i.e., high number of particles), and many iterations to get statistically reliable results. Whereas in the past, typical cluster expansion-based simulations were limited (both by CPU time and memory) to a few hundred thousand atoms, our algorithm can efficiently treat simulation cells with more than one billion atoms.

A billion atomic sites constrains an efficient algorithm to have a small memory footprint (to store the current configuration) and also to be fast (or at least efficiently parallelizable). In some cases, domain decomposition is an efficient way to parallelize the simulations but our algorithm is applicable when the updates on the lattice take place at arbitrarily long distances, so domain decomposition would create all-to-all communication and thus is inappropriate. We note, however, that both algorithms can supplement each other if needed.

II. FREE ENERGY AND THE ISING MODEL

One application of the generalized Ising model is to simulate alloys and we use an alloy example to illustrate the algorithm. Each lattice site is occupied by one atom and there are at least two types of atoms. The atomic interactions may be fairly long-ranged, extending over a dozen lattice sites or so, and usually include “multibody” interactions (i.e., beyond merely pair interactions). We term this range the “sphere of influence” for a given atom, which will be discussed in the next section.

In the most common application of the model, the purpose of simulation is to find the equilibrium thermodynamic state. In other words, the simulation is searching for configurations that minimize the free energy

\[ F = U - TS, \]

where \( U \) is the chemical interaction (due to the interaction of atoms), \( T \) is the temperature, and \( S \) is the configurational entropy. At finite temperatures, the two terms are in competition: \( U \) drives the system towards order while \( -TS \) minimizes \( F \) when the system is disordered. \( F \) is minimized by finding the appropriate “compromise” between the two competing terms. Monte Carlo simulations are used to find the average configuration that minimizes \( F \).

Cases that can be addressed by a small simulation cell are not appropriate for this algorithm.
By way of illustrating the model² and minimizing the free energy, consider an Ising model in two dimensions with \( N \) sites, usually denoted by \( i, j \in \{1, \ldots, N\} \). Let half of the sites be occupied by A atoms and the other half by B atoms, represented by the spin variables \( \sigma_i = +1 \) and \( \sigma_i = -1 \), respectively. Let us restrict the interactions to a single pair interaction between nearest neighbors only. The interaction strength is characterized by a parameter \( J \). The chemical part of the free energy (from Eq. 1) then is

\[
U = -\frac{J}{N_p} \sum_{p_{i,j}} \sigma_i \sigma_j ,
\]

where the sum is over all \( N_p \) nearest neighbor pairs \( p_{i,j} \) on the lattice. At low temperatures (as \( T \) tends toward zero), the contribution from the \( -TS \) term is negligible and \( F \) is minimized by the atomic configuration that minimizes \( U \).

By inspection one can see that, if \( J \) is negative, the configuration that minimizes \( U \) (and also \( F \) at low \( T \)) is that shown in the left hand side of Fig. 1. In that case, every term in the sum is \( \sigma_i \sigma_j = (\pm 1)(\mp 1) = -1 \) and

\[
U = -\frac{J}{N_p} N_p -1 = -\frac{J}{N_p} N_p = J .
\]

On the other hand, if \( J \) is positive, the minimum energy configuration clusters alike atoms together as shown on the right in the figure (phase-separating case like “oil and water”).

In contrast, at finite temperatures, the configurations that minimize the free energy will mimic what we normally observe in nature: there will be some disorder. The competing terms \( U \), which drives the system towards order, and \( -TS \) which drives the system to states of high entropy, strike a balance depending on \( T \) to minimize \( F \). Most systems will undergo a sudden order-disorder phase transition at a specific temperature (like the ice ↔ liquid water transition). Only in the simplest cases can these transitions be identified by analytical means ([13], [14]). For virtually all other systems, MC modeling is used to determine the critical temperature of these transitions. Such information is useful in materials science to engineer new materials [15]. Another crucial problem that can be addressed by these kinds of simulations is precipitate growth, where “bubbles” of a single phase, embedded in the alloy matrix, have a profound impact on the materials properties ([16], [17]).

### III. Generalized Ising Model and Alloy Simulation

In the original Ising model, there was only one interaction (between nearest neighbors) and only two possible values (±1; originally called spins) at each lattice site. But in alloy simulations the situation is much more complicated:

- The possible values of each lattice site (the spins) may extend beyond the binary case. (Our example, discussed below in the benchmark section, is a ternary case with three spin values.³)
- There may be multiple pair-interactions and they may extend well beyond nearest neighbors.
- There may be higher-order interaction terms which include more than two vertices (i.e., the model is not restricted to merely pairwise interactions) [18].
- The lattice of sites may not be a simple lattice but a multilattice with more than one lattice site per unit cell ([20], [21]).

This kind of generalized Ising model is usually called a cluster expansion because the energy \( U \) of Eq. 2 can be generalized to higher order “clusters” (rather than just pairwise interactions):

\[
U = \sum_C J_C \Pi_C ,
\]

which is an expansion of \( U \) in terms of pair, triplet and other multi-body clusters of interaction, \( C \); the \( \Pi_C \) denote the appropriate spin products. For a detailed description, please refer to Refs. [18], [22], [23], [24], [25], [26], [27] and references therein.

In most lattice gas simulations, the spatial extent of the interactions is small relative to the simulation cell. To calculate the change in energy \( \Delta U \) when two particles are exchanged, only sites that are closer than the maximal interaction distance need to be included in the sum Eq. 4: Each atom has a limited “sphere of influence” as depicted in Fig. 2.

In some problems, the atomic sites in the simulation cell may not be on a regular grid because the simulation

---

²A more in-depth but accessible discussion of the model and how it can be implemented can be found in Ref. [12].

³In those “multinary” cases, the structure of the spin products, as given in Eq. 2, has to be changed too: instead of multiplying the spins itself, orthogonal functions of the spins are used, e.g., Chebychev polynomials: \( \sigma_i \sigma_j \rightarrow T_m(\sigma_i)T_n(\sigma_j) \) ([18], [19]).
cell may not be a simple multiple of the “parent cell”. (A common case is the body-centered cubic lattice where the parent cell is not a cube.) An example is shown in Fig. 3 where the vectors of the red cell define the lattice but the blue simulation cell is not a simple multiple of the parent cell. Consequently, the rows and columns of lattice points in the simulation cell do not form a rectangular grid inside the cell, but because the data structure for the parallel algorithm (discussed in the next section) needs to be an orthorhombic grid, we need a simple way to transform the physical simulation cell to an orthorhombic one.

The simulation cell can be transformed to a rectangular grid of points by invoking several ideas from the theory of permutation groups. Detailed discussion can be found in Refs. [28, 21]. Briefly, the simulation cell can be represented by a $3 \times 3$ integer matrix in Hermite normal form (or a $2 \times 2$ matrix in this two-dimensional example). This matrix in turn can be transformed into Smith normal form. The corresponding transformation can be applied to the lattice points inside the simulation cell to map them onto an orthorhombic grid. This transforms the shape of the “sphere of influence” as well, but it will maintain either a cubic or a rectangular form.

### IV. Parallel Algorithm

#### A. Preserving detailed balance

In a thermodynamic simulation of a generalized Ising model, two atoms (of different types) are selected at random (irrespective of their interatom distance) and swapped. Swapping continues until a configuration is reached that minimizes the free energy of the system.

Because the atoms can be arbitrarily far apart, domain decomposition is not an effective parallelization strategy as it would introduce excessive all-to-all communication overhead that would overwhelm computation. On the other hand, one may naively think to merely make a long list of pairs to be swapped, break the list into pieces, and distribute the pieces to different processors. However, simple parallelization over the list does not reproduce the correct thermodynamic behavior because it violates the principle of detailed balance (discussed below). As a result, the algorithm will fail to identify the configuration which minimizes the free energy (Eq. 1). Maintaining detailed balance while parallelizing over a list of candidate swaps is the primary aim of the algorithm.

To understand what the algorithm must do to maintain detailed balance, let’s first examine the sequential algorithm. It proceeds using the well-known Metropolis algorithm [29], as follows:

- randomly select a pair of atoms
- swap the two atoms
- determine the change in energy (that is, determine $\Delta U$ (in Eq. 2) that results from the swap)
- if energy is lowered ($\Delta U$ negative), keep the swap
- otherwise, keep the swap only with a probability $\exp(-\Delta U/k_B T)$.\(^4\) This probability is referred to as the Boltzmann factor.
- repeat until convergence (i.e., until the fluctuations in $\Delta U$ are only statistical deviations)

In the sequential algorithm, each pair of atoms that are selected can be considered independent and the energy change, $\Delta U$, can be safely calculated because the whole system is static and no other changes are being made. Thus, the atoms within a given pair’s sphere of influence that the calculation depends on are not changing. A parallel algorithm, on the other hand, creates situations where multiple atom pairs are being swapped simultaneously. This can lead to inconsistency if the pairs overlap with each other (i.e. are within each other’s sphere of influence) because the order

\[ k_B \] is Boltzmann’s constant and $T$ is the temperature in units of Kelvin.

---

\(^4\)
in which the swaps are made changes the outcome and the simulation becomes history dependent and does not maintain detailed balance.

Detailed balance means that if two states are equally likely at equilibrium, then the rate of transition \( W \) from state 1 to state 2 must be the same as the rate of transition from state 2 to state 1, i.e., \( W(1 \rightarrow 2) = W(2 \rightarrow 1) \). If the probability of the two states are not the same, then detailed balance requires that the ratio of the transition rates be equal to the inverse ratio of the relative probabilities. As mentioned above, if the sites of two consecutive pairs lie within each other’s sphere of influence, then the order in which the pairs are swapped will generally affect the respective values of \( \Delta U \). In other words, the computed Boltzmann factors can be inconsistent with the swaps that are made because the order of the swaps will in general affect the final configuration. In the parallel algorithm one must preserve the order in which swaps are made even though the computations have been distributed across different processors. We do this not by a decomposition of the spatial domain but by a decomposition in time that preserves the essential parts of the swapping order.

\[ \text{B. Parallelization} \]

The partial sequence preserving (PSP) algorithm leads to choices in data and calculation distribution. The principle calculation is swapping the atoms and determining the change in energy. Indeed, this is where all the time is being spent in the sequential algorithm. The main data structure that is maintained is a lattice. Each point of the lattice stores a number (usually \( +1 \) or \( -1 \)) that represents the type of atom at that location. This configurational information is needed to calculate chemical energy, \( U \), of the system. This data structure may be distributed or replicated.

If the lattice is distributed (domain decomposition), a processor may generate a pair of atoms to be swapped that cross processor boundaries. This creates communication dependencies as well as control dependencies. In order to eliminate the communication dependencies, the data structure may be replicated. However, the control dependencies still exist. When a pair of atoms are being swapped, all other atoms within the “sphere of influence” of those atoms must not be changed until it is determined whether or not the atoms will be swapped. Otherwise the energy calculation will not be accurate (because detailed balance is violated).

Candidate atom pairs can be generated independently using a parallel random number generator to avoid duplication and maintain uniformity in the random number distribution. However, each pair of candidate atoms cannot be swapped until it is verified that the pair are independent from all other pairs, that is, they are not affected by any other pair being considered at the time. The communication overhead to determine independence would overwhelm the increase in performance due to parallel energy calculations.

However, in a large system, many of the generated candidate pairs of atoms are completely independent (i.e., they do not fall within each others spheres of influence). The larger the system, the more likely two pairs will be independent. The change in energy may be safely calculated simultaneously because the spheres of influence do not intersect. This fact leads to the proposed parallel algorithm. Rather than using a parallel random number generator so that each processor generates different candidate atom pairs, each processor generates the same pairs of atoms. This allows dependency checking to take place during generation without communication overhead.

As atom pairs are generated, dependency information is recorded in a dependency grid and the atom pair is placed in a list based on its dependency on other atom pairs. Figure 4 shows two atom pairs that have been selected and placed in List 0 which indicates they have no dependencies on other atom pairs. As the atom pair is selected, the cells in the sphere of influence are marked. This count indicates the number of atom pairs that must be swapped before this cell can be swapped. The count then indicates which list a pair should be placed in if that cell is selected. Thus, if the atom at cell \((7,2)\) is selected, that pair should be in list 1 unless the other atom in the pair indicates a different list such as cell \((7,4)\). The pair is always placed in the highest numbered list so that all dependencies are resolved in the correct order. In essence, these dependency-lists maintain the partial order of the atom pairs.

Once the atom pair lists are generated, all atom pairs in List 0 may be swapped simultaneously because they do not depend on the outcome of the swapping of any other pairs that are in list 0. The list can be partitioned among the processors quite simply since all processors have the list. Once List 0 has been processed, List 1 may be similarly processed and so on. In practice, the lists are rolled up and more atom pairs are generated after List 0 has been processed. The entire algorithm is summarized in Fig. 5.

\[ \text{V. Benchmarks} \]

We apply our algorithm to a ternary lattice-gas system: a cluster expansion (CE) for the nickel-aluminum alloy (Ni-Al) for a nickel concentration slightly less than 50\%. The 50-50 mixture crystalizes on a body-centered cubic lattice in the so-called B2 phase, Fig. 6 (left and middle), which substantially increases the melting point of the alloy; hence its industrial importance. This stable B2 phase remains stable even if the nickel concentration is somewhat decreased: then, vacancies build up on the Ni sublattice, Fig. 6 (right), thus lowering the Ni concentration and leaving the Al sublattice unchanged. Consequently, our simulation deals with a ternary system comprised of Ni, Al and vacancies.

The following benchmark data are taken from Monte Carlo simulations for an alloy consisting of 48% nickel, 50% aluminum and 2% vacancies, viz. Ni\(_{48}\)Al\(_{50}\). Our ternary
The performance of the code is dependent on the overhead of managing the dependency list data structures. There are two parameters that directly affect the performance of the code: dependency grid granularity and the number of dependency lists. As stated earlier, the dependency grid is marked to indicate a dependency on atoms within a given sphere of influence. Keeping the grid updated can be time consuming. The time is lessened by making the grid more coarse-grained (each cell represents multiple cells),

corresponds to 50% Ni and 50% Al. Note that periodic continuation of the cube leads to a cubic Ni sublattice and a cubic Al sublattice that permeate each other. If one traverses the crystal from top to bottom, one finds an alternating sequence of Ni and Al layers. Right: A modest decrease in the Ni concentration results in vacancy formation on some cube edges: an atom is missing there, the box represents a vacancy.

Figure 4. The dependency grid and dependency lists preserve the partial-order of the atom pairs. As atom pairs are generated, the cells within its sphere of influence are marked. The pair is placed in a list based on its dependency on other atom pairs.

Figure 5. The partial sequence preserving algorithm. The dashed boxes symbolize administrative tasks, like the initial setup of the simulation and the synchronization afterwards. During these steps, communication between the processes is necessary. The bold box is repeated until convergence is achieved (cf. the Metropolis rules in section IV-A).

Figure 6. Schematic view of the B2 phase of NiAl. Left: The body-centered cubic (bcc) lattice is the basis for the B2 phase. Middle: The lattice points are decorated with Ni (grey) and Al (green) atoms. This arrangement corresponds to 50% Ni and 50% Al. Right: A modest decrease in the Ni concentration results in vacancy formation on some cube edges: an atom is missing there, the box represents a vacancy.

The performance of the code is dependent on the overhead of managing the dependency list data structures. There are two parameters that directly affect the performance of the code: dependency grid granularity and the number of dependency lists. As stated earlier, the dependency grid is marked to indicate a dependency on atoms within a given sphere of influence. Keeping the grid updated can be time consuming. The time is lessened by making the grid more coarse-grained (each cell represents multiple cells),
serial part of the code cores, measured in Monte Carlo flips per second. The fit of Amdahl's N candidate pair is generated that depends on the number of atom pairs the candidate pair depends on. When generated and placed in the appropriate list indicating the dependencies, the number of lists that are maintained affects performance by changing the number of atoms in the dependencies. The number of lists that are maintained and this also reduces the memory footprint for monitoring the dependencies. The number of lists that are maintained affects performance by changing the number of atoms in List 0.

As the simulation proceeds, candidate atom pairs are generated and placed in the appropriate list indicating the number of atom pairs the candidate pair depends on. When a candidate pair is generated that depends on n other pairs where n is the number of lists being maintained, the atom pair generation terminates and atom flipping begins. When the number of lists is small (n ≤ 2) the odds of generating a candidate atom pair that will stop the atom pair generation is higher than when the number of lists is larger (n ≥ 4). Thus more atoms can be generated that will fall into List 0 that can be processed in parallel which amortizes the overhead and improves performance. Figure 9 shows the relative performance per core, \( P_r(L; n) / N \), for various sizes of the dependency boxes, as given by the box length \( L \), and the number of dependency lists, \( n \), that are maintained. The best performance is achieved with greater than 5 dependency lists, \( n > 5 \), and a dependency box size of 10 or 20.

At any given point in the algorithm, the number of atom pairs in List 0 determines the amount of work that can be done in parallel. Since the algorithm relies on random list pair generation, the number of atom pairs in List 0 will vary randomly. Figure 10 shows this distribution on the coarse-graining of the dependency box (given by its length \( L \)) and the number of dependency lists \( n \). \( P_r(L; n) / N \) is normalized to have maximal value 1. The sweet spot of performance is for \( L = 10 \) or \( L = 20 \).

![Figure 7](image-url) Close-up of a plane taken from a Monte Carlo simulation for Ni\textsubscript{48}Al\textsubscript{52} at \( T = 500 \) K. The total system size was \( 1000 \times 1000 \times 1000 = 10^9 \) sites, 48% of which are populated by nickel atoms (black), 50% aluminum atoms (gray) and 2% vacancies (white). The size of the close-up shown in the figure is \( 250 \times 160 \) sites. The simulation confirms the assertion of fig. 6: the vacancies build up in the Ni domains, while the Al domains are free of vacancies. Furthermore, the vacancies nicely align along distinguished crystal orientations. Note that the visible domains of Ni or Al are in-plane domains and do not form three-dimensional domains along distinguished crystal orientations. Behind (and in front of) the Ni or Al are in-plane domains and do not form three-dimensional domains.

![Figure 8](image-url) Number of cores \( N \) vs. relative performance \( P_r(N) := P(N)/P(1) \); \( P(N) \) is the actual performance of the code running on \( N \) cores, measured in Monte Carlo flips per second. The fit of Amdahl’s law yields the serial part of the code \( p_s = 0.2\% \) and the communication part \( p_c = 2.4 \times 10^{-6} \). Parameters of the simulation: system size \( = 1000 \times 1000 \times 1000 = 10^9 \) sites, number of dependency lists \( n = 8 \), dependency box length \( L = 20 \).

![Figure 9](image-url) Relative performance per core, \( P_r(L; n) / N \), depending on the coarse-graining of the dependency box (given by its length \( L \)) and the number of dependency lists \( n \). \( P_r(L; n) / N \) is normalized to have maximal value 1. The sweet spot of performance is for \( L = 10 \) or \( L = 20 \).

![Figure 10](image-url) Distribution of the number of entries in List 0 in a Monte Carlo simulation for Ni\textsubscript{48}Al\textsubscript{52} at \( T = 500 \) K. The total system size was \( 1000 \times 1000 \times 1000 = 10^9 \) sites, number of dependency lists \( n = 4 \), depending on \( n \) and the time spent in generating the dependency lists, \( t_L \), and the time spent in the actual parallel loop, \( t_p \). This ratio represents the cost of the overhead associated with the parallel algorithm. As the value approaches 1, the overhead approaches the computation time and performance is low. As the ratio approaches 0, overhead is low and performance is high. For this experiment using 240 processors running a = \( 1000 \times 1000 \times 1000 = 10^9 \) atom simulation, the average number of entries in List 0 is \( \bar{n}_0 = \left( 8.3 \pm 3.0 \right) \times 10^3 \) and the average ratio is \( \bar{\tau} = 0.36 \pm 0.08 \). Thus, on the average, overhead is fairly low, which allows for good scaling and performance. In general, there is an inverse correlation between the number of entries in List 0 and the ratio \( \tau \). More entries in List 0 amortizes the overhead associated with list generation and communication.
Lastly, Fig. 11 shows the behavior of the ratio $\tau$ versus the number of entries in the 0-dependency-list (List 0) for three different numbers of dependency lists: $n = 2$ (black dots), $n = 4$ (red dots), and $n = 9$ (blue dots). The step-like structure is due to the discretization of the 0-dependency list. The increase in $\tau = t_L/t_p$ before the kink is due to the steady increase in list generation time, $t_L$, as the list size, $n_0$, is growing. The kink itself points at a discontinuity in processing time, $t_p$, whenever $n_0$ hits a multiple of the number of parallel processes (here: 30 nodes at 8 cores = 240 processes). Crossing this frontier, at least one of the processes has to enter an additional loop, hence the jump upwards in processing time, $t_p$, causing a sudden drop in $\tau$. Note that as the number of dependency lists increases, two effects are seen: the number of entries in List 0 increases and the step-like structure is less noticeable as the unbalanced processing is lessened.

VI. CONCLUSIONS

This research demonstrates large simulations (many lattice sites) of lattice gases using parallel computing via a partial sequence preserving (PSP) algorithm. In some lattice gas models, the strategy of domain decomposition is effective, but in cases with long-range interactions between lattice sites and where the sites that are far apart are swapped (as in thermodynamic simulations), domain decomposition is an ineffective parallel strategy. Instead, we decompose the problem by letting different processors handle the swapping operation in parallel. Nevertheless, our PSP algorithm and domain decomposition techniques can be combined in a straightforward manner if required.

To maintain detailed balance, to get the thermodynamics correct, the partial order of the swaps must be maintained even though they are performed by different processors. Thus, if the sites that are swapped by different processors “overlap”, then the order of the swaps affects the outcome of the Metropolis algorithm. To avoid overwhelming the computation by the overhead of communication between processors to maintain swapping order, we create a data structure that maintains the partial-order of the swaps and does not require communication between processors but is still small enough that each processor can have a complete copy. Different parameters of the data structure can be tuned to maximize efficiency. The approach is not efficient on small simulation cells but becomes more and more efficient with increasing cell size.

With this algorithm, we have done simulations containing one billion atoms. This is large enough that atomistic simulations of mesoscopic physics are possible. We modeled the Ni-Al superalloy with explicit treatment of vacancies as a third component to illustrate the PSP algorithm. This alloy is the primary component in high-performance jet engine turbine blades and is under intense scrutiny by the materials science community. Increases in the performance turbine blades has dramatic economic impact. With large simulation cells we can decrease the statistical uncertainty in some important quantities of an alloy modeled by the cluster expansion. The algorithm will also be useful in other cluster expansion modeling such as the growth and kinetics of precipitate formation.

ACKNOWLEDGMENT

G. L. W. H. gratefully acknowledges financial support from the National Science Foundation through Grant Nos. DMR-0650406 and DMR-0908753. T. C. Kerscher and S.
Müller would like to thank the German Research Foundation (DFG) for Grant No. Mu 1648/3. The authors are grateful for extensive use of the Fulton Supercomputing Laboratory at Brigham Young University.

REFERENCES


