Parallelization of the Wolff single-cluster algorithm

J. Kaupužs* and J. Rimšāns
Institute of Mathematics and Computer Science, University of Latvia, 29 Rainja Boulevard, LV-1459 Riga, Latvia

R. V. N. Melnik
M²NeT Laboratory, Wilfrid Laurier University, Waterloo, Ontario, Canada N2L 3C5

(Received 30 April 2009; revised manuscript received 12 January 2010; published 3 February 2010)

A parallel [open multiprocessing (OpenMP)] implementation of the Wolff single-cluster algorithm has been developed and tested for the three-dimensional (3D) Ising model. The developed procedure is generalizable to other lattice spin models and its effectiveness depends on the specific application at hand. The applicability of the developed methodology is discussed in the context of the applications, where a sophisticated shuffling scheme is used to generate pseudorandom numbers of high quality, and an iterative method is applied to find the critical temperature of the 3D Ising model with a great accuracy. For the lattice with linear size $L=1024$, we have reached the speedup about 1.79 times on two processors and about 2.67 times on four processors, as compared to the serial code. According to our estimation, the speedup about three times on four processors is reachable for the $O(n)$ models with $n \geq 2$. Furthermore, the application of the developed OpenMP code allows us to simulate larger lattices due to greater operative (shared) memory available.

DOI: 10.1103/PhysRevE.81.026701 PACS number(s): 05.10.Ln, 75.10.Hk, 05.50.+q

I. INTRODUCTION

The cluster algorithms are very useful in Monte Carlo (MC) simulations of the lattice models near criticality, where the usual Metropolis algorithm suffers from the problem of critical slowing down. The Swendsen-Wang and Wolff cluster algorithms [1–3] are well known. In distinction from the Metropolis algorithm, they ensure reasonably small autocorrelation times for large but finite lattices even at the critical point [4–7] and therefore are widely used in MC simulations of lattice models. For example, the Wolff algorithm has been recently used in simulations of the Ising spin model in $d=5$, 6, 7, and 8 dimensions [8], as well as in the MC study of critical interfaces in the random anisotropy models [9] and random-bond Potts model [10]. With a slight modification, it has been recently applied to the study of Goldstone mode singularity in the XY model [11,12]. Apart from the spin models on regular lattices, it has been successfully used also in MC simulations on Barabasi-Albert networks [13]. In [14] the application of the cluster algorithms to the quantum spin systems has been considered. In fact, there is a no-go theorem that prevents the construction of an efficient Wolff-type embedding algorithm in the standard Wilson formulation for such models, but it is has proven to be possible in the framework of D theory [14].

The famous Wolff single-cluster algorithm [1] is particularly effective in the simulation of equilibrium properties. However, even using the cluster algorithms, the study of critical-point properties with a reliable determination of the critical exponents may require quite large computational times and resources. A parallelization of the computer code is a common method to reduce the computation time. A trivial parallelization, by running several independent MC realizations simultaneously, can provide a satisfactory solution of the problem in many applications. In fact, it does not require any parallel code. However, in applications where each of the runs should be long enough it may be useful to speed up any of them by parallelizing the simulation code. The Swendsen-Wang algorithm is relatively easy to parallelize (see, e.g., [3]) as compared to the Wolff algorithm. However, it is known that the Wolff single-cluster algorithm can even be several times more efficient in the simulation of equilibrium properties (e.g., [15]). Therefore the development of an efficient parallel version of this algorithm represents an important step forward. Two versions of the parallel Wolff cluster algorithm, applied to the two-dimensional (2D) Ising model, have been already proposed and tested in [15]. Unfortunately, due to the irregular size, shape and position of the Wolff clusters, this method does not easily lend itself to efficient parallel implementation. One of the parallel versions proposed in [15] gave fairly good performance. The basic idea of this parallelization is to split the lattice into N partitions (strips), where N is chosen to be an integer multiple of the number of processors P to ensure load balancing. Each partition contains W columns and partition M is assigned to processor M mod P. In such a way, each of the processors controls certain fraction of the lattice, and a data exchange between processors is necessary to treat correctly the spins on the partition boundaries. The larger is W the less communications are necessary, but the load balancing is better for smaller W. The speedup around 10 with efficiencies around 35% have been reached by this algorithm at the critical temperature of 2D lattices with linear sizes greater than $L=512$ [15].

Note that the average size of the Wolff cluster at the critical temperature is $C \sim L^{2-\eta}$ [1,3], where η is the critical exponent, which is exactly 1/4 for the 2D Ising model and has a remarkably smaller positive value in the three-dimensional (3D) case. Hence, the Wolff cluster typically occupies the fraction $C/L^d \sim L^{2-d-\eta}$ of the lattice in d dimensions. Namely, this fraction decays with L as $\sim L^{-1/4}$ in two dimensions and slightly faster than L^{-1} in three dimensions. It

*kaupuzs@latnet.lv

means that, in contrast to the 2D case, the Wolff clusters near the critical temperature will typically occupy only a very small fraction of the whole lattice in the 3D case, as it is the case for large lattices with $L \sim 1024$ considered in our study. It may cause certain difficulties in reaching an acceptable load balancing and efficiency for such 3D lattices, using the algorithm of [15]: a good load balancing could be reached only at remarkably smaller values of W than in the 2D case, but smaller values of W imply larger time losses for communications. The algorithm of [15] refers to the so-called message passing interface (MPI) parallelization technique, where each processor works with its own memory, which is related to certain fraction of the lattice in the MC simulations. We present a parallel implementation using the well-known OpenMP technique, which works with shared memory. In this case we do not use any fixed separation of the lattice, but only a separation of the current list of the lattice sites belonging to the wave front of the growing cluster.

II. ITERATIVE SIMULATION METHOD

As an example, we consider the 3D Ising model on simple-cubic lattice with the Hamiltonian H given by

$$H/T = -\beta \sum_{\langle ij \rangle} \sigma_i \sigma_j, \tag{1}$$

where T is the temperature measured in energy units, β is the coupling constant ($\beta = J/T$, where J is the interaction energy), and the summation takes place over all pairs $\langle ij \rangle$ of the neighboring spins $\sigma_i = \pm 1$ in the 3D lattice with periodic boundary conditions. A spin configuration $\{\sigma\}$ in Eq. (1) appears with the probability

$$\mathcal{P}(\{\sigma\}) = \frac{1}{Z} e^{-H(\{\sigma\})/T} \tag{2}$$

according to the Boltzmann distribution, where $Z = \Sigma_{\{\sigma\}} \exp[-H(\{\sigma\})/T]$ is the partition function. In our MC simulation, we measure the dimensionless (normalized to J) energy per spin $\varepsilon = N^{-1} \Sigma_{\langle ij \rangle} \sigma_i \sigma_j$ and magnetization $m = N^{-1} \left| \Sigma_{i=1}^{N} \sigma_i \right|$ and determine averages of the type $\langle \varepsilon^k m^l \rangle$, where $N = L^3$ is the total number of spins.

In studying the critical-point phenomena it can be useful to design an algorithm which automatically finds the critical point. Such an algorithm, known as the invaded cluster algorithm, has been proposed in [16,17]. Recently, an algorithm with similar properties, called the locally converging Wolff algorithm, has been considered [18]. We present an algorithm, which allows to find iteratively certain pseudocritical coupling $\tilde{\beta}_c(L)$, converging to the true critical coupling β_c at $L \to \infty$. In our case, the fluctuation amplitude for the coupling can be well controlled and, in principle, reduced to an arbitrarily small value. Besides, our method allows to recalculate the simulation results for any slightly different from $\tilde{\beta}_c(L)$ coupling.

The pseudocritical coupling corresponds to a given value U_0 of $U = \langle m^4 \rangle / \langle m^2 \rangle^2$. The quantity U is related to the Binder cummulant $1 - U/3$, which is zero in the high-temperature phase ($\beta < \beta_c$) and 2/3 in the low-temperature phase ($\beta > \beta_c$) at $L \to \infty$. Consequently, $\tilde{\beta}_c(L)$ tends to β_c at $L \to \infty$ for any $1 < U_0 < 3$. At $\beta = \beta_c$, the ratio U tends to certain universal value $U^* \approx 1.6$ [19,20] when $L \to \infty$. Therefore we have chosen $U_0 = 1.6$ to obtain pseudocritical couplings closer to β_c. At a given L, we use the Newton's iterations

$$\tilde{\beta}_c^{(n+1)} = \tilde{\beta}_c^{(n)} - \frac{U - U_0}{\partial U / \partial \beta} \tag{3}$$

to find $\tilde{\beta}_c$, where $\tilde{\beta}_c^{(k)}$ ($k = n, n+1$) is the value of $\tilde{\beta}_c$ in the kth iteration, whereas U and $\partial U / \partial \beta$ are estimated from the simulation of certain MC steps in the nth iteration at $\beta = \tilde{\beta}_c^{(n)}$. The first iteration has been used only for equilibration of the system at a reasonably chosen $\beta = \tilde{\beta}_c^{(1)}$, setting $\tilde{\beta}_c^{(2)} = \tilde{\beta}_c^{(1)}$ afterwards. In a few following iterations, $\tilde{\beta}_c^{(n)}$ reaches $\tilde{\beta}_c$ within the statistical error and further fluctuates around this value. In principle, the fluctuation amplitude δ can be reduced to an arbitrarily small value by increasing the number of MC steps in one iteration $\mathcal{N}_{\mathrm{MCS}}$. The pseudocritical coupling can be estimated by averaging over $\tilde{\beta}_c^{(n)}$, discarding some first iterations. Quantities $\langle \varepsilon^k m^l \rangle$ at $\beta = \tilde{\beta}_c$ also can be evaluated by such an averaging over their values estimated in a set of iterations. However, such a method gives some systematic errors of order δ^2 since corrections of this order have been neglected in the Newton's iterations. Such errors, however, always tend to zero at $\mathcal{N}_{\mathrm{MCS}} \to \infty$.

For a refined estimation, we expand $\ln \langle \varepsilon^k m^l \rangle$, evaluated in each iteration, in the Taylor series around the current $\tilde{\beta}_c^{(n)}$ using Eqs. (1) and (2). It allows us to estimate the values of $\langle \varepsilon^k m^l \rangle$ at any given β near $\tilde{\beta}_c$, taking into account also the nonlinear expansion terms. The averaging over a set of iterations then gives us refined estimates of $\langle \varepsilon^k m^l \rangle$ at this β. Obviously, we can determine from this calculation any secondary quantity like U at the considered β. In this way, we find also the value of $\beta = \tilde{\beta}_c$ at which $U = U_0$ holds. We have Taylor-expanded $\ln \langle \varepsilon^k m^l \rangle$ instead of $\langle \varepsilon^k m^l \rangle$, since it leads to somewhat more symmetric and elegant formulas. Namely, for any quantity x measured in MC simulation we obtain

$$\frac{\partial \ln \langle x \rangle}{\partial \beta} = N \left(\langle \varepsilon \rangle - \frac{\langle x\varepsilon \rangle}{\langle x \rangle} \right), \tag{4}$$

$$\frac{\partial^2 \ln \langle x \rangle}{\partial \beta^2} = N^2 \left[\langle \varepsilon \rangle^2 - \langle \varepsilon^2 \rangle - \left(\frac{\langle x\varepsilon \rangle}{\langle x \rangle} \right)^2 + \frac{\langle x\varepsilon^2 \rangle}{\langle x \rangle} \right], \tag{5}$$

$$\frac{\partial^3 \ln \langle x \rangle}{\partial \beta^3} = N^3 \left[2\langle \varepsilon \rangle^3 - 3\langle \varepsilon \rangle \langle \varepsilon^2 \rangle + \langle \varepsilon^3 \rangle - 2\left(\frac{\langle x\varepsilon \rangle}{\langle x \rangle} \right)^3 + 3\frac{\langle x\varepsilon \rangle \langle x\varepsilon^2 \rangle}{\langle x \rangle^2} - \frac{\langle x\varepsilon^3 \rangle}{\langle x \rangle} \right]. \tag{6}$$

Note that it holds for any model with $H/T = -\beta N \varepsilon$, obeying Boltzmann statistics (2).

The fluctuation amplitude δ must be small enough to ensure a fast convergence of the Taylor expansions. We have reached it by making sufficiently large number of MC steps

TABLE I. The values of $\tilde{\beta}_c$ depending on L, estimated from different sets of iterations (numbered by n) at the simulation parameters given in columns 5–7.

L	$\tilde{\beta}_c(n=3,4)$	$\tilde{\beta}_c(n=5,6)$	$\tilde{\beta}_c(n\geq 5)$	$\mathcal{N}_{mes}$	$\mathcal{N}_{run}$	$\mathcal{N}_{it}$
1024	0.221654620(38)	0.221654598(51)	0.221654628(24)	33000	8	48
864	0.221654603(61)	0.221654633(53)	0.221654640(27)	33000	8	72
768	0.221654550(70)	0.221654708(34)	0.221654669(29)	33000	6	72
640	0.221654581(60)	0.221654583(68)	0.221654615(31)	33005	12	108
512	0.22165457(21)	0.22165491(13)	0.221654662(45)	33000	4	108
432	0.22165461(18)	0.22165467(21)	0.221654637(58)	33000	6	108
384	0.22165448(23)	0.22165461(26)	0.221654567(65)	31250	5	116
256	0.22165530(42)	0.22165439(47)	0.22165460(11)	46875	4	80
216	0.22165443(35)	0.22165556(28)	0.22165460(13)	40000	5	110
160	0.22165369(77)	0.22165406(75)	0.22165414(18)	50000	4	120

in one iteration, which corresponds to some 5000 or larger number of sweeps $\mathcal{N}_{sw}$ (spin flips per N). Possible systematic errors have been controlled by comparing the simulation results for different $\mathcal{N}_{sw}$ in one iteration, as well as evaluating the influence of nonlinear corrections in the Taylor series. With our choice of $\mathcal{N}_{sw}$ and the total number of iterations around 100, even a simple averaging (ignoring the nonlinear corrections) gives satisfactory results with systematic errors smaller than σ, where σ denotes the standard (statistical) error. The refined estimation then gives negligible (smaller than 0.1σ) systematic errors.

A relevant question is how many iterations should be discarded from the beginning of simulation. To clarify this point, we have analyzed our simulation results obtained by the serial Wolff algorithm for a set of sizes $160 \leq L \leq 1024$. Each simulation has been started with all spins up at $\beta = \tilde{\beta}_c^{(1)}$ ranging from 0.221 6543 to 0.221 65475 (0.221 6546 for $L \geq 640$). The MC measurements have been performed after each $L/8$ Wolff clusters. The number of measurements $\mathcal{N}_{mes}$ in one iteration is about 33 000 or larger. Pseudocritical couplings and related averages have been estimated collecting the data from several ($\mathcal{N}_{run}$) runs, only the iterations number $n \geq 5$ being included. The same has been done with two subsets of data including only the iterations number $n = 3,4$ (case 1) and $n=5,6$ (case 2). The obtained values of $\tilde{\beta}_c$ are compared in Table I, providing also the corresponding parameters $\mathcal{N}_{run}$ and $\mathcal{N}_{mes}$, as well as $\mathcal{N}_{it}$—the total number of iterations with $n \geq 5$. The indicated (in brackets) standard errors are estimated by the jackknife method (see, e.g., [3].).

The pseudocritical couplings in Table I are rather close to the true critical coupling β_c, since $\tilde{\beta}_c$ changes with L only very slightly. In fact, the actual values of about 0.221 6546 are well consistent with those reported for β_c in literature [20]. The estimates obtained from the third and fourth iterations ($n=3,4$) satisfactory well agree with those obtained from the following iterations ($n \geq 5$). However, in nine cases from ten and for all $L \geq 384$, $\tilde{\beta}_c$ of $n=3,4$ appears to be slightly smaller than that of $n \geq 5$. It points out to a possible systematic deviation within about one σ for $n=3,4$. The agreement between the estimates at $n=5,6$ and $n \geq 5$ is better, since no such systematic deviation can be mentioned. Note that the indicated standard errors for $n=3,4$ and $n=5,6$ are estimated rather approximately because of quite small total number of the used iterations.

We can judge from the above comparison that the first four iterations should likely be discarded for a very accurate and reliable estimation of the critical coupling. Accordingly, it makes sense to speed up any individual run by parallelizing the code to obtain a good and reliable result in a shorter time. We have tested also the Swendsen-Wang algorithm as a possible alternative to the Wolff algorithm. In this case, we have obtained (from two runs and totally 40 iterations with $n \geq 5$) $\tilde{\beta}_c = 0.221\ 654\ 45(16)$ at $L=256$ in agreement with the result of the Wolff algorithm. However, the estimated standard error is about 1.5 times larger despite the fact that the total number of the used sweeps (MC steps in this case) is not smaller, but even 3.37 times larger. So, we would need about 7.5 times larger computational time with the Swendsen-Wang algorithm to reach the same accuracy for $\tilde{\beta}_c$ as with the Wolff single-cluster algorithm. This advantage of the Wolff algorithm shows up if one starts the simulation from the ordered state. The latter is recommended, since the Wolff algorithm exhibits relatively poor equilibration properties when starting from the disordered state [5]. Hence, the serial Wolff algorithm in our application appears to be faster than the Swendsen-Wang algorithm, even if the latter would be speeded up 3.2 times, as in [21], by parallelizing the code. Thus, we will focus further on the parallelization of the Wolff algorithm.

III. ESTIMATION OF THE CRITICAL COUPLING

The iterative method discussed in Sec. II allows us to estimate the critical coupling β_c by fitting the data of the pseudocritical coupling $\tilde{\beta}_c(L)$. According to the general finite-size scaling arguments (see, e.g., [19,20]), we have

$$\tilde{\beta}_c(L) - \beta_c \propto L^{-1/\nu} \tag{7}$$

at $L \to \infty$, where ν is the critical exponent of the correlation length. The plot of $\tilde{\beta}_c$ vs L and the linear fit to Eq. (7) is shown in Fig. 1. This plot has a rather small asymptotic slope

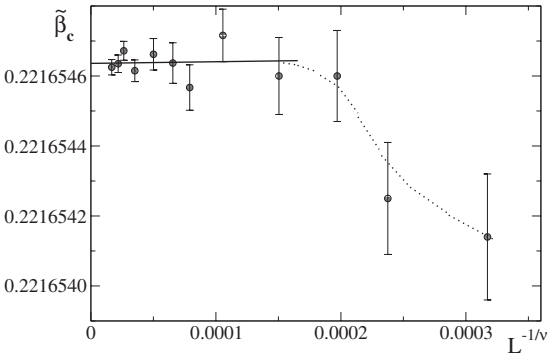

FIG. 1. The pseudocritical coupling $\tilde{\beta}_c$ vs $L^{-1/\nu}$. The solid straight line is the linear fit over the range of sizes $L \in [256,1024]$, yielding the estimate of the critical coupling $\beta_c = 0.221\ 654\ 636(20)$. The dotted line is guide to eyes.

(it almost saturates), since $U=1.6$ is very close to its universal critical value. Due to the saturation effect, the fit result for β_c is not sensitive to any small variation in ν, and we have chosen the widely accepted (for the 3D Ising model) value $\nu=0.63$. Figure 1 contains all data of Table I as well as extra data points for $L=320$ and $L=192$. Besides, the values for $L=1024$, 864, and 768 are estimated including also additional iterations produced by the parallel code, as discussed in Sec. VI. The linear fit over the range $L \in [256,1024]$ gives us $\beta_c=0.221\ 654\ 636(20)$. This value is similar but more accurate and slightly larger than the known ones $\beta_c = 0.221\ 654\ 55(5)$ [22] and $\beta_c=0.221\ 654\ 57(3)$ [23], obtained from the Binder cummulant and the susceptibility data, respectively. As one can see from Fig. 1, the asymptotic behavior of $\tilde{\beta}_c(L)$ estimated from $L \geq 256$ is not well consistent with the behavior at relatively smaller sizes $L \leq 192$. It means that a truly reliable estimation of β_c from the Binder cummulant data should be based on the simulation of larger than $L=192$ lattices. Moreover, to see the asymptotic behavior, it is necessary to have such data over a sufficiently wide range of sizes. Hence, although one usually assumes that $L=128$ is already quite large lattice size sufficient for good estimations, our data show that the data up to $L=1024$ are desirable to do the estimation of the critical coupling in a good and reliable way.

IV. GENERATION OF PSEUDORANDOM NUMBERS

The choice of a random number generator plays some role in specific implementations of our parallel Wolff algorithm. It may also be important in obtaining high quality results. In particular, we have found that some of the simulated quantities, such as specific heat (C_V) of 3D Ising model near criticality, are rather sensitive to the quality of pseudorandom numbers (PRNs). Therefore, a relevant question is how a random number generator can be improved.

The problem of removing correlations in the subtract-with-carry (SWC) generator proposed by Marsaglia and Zaman [24] has been studied by Lüscher [25]. Given the first r PRN's $x_0, x_1, \ldots, x_{r-1}$ and the "carry bit" c_{r-1}, the nth PRN ($n \geq r$) produced by the SWC generator is

$$x_n = (x_{n-s} - x_{n-r} - c_{n-1}) \bmod b, \tag{8}$$

where $c_n=0$ if $x_{n-s}-x_{n-r} \geq 0$ and $c_n=1$ otherwise. Here b, r, and s are positive integers, b is called the base and $r>s$ are the lags. The SWC generator fails to pass some correlation tests. Lüscher has proposed to discard some of the PRN's of this sequence and use only the remaining ones. In this way, the popular RANLUX generator has been developed. As proposed by James [26], one generates 24 PRNs, then discards $p-24$ ones, and so on. The parameter $p \geq 24$ defines the so-called "luxury level," i.e., $p=48,97,223,389$ correspond to luxury levels 1–4, respectively.

Later, a random-walk test performed in [27] at $r=24$ and $s=10$ has shown that the RANLUX generator shows better results in this test as compared to the SWC generator. At luxury level 2 the deviations from the expected probability distribution lie on the boundary of observations in a test with 10^{11} stochastic trajectories, using about 10^{13} random numbers, whereas level 3 is safe in runs using less than 10^{15} PRNs [27]. In this case, however, only a small fraction 24/223 of the PRN's generated by the SWC algorithm is used, which makes the RANLUX generator quite slow. Since we deal with long MC simulations, a good but faster generator would be more optimal in our application.

The problem of improving pseudorandom number generators has been recently addressed in several papers, e.g., [28], and here we present additional ideas in addressing this issue.

The linear congruential generators providing the sequence

$$I_{n+1} = (aI_n + c) \bmod m \tag{9}$$

of integer numbers I_n is a convenient choice. We have tested some generators including that of [29] with $a=843\ 314\ 861$, $c=453\ 816\ 693$, and $m=2^{31}$. The G05CAF generator of NAG library with $a=13^{13}$, $c=0$, and $m=2^{59}$ (generating odd integers) has been extensively used in [20]. We have compared the results of both generators for the 3D Ising model, simulated by the Wolff cluster algorithm, and have found a disagreement by almost 1.8% in the maximum value of C_V for the system size $L=48$. Application of the standard shuffling scheme (see, e.g., [3], p. 391) with the length of the shuffling box (string) $N_{\mathrm{sh}}=128$ appears to be not helpful in removing the discrepancy. The problem is that the standard shuffling scheme, where the numbers created by the original generator are put in the shuffling box and picked up from it with random delays in about N_{sh} steps, effectively removes the short-range correlations between the pseudorandom numbers, but nevertheless it does not essentially change the block averages $\langle I_n \rangle_k = k^{-1} \Sigma_{j=n}^{n+k-1} I_j$ over k subsequent steps if $k \gg N_{\mathrm{sh}}$. It means that such a shuffling is unable to modify the low-frequency tail of the Fourier spectrum of the sequence I_n to make it more consistent with white noise (an ideal case). The numbers I_n are repeated cyclically and the block averages over the cycle do not fluctuate at all in contradiction with truly random behavior.

To resolve this difficulty, we have applied a second shuffling as follows. We have split the whole cycle of length m of the actual generator with $m=2^{31}$ in 2^{20} segments each consisting of 2048 numbers. Starting with 0, we have recorded the first numbers of each segment. It allows us to restart the

generator from the beginning of any segment. The last pseudorandom number generated by our shuffling scheme is used to choose the next number from the shuffling box, exactly as in the standard scheme. In addition, we have used the last but one number to choose at random a new segment after each 2048 steps. This double-shuffling scheme mimics the true fluctuations of the block averages even at $k \gg m$. We have used a very large shuffling box with $N_{\mathrm{sh}}=2^{20}$ to make the shuffling more efficient by mixing the pseudorandom numbers from many segments. Such a shuffling scheme has an extremely long cycle: the logarithm of its length is comparable with $\ln(N_{\mathrm{sh}}!)$.

A hidden problem is the existence of certain long-range correlations in the sequence I_n of the original generator of [29]. Namely, pseudorandom numbers of a subset, composed by picking up each 2^kth element of the original sequence, appear to be rather strongly correlated for $k \geq 20$. It is observed explicitly by plotting such a subsequence I_n^* vs n, particularly, if the first element is chosen $I_1^*=0$. These correlations reduce the effectiveness of our second shuffling. Correlations of this kind, although well masked, exist also in the sequence of G05CAF generator. Namely, if we choose $I_1^*=1$ and $k=25$ and generate the coordinates (x,y) by means of this subset (known as the spectral test), then we observe that the x-y plane is filled by the generated points in a nonrandom way. The origin of these correlations, obviously, is the choice of modulo parameter m as a power of 2. Apparently, it is the reason for systematic errors in some applications of Swendsen-Wang algorithm discussed in [30].

A promising alternative, therefore, is to use the well-known Lewis generator (see, e.g., [3]), where $m=2^{31}-1$ is a prime number, $a=7^5$, and $c=0$, as the original generator of our double-shuffling scheme. This generator has been tested, e.g., in [31], providing good results in relatively short simulations without any shuffling. As before, the cycle is split in 2^{20} segments. However, the first segment now starts with 1. Besides, the first and the last segments contain only 2047 elements instead of 2048. After all numbers of the previous segment are exhausted, a new segment is chosen as follows: if the last but one random number of our shuffling scheme is I, then we choose the kth segment, where $k=1+[I/2048]$. Since we never have $I=0$ or $I=m$, it ensures that each segment is chosen with the probability proportional to its length. We have used the shuffling box of length $N_{\mathrm{sh}}=10^6$ for this scheme.

Good results in the directed one-dimensional random-walk test (see, e.g., [27]) may be important to ensure that the pseudorandom number generator works well with the cluster algorithms. At discrete times, the random walker either makes step forward with the probability μ, or stops with the probability $1-\mu$, starting a new random walk afterwards. The probability of stopping at nth time step is $P(n)=\mu^{n-1}(1-\mu)$. As in [27], we have calculated the relative deviation $\delta P(n)=(P^*(n)/P(n))-1$ of the observed in MC simulation probability $P^*(n)$ from the exact value $P(n)$ for $n \leq 100$ at $\mu=31/32$. We have tested our latter shuffling scheme, using up to 10^{12} stochastic realizations of the random walk. For comparison, no more than 10^{11} stochastic trajectories have been used in [27] to test the RANLUX generator. Our results for a set of 10^{12} trajectories are shown in Fig. 2. Generally, only expected statistical fluctuations of $\delta P(n)$ around zero, which look different for different runs, and no detectable systematic deviations have been observed. Taking into account the very small values of $\delta P(n)$ reached in our test simulation, it confirms the high quality of the generated pseudorandom numbers.

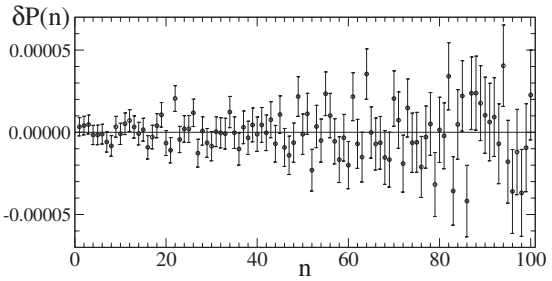

FIG. 2. The relative error $\delta P(n)$ in the MC measured probability of a random walk of length n at $\mu=31/32$, estimated from a set of 10^{12} stochastic trajectories.

As another test, we have simulated by the Wolff algorithm the mean energy $\langle \varepsilon \rangle$, specific heat C_V, as well as its derivatives $C_V'=\partial C_V/\partial \beta$ and $C_V''=\partial^2 C_V/\partial \beta^2$ for the 2D Ising model at the critical point and have compared the results with those extracted from exact formulas (see, e.g., Eqs. (7) and (8) in [32]). The test simulations consisting of 4.8×10^8 and 2.4×10^7 cluster-algorithm steps have been made for the lattice sizes $L=48$ and $L=256$, respectively. The values provided by the G05CAF generator and our two shuffling schemes agreed with the exact ones within the errors about one σ. The most serious deviation of 2.37σ has been observed for C_V'' in the case of $L=48$ simulated by our first shuffling scheme. At $L=48$, one standard deviation σ corresponded to $\sim 0.0009\%$ relative error for $\langle \varepsilon \rangle$, $\sim 0.02\%$ error for C_V, $\sim 0.2\%$ error for C_V', and $\sim 0.35\%$ error for C_V''. At $L=256$ these errors were $\sim 0.0012\%$, $\sim 0.12\%$, $\sim 3\%$, and $\sim 4\%$, respectively. We have used our second shuffling scheme with the Lewis generator in the simulations and parallel implementations discussed here.

V. PARALLEL VERSION OF THE WOLFF ALGORITHM

According to [1], one step of the Wolff single-cluster algorithm for the Ising model consists of the following substeps:

(1) Choose a seed spin of the new cluster at random and flip it;

(2) Look in turn at each of the neighbors of that spin and find the ones, which are pointing in the opposite direction as the flipped seed spin. Each of them is added to the cluster and simultaneously flipped with probability $P_{\mathrm{add}}=1-e^{-2\beta}$.

(3) For each spin that was added in the last step, examine each of its neighbors to find the ones which are pointing in the opposite direction, adding each of them to the cluster and simultaneously flipping with probability $P_{\mathrm{add}}=1-e^{-2\beta}$. This step is repeated until there are no new spins added to the cluster.

This algorithm is formulated in [3] in a slightly different way, where the flipping of cluster spins is postponed to an extra step.

A new generation of spins is added to the growing Wolff cluster at each iteration of step 3. Our basic idea is to perform any of such iterations by using parallel threads, provided that the wave front of the growing cluster, consisting of the spins added in the last step, occupies more than N_{min} lattice sites. We will call these spins the wave-front spins. If the wave front contains $\leq N_{min}$ spins, then it is treated serially by the master processor. Here N_{min} is an optimization parameter. Apart from the parallelization of step 3, the pseudorandom numbers in our algorithm are also generated in parallel and stored in an array for further use when necessary. Finally, the Monte Carlo measurements of energy and magnetization are also performed by parallel threads, which is quite simple and therefore will not be discussed in detail.

The parallel treatment of one iteration of step 3 is performed according to the following scheme:

(1) divide the list of the wave-front coordinates (coordinates of those spins added in the last step) between the processors. Three arrays with these x, y, and z coordinates, as well as the array of spin variables are stored in the shared memory.

(2) Perform step 3 of the Wolff algorithm in parallel only for the subset of those neighboring spins, which are located in one of the six possible directions from each of the considered wave-front spins. The work of processors is synchronized by putting the OMP BARRIER after this substep. Then the same is performed for the remaining five directions, putting the OMP BARRIER after each of them. Each processor treats certain fraction of the cluster wave front, assigned in step 1 of this scheme. Besides, each processor forms its own lists of x, y, and z coordinates of newly added spins and counts the number of elements in these lists. These lists and numbers of elements are shared variables; however, each processor stores them in a separate subdomain of the shared memory.

(3) Form the common lists of the x, y, and z coordinates of newly added spins. It is done in parallel, in such a way that each processor writes its own list in a certain place of the common shared arrays, determined according to the values of shared variables (numbers of elements) defined in the previous step. The total number of elements in the common lists is determined by the master processor.

In the above scheme, it is necessary that each processor works with certain fraction of the shared arrays assigned to it according to the thread number. It is reached by using the following structure in the parallel region of the FORTRAN code:

```
DO ID=1,IPROC
IF(OMP_GET_THREAD_NUM().EQ.ID-1)
THEN
……
END IF
END DO
```

Here IPROC is the number of processors used. All the operators of steps 2 and 3 in the above scheme are put inside the logical IF between THEN and END IF. In this case certain value of ID is assigned to each thread (these are numbered from 0 to IPROC-1), which allows to organize explicitly its work depending on the thread number.

The splitting of the procedures of adding and flipping of spins by treating separately each of the six directions with OMP barriers in between, as explained in point 2 of the above scheme, is a very essential point. It ensures that the parallel algorithm works correctly. Namely, it excludes the situations where different processors try to flip simultaneously the same spin, i. e., write simultaneously in the same unit of the shared memory. Without the separation, this would be possible if such a spin is a neighbor of two or several wave-front spins. Even if the result of simultaneous flipping could appear to be correct (once the spin is added to the cluster, it can be formally added at the same time repeatedly, although it could fail technically), a problem remains such that the newly added spin can appear two or several times in their common list formed in step 3 of our scheme. Excluding the simultaneous flipping, our separation automatically resolves this problem, as well.

We propose to generate the pseudorandom numbers in advance. We have found empirically how it can help to make our parallel code more efficient. In fact, the adding of a spin with the probability P_{add} requires only to compare a uniformly distributed random number $r \in (0,1)$ with P_{add}. Therefore we need not to store the generated random numbers, but only a random sequence of two numbers, 0 and 1, where 1 corresponds to $r < P_{\mathrm{add}}$ (acceptance) and 0 to $r > P_{\mathrm{add}}$ (rejection). We generate such a sequence in advance and store it in an array RANDO of certain dimension M to use it in parallel, as well as in serial treatments of the cluster wave front. Each processor takes the elements from certain fraction of this array. The number of such fractions is equal to the number of processors used the parallel treatment. In the case of the serial treatment, the elements are taken from that fraction, which contains more elements not used yet. The supply and use during one complete treatment of a cluster-wave front is organized as follows:

(1) Check the number of still not used or available elements in all fractions of the array RANDO. If, for any of the used fractions, two conditions are satisfied: (1) the number of available elements n_{av} is smaller than n^*, where n^* is certain number of available elements at which the treatment of the currently considered wave front can be surely completed, and (2) the number of available elements is smaller than a half of all elements, then generate pseudorandom numbers (in parallel) to replace all the already used elements of the array RANDO with new elements.

(2) If, after step 1, $n_{av} \geq n^*$ holds for all the used fractions of RANDO, then treat the current wave front (or complete to do this if step 2 is repeated), counting how many unused elements are left in the fractions of RANDO. Otherwise, determine a part of the wave front which can be surely treated (and is not treated in previous iterations of step 2), then treat this part, counting how many unused elements are left in the fractions of array RANDO, and return to step 1.

Condition (2) in step 1 is added to ensure that the array RANDO is completed with new elements only in sufficiently large portions, which increases the efficiency of the code. We have used a rather large array of dimension $M=40\,000\,000$. It, again, increases the efficiency of the parallel generation of

the pseudorandom numbers. Besides, at such a large M, we practically never return in the above scheme from step 2 back to step 1 in the actually considered simulations near the critical point.

Consider now the parallel generation of pseudorandom numbers, using the shuffling scheme described in Sec. IV. In this case the parallel threads should generate independent sequences of pseudorandom numbers. It is achieved by using different initial sets of variables characterizing the current state of the shuffling scheme, further called the generator state variables (such as the arrays representing the shuffling boxes, and last generated number), for each of the threads.

It is convenient to store all the different sets of state variables for different threads in the shared memory, since then they are not lost outside the parallel regions of the code and can be written in a file to continue the simulation from a record, if necessary. In principle, one can always generate a random number just when it is necessary, relating explicitly certain set of variables to certain thread number within the already mentioned structure of the logical IF. However, to get an efficient parallel code in this way is a difficult task. We have tried to generate separately the random sequences of 0 and 1, stored in the array RANDO and used later according to the described scheme. In this case nothing becomes better if we try to do the parallel generation working just with the shared variables. However, it allows to test how efficient the parallel generation of the pseudorandom numbers alone is. Surprisingly, we have found that it is even slower than the serial generation, although this conclusion may depend on the specific generator and specific computer used. We have performed our test simulations on the Opteron cluster containing 4 cores (2 sockets $\times$ 2 cores per socket) per node running at 2.2 GHz (Shared Hierarchical Academic Research Computing Network: www.sharcnet.ca cluster *narwhal*). An essential property of the used pseudorandom number generator is the presence of a large shuffling box (see Sec. IV). It means that the search of elements with randomly chosen addresses in a large array takes place permanently. We have tested by a serial code that such a use of the shuffling box slows down the generator remarkably (although it does not slow down too much the whole simulation). This has led us to a conclusion that searching random addresses in the shared memory by several processors simultaneously is quite inefficient. Indeed, we have found that the efficiency of the parallel generation becomes close to 100% if private instead of shared variables are used for our shuffling scheme locally inside a parallel region. Namely, we update the array RANDO as follows:

(1) Define the private state variables of the shuffling scheme and assign them the values of the corresponding shared variables depending on the thread number;

(2) Generate in parallel the pseudorandom numbers and update the array RANDO (shared variable), working with the private state variables; and

(3) Assign the final values of the private state variables used by each thread to the corresponding shared variables.

A high efficiency can be reached only if the assignment steps 1 and 3 take a relatively small time as compared to the generation step 2. Since we use a large shuffling box of the size $N_{\mathrm{sh}}=10^6$, we take even much larger dimension $M=4\times10^7$ of the array RANDO to ensure this. After the implementation of this scheme, the generation of the pseudorandom numbers is parallelized very efficiently.

However, the problem with searching of array elements appears also in the treatment of the wave front of the growing Wolff cluster. The elements in this case are the values of the wave-front spins with coordinates taken from certain lists, as explained before. These coordinates are not ordered, but also are not completely random. We have observed some speeding up (relative to the serial treatment) in the parallel treatment of the wave front on two and four processors, if the wave front contained more than 500 spins. Therefore we have set the optimization parameter $N_{min}=500$ in our algorithm. However, we have not observed that the speedup becomes almost proportional to the number of processors even for much larger wave fronts containing more than 10 000 spins. The reason for this could be, again, a not quite efficient parallel searching in shared arrays.

There are two possible ways to implement the adding of a spin to the cluster (simultaneously flipping it) with certain probability P_{add}. One way is to look first whether the actual spin is opposite to the cluster spins and, if it is true, then look for a random variable to add or reject the spin. Another way is to look first for the random variable and, if it is such that the spin can be added, then add the spin if its direction (sign) is appropriate. In the first case, it is always necessary to search in an array to determine the spin value, and in fewer cases one needs to have a random variable. In the other case it is vice versa. So, if the searching is relatively slow as compared to the generation of a random number, then the second implementation is more efficient. However, in our simulations, the first implementation works better both in serial and in parallel code, and the tests and discussions that follow refer only to this case.

VI. TEST SIMULATIONS

We have performed a series of test simulations to estimate the efficiency of the developed parallel procedure, as well as to compare the results with those of the serial code. We have tested also the influence of the lattice sizes on the estimation of critical exponent η.

Note that initialization of parallel region and OMP barriers requires some time, which can even exceed that of a serial calculation of the corresponding simulation step. Therefore, only a large enough generation of newly added spins, forming the wave front of a growing Wolff cluster, is worth to be treated in parallel, and the parallel code is more efficient for larger lattice sizes. As we have mentioned already, our simulations showed some speedup of this step only if the wave front contained more than some $N_{min}=500$ spins. Hence, smaller wave fronts have been treated serially. For the lattice size $L=1024$, used in our test simulations, the wave front at $\beta\simeq\beta_c\simeq0.221\,654\,6$ typically contains several thousands of spins so that it is most often treated in parallel. For much smaller lattices, e.g., $L\leq128$ considered in [20], the wave front at $\beta\simeq\beta_c$ usually contains less than 500 spins. However, since the generation of the pseudorandom numbers is always parallel, some speedup is observed even for such relatively small lattices.

TABLE II. A set of quantities for three different system sizes L, evaluated at $\beta=0.221\,654\,6$ from the simulations with serial code.

L	$\langle\varepsilon\rangle$	C_V	χ/L^2	U	$10^{-4}\partial U/\partial\beta$
1024	−0.9907293(22)	107.52(76)	1.1985(40)	1.6036(31)	−12.96(14)
864	−0.9907639(23)	104.41(53)	1.2099(35)	1.6040(27)	−9.968(81)
768	−0.9907903(25)	100.92(51)	1.2111(33)	1.6056(24)	−8.191(63)

We have compared the simulation times of the parallel code with those of the serial code. In our test simulations on the Opteron cluster described before, the parallel code speeded up the simulation near the critical point (i.e., at $\beta=0.221\,654\,6$) about 1.79 times on two processors and about 2.67 times on four processors for the lattice of linear size $L=1024$. It corresponds to 89% and 67% efficiency, respectively. These estimates represent average values over long runs in an equilibrium state. In fact, we have used here an initial spin configuration obtained after one of the long serial runs considered in Sec. II. In the following test simulations, 15000 measurements and $15\,000\times128$ clusters of average size 1 317 115.5 have been generated by the serial code in 390.36 h. The same number of measurements and clusters has been generated in 217.81 h by the parallel code on two processors. The average cluster size was slightly different in this case (due to fluctuations), i.e., 1 312 678.2. It has been taken into account that the simulation time is proportional to the mean cluster size for a precise comparison of the simulation speeds—spin flips per time. Similarly, the simulation including 33 000 measurements and $33\,000\times128$ clusters of average size 1 304 379 has been performed in 319.14 h on four processors, leading to the above mentioned estimate of the speedup and efficiency.

Thus, using four processors, the code generated on average about 221 Wolff clusters per minute in the equilibrium state at $\beta=0.221\,654\,6$ and $L=1024$. There is, however, an initial delay of about 25 s due to the initialization of the shuffling scheme. It has been subtracted from the total simulation times to obtain the above mentioned estimates.

We have also performed longer test simulations, applying the iterative scheme introduced in Sec. II, with the aim to compare the results with those of the serial code for $L=768,864,1024$. Only one run ($\mathcal{N}_{run}=1$) with the total number of the used iterations $\mathcal{N}_{it}=10$ for $L=768$ and $\mathcal{N}_{it}=8$ for $L=864,1024$ have been performed. Two processors have been used for $L=768,864$ and 4 processors for $L=1024$. The first four iterations have been discarded for two smallest sizes $L=768$ and $L=864$, as in the case of the serial simulations (Sec. II). We have used the already equilibrated initial spin configuration for $L=1024$, discarding only the first two iterations. Other simulation parameters, not mentioned here, are the same as in the serial simulations.

The pseudocritical couplings, provided by our parallel simulations, i.e., $\tilde{\beta}_c=0.221\,654\,693(81)$ for $L=768$, $\tilde{\beta}_c=0.221\,654\,582(62)$ for $L=864$, and $\tilde{\beta}_c=0.221\,654\,608(60)$ for $L=1024$ agree well within one standard error with the corresponding values of the serial simulations given in fourth column of Table I. Apart from the pseudocritical coupling, we have compared also some other quantities evaluated at $\beta=0.221\,654\,6$. The simulation results obtained by the serial and the parallel codes are given in Tables II and III, respectively.

Here $C_V=N(\langle\varepsilon^2\rangle-\langle\varepsilon\rangle^2)$ is the specific heat, $\chi=N\langle m^2\rangle$ is the susceptibility, and the other quantities are defined in Sec. II. As we can see, the values in most of the cases agree within the indicated error bars of one σ. In relatively fewer cases larger deviations are expected from the statistics, and we observe them for C_V and $\partial U/\partial\beta$ at $L=864$. Note, however, that the standard errors (σ) for the parallel simulation have been estimated rather approximately from a few iteration, i.e., only eight iterations in this case. More realistic estimates of σ correspond to 3σ of the serial simulation, since the latter contains 72 such iterations and $\sigma\propto1/\sqrt{\mathcal{N}_{it}}$ holds. Thus, we can see that the discrepancies always are small enough and the overall agreement is good.

We have also tested the influence of the lattice sizes on the estimation of the critical exponent η, describing the critical behavior of the two-point correlation function [19], as well as the finite-size scaling of the susceptibility at $\beta=\beta_c$,

$$\chi\propto L^{2-\eta}:L\to\infty. \tag{10}$$

According to the finite-size scaling theory, the same asymptotic scaling relation is true for any fixed ratio of L/ξ (at $L\to\infty$), where ξ is the correlation length. In particular, it is true for $\chi(L)$ determined at the psedocritical coupling $\beta=\tilde{\beta}_c(L)$. It reveals a possibility to estimate η without determination of β_c. It might be a great advantage, since the result

TABLE III. A set of quantities for three different system sizes L, evaluated at $\beta=0.221\,654\,6$ from the simulations with parallel code.

L	$\langle\varepsilon\rangle$	C_V	χ/L^2	U	$10^{-4}\partial U/\partial\beta$
1024	−0.9907291(46)	107.7(2.2)	1.201(10)	1.6010(78)	−13.04(33)
864	−0.9907654(56)	102.12(66)	1.2137(92)	1.5983(59)	−9.58(15)
768	−0.9907884(68)	101.7(1.3)	1.205(10)	1.6076(67)	−8.19(15)

TABLE IV. The susceptibility χ, normalized to L^2 and determined at $\beta=\tilde{\beta}_c(L)$, and the critical exponent η estimated from (11) depending on lattice sizes.

L	χ/L^2	η
1024	1.2046(28)	
512	1.2367(16)	0.0356(19)
256	1.2656(15)	0.0312(11)
128	1.2913(10)	0.02743(96)
64	1.31466(78)	0.02041(64)
32	1.32835(59)	

of estimation at $\beta=\beta_c$ is rather sensitive to the precise value of β_c. According to Eq. (10), we can evaluate η, e.g., as

$$\eta = -\frac{\ln[f(2L)/f(L/2)]}{\ln 4}, \tag{11}$$

where $f(L)=\chi(L)/L^2$ at $\beta=\tilde{\beta}_c(L)$. Such an estimation for various L shows how the result depends on L. The convergence to the true value of η is expected at $L\to\infty$. Our results for a set of sizes are collected in Table IV. These values of η do not differ much from those usually reported in literature [19]. Nevertheless, the value $\eta=0.020\,41(64)$ estimated from relatively small sizes $L=128$ and $L=32$ is remarkably smaller than that extracted from the largest sizes, i.e., $\eta=0.0356(19)$. Although the latter value is quite close to the estimate $\eta=0.0366(8)$ obtained in [20] by an extrapolation from much smaller lattice sizes, the asymptotic value could be even larger according to the observed tendency of increasing with L (see Table IV). Hence, it would be very useful to have simulation data for even larger than $L=1024$ lattice sizes to make a reliable conclusion about the asymptotic value of η.

VII. DISCUSSION

The developed parallel implementation of the Wolff single-cluster algorithm for the 3D Ising model may prove to be indispensable in the actual simulations of large lattices, as it allows to speed up the simulation some 1.79 times on two processors and 2.67 times on four processors for $L=1024$. These values, however, can depend on the specific computer used. Another advantage of the proposed parallelization is that the parallel code can use a larger operative memory, i.e., that of several processors, which allows to simulate larger lattices. For instance, $L=1024$ is almost the maximal linear lattice size for the serial code run on one processor of the actually used Opteron cluster, whereas the use of two processors allows us to extend the simulations to $L=1280$.

Also for the lattice size $L=1024$, actually simulated mainly by the serial code, the parallel code would allow to obtain the current results faster and even with slightly smaller overhead. Actually, the trivial parallelization has been used in the serial simulations. Namely, we have performed eight runs, each including ten iterations, from which the first four have been discarded. The total simulation took about one year of real time. The result of the same accuracy could be obtained by using hybrid parallelization with four runs, each on two processors, discarding four, and keeping 12 iterations from each of the run. Thus, 16 iterations would be performed by each run in about 0.9 years, in accordance with the estimated speedup. Hence, the overhead also would be decreased by a factor of 0.9. Since we have decided to discard the first four iterations, it would be impossible to obtain any result by the serial code in less than 0.4 years. The already mentioned one-year calculation results of the serial code could be obtained in about 0.38 years with only 1.5 times larger overhead by using the same number of parallel (four processor) instead of serial runs. If only a very short initial equilibration is used, then the application of the parallel code gives smaller effect. However, even in this case the parallel code helps to reduce the real simulation time. It is necessary to perform at least few iterations to verify the convergence of the pseudocritical coupling to certain value. In our example, it would take about 3 months for only two iterations. The parallel code would allow to reduce this time to about 1.1 months.

As a continuation of this work, similar codes can be elaborated for $O(n)$ models [1,19]. These are the lattice spin models, in which the local order parameter is an n component vector, which can be rotated continuously. The same principles of the parallelization can be used here. In this case, the spin flips for any one of the Wolff clusters are reflections with respect to a randomly chosen plane. As a result, the acceptance probability for a spin flip needs to be recalculated permanently. Due to these extra operations, one can expect that the initialization times of the parallelization, as well as the times spent for inefficiently parallelizable searching of random addresses in shared arrays will be relatively smaller, i.e., the speedup and efficiency higher than in the Ising case.

We have estimated this effect by using certain testing algorithm. In principle, the adding of a spin to the cluster with probability $P_{\text{add}}=1-e^{-2\beta}$ can be realized by means of two random numbers as follows. First, a uniformly distributed random variable $\varphi\in[0,\pi/2]$ is generated, and then the spin is added with the probability $\tilde{P}_{\text{add}}(\varphi)=\pi\beta\sin(\varphi)e^{-2\beta\cos\varphi}$. Since we have $\langle\tilde{P}_{\text{add}}\rangle_\varphi=P_{\text{add}}$, it represents another method of simulation of the Ising model, further called the testing algorithm. It mimics essential simulation features of the n component vector models, where the spin adding probability has the form $P_{\text{add}}=1-e^{\beta(\cos\varphi_1-\cos\varphi_2)}$ with φ_1 and φ_2 being the angles between a reference spin and a neighboring spin before and after its flipping, respectively. Like this formula, $\tilde{P}_{\text{add}}(\varphi)$ contains two trigonometric functions and the exponent. However, an extra pseudorandom number now is needed for the random variable φ. We have used the Lewis generator to produce it just when necessary. This procedure is so fast that it does not essentially influence our time-testing results. Although the Lewis generator is not appropriate for accurate large scale MC simulations, it can be used for the speedup estimation in our testing algorithm. Applying our parallelization method, we have reached the speedup about 3.03 times on four processors. It corresponds to 76% efficiency, which is higher than 67% reached in the standard simulation of the Ising model. Thus, the effect of recalcula-

tion P_{add} at each step is evident: the speedup of parallelization on four processors increases from ≈ 2.67 to ≈ 3.03. It is relevant for the n component vector models [called also $O(n)$ models] with $n \geq 2$, where such recalculation is required, and therefore the speedup about 3 can be indeed reached. This estimation most precisely corresponds to the $XY(n=2)$ model, where the spin state is given by an angle φ. In other cases, the effect from recalculation of P_{add} will be similar or even larger, as in the case of large n, where the calculation of the scalar products represented by $\cos \varphi_1$ and $\cos \varphi_2$ will require many arithmetic operations.

VIII. CONCLUSIONS

A parallel (OpenMP) implementation of the Wolff single-cluster algorithm for the 3D Ising model has been proposed (Sec. V). The parallel algorithm may prove to be indispensable in the simulations of large lattices with $L \sim 1024$ near the critical point. We have tested it within the discussed here applications, using the iterative method described in Sec. II and certain shuffling scheme (Sec. IV) as the generator of pseudorandom numbers. According to the tests described in Sec. IV, this shuffling scheme produces pseudorandom numbers of a high quality. Test simulations for the lattice with linear size $L=1024$ have been performed, showing a speedup about 1.79 times on two processors and 2.67 times on four processors relative to the serial code. Based on the results of certain testing algorithm, we argue (Sec. VII) that somewhat larger speedups, about three times on four processors, can be reached by our method applied to n component ($n \geq 2$) vector models. Besides, the parallel code allows to simulate larger lattices. The actual simulation results for $L = 768, 864, 1024$, provided by the parallel code, agree with those of the serial code.

As a result of our simulations, the critical coupling β_c, i.e., $\beta_c = 0.221\,654\,636(20)$ (Sec. III) has been reliably estimated with an unprecedented accuracy. The influence of lattice sizes on the estimated value of the critical exponent η has also been tested (Sec. VI), showing that the simulation of even larger than $L=1024$ lattices is desirable to obtain a reliable result here.

ACKNOWLEDGMENTS

This work was made possible by the facilities of the Shared Hierarchical Academic Research Computing Network (SHARCNET:www.sharcnet.ca). R.V.N.M. acknowledges the support from the NSERC and CRC program.

[1] U. Wolff, Phys. Rev. Lett. **62**, 361 (1989).
[2] J.-S. Wang and R. H. Swendsen, Physica A **167**, 565 (1990).
[3] M. E. J. Newman and G. T. Barkema, *Monte Carlo Methods in Statistical Physics* (Clarendon, Oxford, 1999).
[4] S. Gündüç, M. Dilaver, M. Aydın, and Y. Gündüç, Comput. Phys. Commun. **166**, 1 (2005).
[5] J. Du, B. Zheng, and J.-S. Wang, J. Stat. Mech. (2006), P05004.
[6] D. Ivaneyko, J. Ilnytskyi, B. Berche, and Yu. Holovatch, Physica A **370**, 163 (2006).
[7] U. K. Rößler, Phys. Rev. B **59**, 13577 (1999).
[8] B. Berche, C. Chatelain, C. Dhall, R. Kenna, R. Low, and J.-C. Walter, J. Stat. Mech. (2008), P11010.
[9] Y. Deng, Phys. Rev. E **73**, 056116 (2006).
[10] J. L. Jacobsen, P. Le Doussal, M. Picco, R. Santachiara, and K. J. Wiese, Phys. Rev. Lett. **102**, 070601 (2009).
[11] J. Kaupužs, R. V. N. Melnik, and J. Rimshans, Eur. Phys. J. B **55**, 363 (2007).
[12] J. Kaupužs, R. V. N. Melnik, and J. Rimšāns, Comm. Comp. Phys. **4**, 124 (2008).
[13] F. W. S. Lima and D. Stauffer, Physica A **359**, 423 (2006).
[14] B. B. Beard, M. Pepe, S. Riederer, and U.-J. Wiese, Comput. Phys. Commun. **175**, 629 (2006).
[15] S. Bae, S. H. Ko, and P. D. Coddington, Int. J. Mod. Phys. C **6**, 197 (1995).
[16] J. Machta, Y. S. Choi, A. Lucke, T. Schweizer, and L. V. Chayes, Phys. Rev. Lett. **75**, 2792 (1995).
[17] J. Machta, Y. S. Choi, A. Lucke, T. Schweizer, and L. M. Chayes, Phys. Rev. E **54**, 1332 (1996).
[18] E. Faraggi and D. T. Robb, Phys. Rev. B **78**, 134416 (2008).
[19] A. Pelissetto and E. Vicari, Phys. Rep. **368**, 549 (2002).
[20] M. Hasenbusch, Int. J. Mod. Phys. C **12**, 911 (2001).
[21] G. T. Barkema and T. MacFarland, Phys. Rev. E **50**, 1623 (1994).
[22] Y. Deng and H. W. J. Blöte, Phys. Rev. E **68**, 036125 (2003).
[23] J. A. Plascak, A. M. Ferrenberg, and D. P. Landau, Phys. Rev. E **65**, 066702 (2002).
[24] G. Marsaglia and A. Zaman, Ann. Appl. Probab. **1**, 462 (1991).
[25] M. Lüscher, Comput. Phys. Commun. **79**, 100 (1994).
[26] F. James, Comput. Phys. Commun. **79**, 111 (1994).
[27] L. N. Shchur and P. Butera, Int. J. Mod. Phys. C **9**, 607 (1998).
[28] L.-Y. Deng, R. Guo, D. K. J. Lin, and F. Bai, Comput. Phys. Commun. **178**, 401 (2008).
[29] G. E. Forsytne, M. A. Malchom, and C. B. Moler, *Computer Methods for Mathematical Computations* (Prentice-Hall, Englewood Cliffs, NJ, 1977).
[30] G. Ossola and A. D. Sokal, Phys. Rev. E **70**, 027701 (2004).
[31] A. M. Ferrenberg, D. P. Landau, and Y. J. Wong, Phys. Rev. Lett. **69**, 3382 (1992).
[32] J. Kaupužs, Int. J. Mod. Phys. C **16**, 1121 (2005).