Universidade Federal do Rio Grande do Sul
Instituto de Física
Trabalho de Conclusão de Curso

Estudo de Modelos de Rede por Meio de Algoritmos de Execução em Paralelo

Luis Carlos Fagundes Latoski

Trabalho de Conclusão de curso apresentado como requisito parcial para obtenção do título de Bacharel em Física com ênfase em Astrofísica.

Orientador:
Prof. Dr. Heitor Carpes Marques Fernandes.

Porto Alegre, RS
Janeiro de 2020

Latoski, Luis Carlos F.
Estudo de Modelos de Rede por Meio de Algoritmos de Execução em Paralelo / Luis Carlos Fagundes Latoski. -- 2020.
62f.

Orientador: Dr. Heitor Carpes Marques Fernandes.

Trabalho (Conclusão de curso) - Universidade Federal do Rio Grande do Sul, Instituto de Física, Porto Alegre, BR-RS, 2020.

Monte Carlo, Modelo de Ising, Paralelização, Gás de Rede, Difusão. I. Fernandes, Heitor C.M., orient.

Agradecimentos

A minha companheira Luana e amigos, que pacientemente respeitaram minha ausência no último semestre, me incentivando sempre a dar meu melhor.

A minha mãe, Miriam, que sempre esteve ao meu lado nos momentos mais difíceis.

Ao meu orientador Heitor, que dedicou grande parte do seu tempo nos últimos meses ao desenvolvimento desta monografia.

E a todos que participaram direta ou indiretamente desta longa jornada, que não só ajudaram a construir esta monografia, como também a pessoa que sou hoje.

Obrigado por sempre acreditarem em mim e na minha capacidade. Principalmente quando até eu deixei de acreditar.

Esta trabalho dedico à vocês.

Resumo

Neste trabalho estudamos por meio de simulações Monte Carlo, o impacto da paralelização de algoritmos no Modelo de Ising e no Gás de Rede, ambos em duas dimensões. Inicialmente, utilizamos o modelo de Ising para o estudo dos conceitos da programação em paralelo e na implementação de modificações simples na Cadeia de Markov subjacente. Os resultados conhecidos para o modelo foram reproduzidos e a abordagem em paralelo apresentou reduções expressivas nos tempos de execução, chegando a $\approx 90\%$ em alguns casos. Nas estratégias de paralelização adotadas, foi observado que estas não influenciaram os tempos de correlação entre as medidas, e, com isto, não alteraram a eficiência estatística das amostragens realizadas.

O estudo da paralelização dos algoritmos para a difusão de partículas no Gás de Rede foi desenvolvido em duas etapas. Primeiro, foram desconsideradas interações com troca de energia ($\epsilon = 0$), obtidos os coeficientes e os expoentes do desvio quadrático médio, com a finalidade de caracterizar se as distintas estratégias de paralelização apresentavam comportamento difusivo. Também foram elaboradas estratégias de paralelização que permitissem a maior independência possível entre os graus de liberdade na movimentação das partículas do sistema. Os resultados conhecidos foram reproduzidos, em paralelo, com uma redução explícita no tempo de execução de até $\approx 70\%$ em alguns casos. Em todas as abordagens, foi observado comportamento difusivo, porém com diferentes valores para os coeficiente de difusão.

Palavras-chave: Monte Carlo, Modelo de Ising, Paralelização, Gás de Rede, Difusão.

Abstract

In this work we study through Monte Carlo simulations, the impact of the parallel running algorithms on the Ising Model and on the Lattice Gas, both in two dimensions. Initially, the Ising model was used to study the concepts of parallel programming and to implement simple modifications to the underlying Markov chain. The known results for the model were reproduced and the parallel approach showed significant reductions in execution times, reaching $\approx 90\%$ in some cases. In the adopted parallelization strategies, it was observed that they did not influence the correlation times between the measurements, and, thus, did not change the statistical efficiency of the sampled configurations.

The study of the parallel version of the algorithms for particle diffusion in Lattice Gas was developed in two steps. First, interactions were neglected, energy exchange ($\epsilon = 0$), and the coefficients and exponents of the mean square deviation were obtained, in order to characterize whether the different parallelization strategies presented diffusive behavior. Parallelization strategies were also developed to allow the greatest possible independence between the degrees of freedom during the movement of the system particles. The known results have been reproduced in parallel with an explicit reduction in execution time of up to $\approx 70\%$ in some cases. In all approaches diffusive behavior was observed, but with different values for diffusion coefficients.

Keywords: Monte Carlo, Ising Model, Parallel Programming, Lattice Gas, Diffusion.

Sumário

Capítulo 1

Introdução

Nas últimas décadas o Método de Monte Carlo (MC) se estabeleceu com uma das principais ferramentas no estudo de sistemas que passam por transições de fase [1]. Este método, quando aplicado a simulações de sistemas físicos se diferencia de outros, como o Método de Dinâmica Molecular (DM), visto que este não consiste em integrar diretamente as equações de movimento [2]. Esta abordagem permite uma maior flexibilidade no desenvolvimento de uma dinâmica eficiente para simulações do sistema, porém, em contrapartida, o tempo nestas simulações não corresponde ao tempo das simulações de DM.

Com o objetivo de estudar os comportamentos difusivos de gases de rede, abordaremos o método de MC utilizando as técnicas da computação em paralelo. O conceito de difusão é amplamente utilizado em muitas áreas da ciência como química, biologia, economia, e outras. Entretanto, a ideia central desta, é comum a todas: o objeto de interesse, com comportamento difusivo, se espalha a partir de um ponto de maior concentração para áreas de menor concentração. Na física, o fenômeno da difusão, verificado por exemplo em partículas de um gás (autodifusão, ou *self diffusion*), tem sido de suma importância na consolidação de teorias físicas como Movimento Browniano [3] e Teoria Cinética dos Gases [4]. Boltzmann, ao estudar comportamentos atômicos em processos de transporte macroscópicos introduziu a equação que leva seu nome, que serviu como base para o estudo destes fenômenos por mais de 140 anos. Atualmente, a teoria moderna sobre difusão foi desenvolvida por Albert Einstein, Marian Smoluchowski e Jean-Baptiste Perrin, principalmente através do estudo do Movimento Browniano [5].

Um dos modelos utilizados para simulações destes sistemas é o Gás de Rede (*Lattice*

Gas em inglês), que consiste em uma discretização do espaço ocupado pelas partículas de um sistema. Utilizando DM ou MC, a análise da difusão destas simulações pode nos trazer resultados muito semelhantes aos esperados em medidas experimentais [6]. As limitações das técnicas usuais começam a aparecer quando tratamos de sistemas maiores e mais densos. As simulações computacionais tradicionais não se mostram eficientes em execuções muito extensas, motivo pelo qual os pacotes de MD como LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator) e GROMACS (GROningen MAchine for Chemical Simulations) migraram para a computação em paralelo e obtiveram bons resultados com esta abordagem.

Em 1965, Gordon Moore, através de poucos dados disponíveis na época, mostra que havia tendência de que o número de transistores por chip, em um processador, dobraria a cada dois anos [7]. Como podemos ver na figura 1.1, com o passar dos anos, a predição de Moore se mostrou correta.

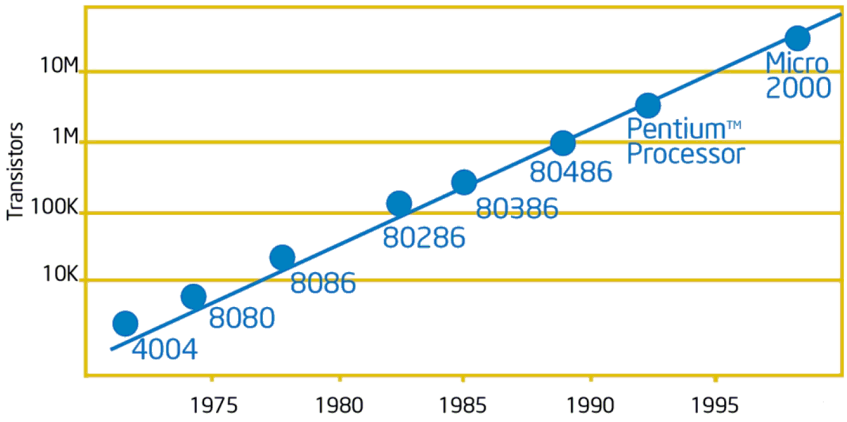

FIGURA 1.1. Número de transistores por ano, verifica-se que o comportamento é similar ao previsto por Moore. Imagem retirada da Ref. [8].

Este grande número de transistores foi utilizado com o simples objetivo de atingir uma mais alta performance. Porém, esta viria a um custo elevado no consumo de energia, como mostra a figura 1.2.

O comportamento exponencial para energia gasta por um processador, em relação a performance destes como eram construídos na época, ficou conhecido como barreira

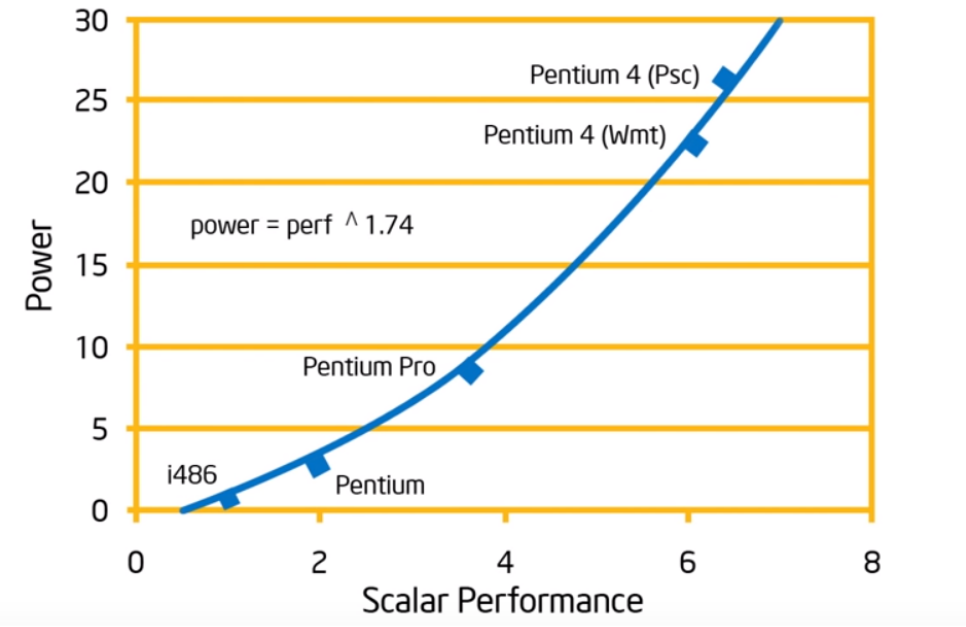

FIGURA 1.2. Energia consumida em função da performance normalizada pela performance mínima. Imagem retirada da Ref. [8].

de energia (*power wall* em inglês). Um limite explícito para o avanço tecnológico como se conhecia. A solução para este problema foi a criação de processadores com múltiplos núcleos e *threads* [1].

A transição para processadores *multithreads* ainda não está completa. Esta arquitetura depende muito mais do desenvolvedor para utilizar de maneira eficiente e inteligente o número de *threads* a sua disposição. Atualmente existe um grande esforço de empresas como Intel e Nvidia para que esta migração seja mais simples e eficiente. Uma observação importante é que o código deve ser desenvolvido de modo que não importa o número, ou ordem com a qual os *threads* realizem operações, a saída será a mesma. A utilização de *threads* de maneira simultânea, reduzindo o tempo de execução e a energia gasta para realizar um processo, é trabalho da programação em paralelo.

Os projetos *OpenMP*[2] e *OpenMPI* são as linguagens de programação em paralelo mais utilizada atualmente. Sua praticidade e simplicidade, faz com que usuários com pouca familiaridade com ambientes e notações complexas da linguagem *multithread*, se

[1] *Threads*, neste contexto são linhas de execução dentro do núcleo de um processador.
[2] Biblioteca para desenvolvimento de códigos utilizando *multithreading*. Disponível para C, C++ e Fortran.

adaptem facilmente à programação em paralelo.

Este trabalho tem como objetivo a implementação de algorítimos em paralelo, em simulações físicas que utilizam o método de MC. Esta abordagem é essencial já que em sistemas complexos o custo computacional das execuções em série pode ser muito grande. Primeiramente, através da aplicação ao conhecido Modelo de Ising [9] nos familiarizaremos com os conceitos da programação em paralelo, fazendo comparações com execuções em série cujos resultados já são bem conhecidos [10]. Após, será feita a aplicação dos conceitos e estratégias desenvolvidos para o estudo do impacto da paralelização na difusão em gases de rede com e sem interações. Apesar de parecer uma abordagem simples, não encontram-se artigos na literatura que o façam de forma simples e objetiva. Por hora, nossa contribuição será com simulações e evidências computacionais. Como desejamos estudar o comportamento difusivo destes sistemas, trabalharemos em regimes de baixa densidade e alta temperatura. Nestas condições, o sistema deve se encontrar em uma fase fluida, onde não há separação de fases, estado onde o comportamento difusivo é esperado.

Capítulo 2

Metodologia

Neste capítulo serão descritos os métodos utilizados para obtenção dos resultados deste trabalho, além de uma breve introdução a conhecimentos prévios necessários para compreender o andamento da pesquisa e a discussão dos resultados obtidos.

2.1 Simulações de Monte Carlo

As simulações de MC consistem na elaboração de um modelo para um sistema físico, executável em um computador, que nas devidas circunstâncias, reproduz resultados esperados na obtenção de variáveis e observáveis macroscópicos do sistema. a partir de configurações microscópicas. Para tal, fazemos o sistema modelo passar por uma variedade de estados de modo que a probabilidade p_μ deste se encontrar em um estado particular μ é igual ao peso ω_μ que o estado teria no sistema, em equilíbrio termodinâmico, que desejamos simular.

Para garantir este mapeamento entre modelo e experimento, devemos escolher uma dinâmica para nossa simulação, i.e. uma regra para mudar de um estado para outro durante a simulação, que resulte em cada estado aparecendo exatamente com a probabilidade esperada para este. As probabilidades de transição (ou regras de transição) $A(\mu \to \nu)$ de um estado para outro, nos ajudam a escolher os estados pelo qual faremos o sistema passar, e nestes estados faremos as medidas dos observáveis de interesse.

Na Mecânica Estatística de Equilíbrio, o espaço de fase apresenta 6^N coordenadas, onde N é o número de partículas [1]. Dado a dimensão deste, vemos que só é possível

amostrar uma pequena fração dos estados possíveis para o sistema, principalmente quando N é grande. A vantagem deste método é que esta pequena fração de estados, se escolhidos de maneira apropriada, devem ser suficientes para obtermos as medidas corretas. As medidas para valores esperados dos observáveis em nosso modelo, serão obtidas como médias destes sobre os estados pelo qual o sistema passa. A utilização de algoritmos em paralelo para simulações de MC possui premissas [11, 12]. O Método não depende de cálculos muito complexos, mas sim em uma grande repetição destes, fazendo com que a paralelização se mostre muito eficiente [13, 14]. Em contrapartida, trabalharemos com medidas de tempo em Monte Carlo *Steps* (ou *MCS*) e não com tempo mensurável em laboratório.

2.2 Balanço Global e Detalhado

Dada a importância das propriedades de balanço global e detalhado na maior parte dos processos Markovianos estudados neste trabalho, esta seção se dedica a expor as equações que regem esta. Cadeias de Markov, ou processos Markovianos, são processos estocásticos em sistemas discretos que em sua evolução, só há dependência do estado atual do sistema, e não de seus precedentes. O conceito de balanço detalhado foi introduzido por Ludwig Boltzmann, e seus principais argumentos se baseiam na reversibilidade de processos microscópicos[15].

Nos modelos que serão estudados neste trabalho, as condições de balanço global e balanço detalhado devem ser quase sempre satisfeitas para garantirmos a reversibilidade das cadeias de Markov simuladas. Um processo Markoviano é chamado de reversível exatamente quando satisfaz as equações de balanço detalhado. A condição de balanço global nos diz que existe um estado de equilíbrio para o processo Markoviano que satisfaz a seguinte premissa: as taxas com que o sistema faz transições para dentro e fora de cada estado μ devem ser iguais. Matematicamente, isto implica em satisfazer a seguinte equação,

$$\sum_{\nu} p_{\mu} A(\mu \to \nu) = \sum_{\nu} p_{\nu} A(\nu \to \mu) \quad . \tag{2.1}$$

Para garantir que os estados gerados serão compatíveis com a distribuição estatística esperada, uma segunda condição deve ser satisfeita. As taxas com o qual o sistema faz transição de um particular estado μ para um estado ν deve ser igual à taxa com a qual o sistema faz transição do estado particular ν para o estado μ, ou matematicamente,

$$p_\mu A(\mu \to \nu) = p_\nu A(\nu \to \mu) \ , \tag{2.2}$$

que é exatamente a condição de balanço detalhado que deve ser satisfeita em nossos modelos.

2.3 Algoritmo de Metropolis

John von Neumann foi o primeiro a estudar algoritmos de rejeição [16], especificamente na simulação de transporte de nêutrons. A ideia básica de Von Neumann consistia em: para amostrar uma distribuição de probabilidades específica, basta amostrar uma distribuição qualquer, desde que mantenhamos apenas as boas amostras. O Algoritmo de Metropolis, introduzido por Nicholas Metropolis e seus colaboradores em 1953 [17], é formulado como um caso particular dos algoritmo de rejeição. Neste, o critério de aceitação é utilizado de forma a satisfazer a condição de balanço detalhado, gerando passos em uma cadeia de Markov. Como nos modelos que serão estudados consideramos o sistema em contato com um reservatório térmico a uma certa temperatura T, esperamos que a distribuição de probabilidades gerada no equilíbrio seja a distribuição de Boltzmann no *ensemble* canônico ($e^{-\frac{E}{k_B T}}$).

Para satisfazer tais condições, o seguinte critério de aceitação foi proposto,

$$A(\mu \to \nu) \ = \ \begin{cases} \exp(-\frac{\Delta E}{k_B T}), & \text{se} \ \ \Delta E > 0 \\ 1, & \text{se} \ \ \Delta E \leq 0 \end{cases} \ , \tag{2.3}$$

que nos diz que a transição de um estado μ para um estado ν depende apenas da diferença de energia ΔE entre estes estados, da temperatura de banho térmico T a qual o sistema é sujeito, e da constante de Boltzmann k_B. Caso $\Delta E \leq 0$, a transição entre estados é sempre aceita. Caso contrário, a probabilidade de transição será dada pelo fator de Boltzmann relacionado à diferença de energia entre os estados. Satisfazendo não só as condições de balanço global e detalhado, como também de ergodicidade do sistema[1], já

[1] Neste caso, cada estado deve ser acessível por todos os outros em um número finito de passos.

que as transições, neste caso, são processos reversíveis. Colocando as informações de peso (probabilidade) de cada estado nas transições, o valor esperado para um dado observável poderá ser tomado como uma média simples deste, sobre os estados pelo qual o sistema passa.

2.4 *Multithreading* e Paralelização

Com o objetivo de facilitar o estudo posterior de sistemas complexos, serão desenvolvidos algoritmos de execução em paralelo para aplicação ao modelo de Ising (capítulo 3) e ao gás de rede (capítulo 4). A paralelização de algoritmos na maior parte dos casos não é trivial e depende do sistema com o qual se trabalha. Cabe ao desenvolvedor, elaborar uma estratégia de paralelização adequada ao problema. Tendo como base o Modelo de Ising 2D em uma rede quadrada, serão desenvolvidos estudos sobre os principais conceitos e métodos de aplicação da paralelização, estudando o impacto desta nas propriedades estatísticas do sistema, como ergodicidade e balanço global. Após, utilizando a ferramenta *OpenMP*, serão desenvolvidos códigos paralelizados com diferentes estratégias de evolução. Os resultados para as simulações geradas por estes distintos códigos serão comparados com a simulação em série para validação destes.

2.4.1 Estratégia de Paralelização

Como visto previamente, a maior dificuldade na elaboração de algoritmos de execução em paralelo para simulações físicas é, definitivamente, a criação de uma estratégia de paralelização (dependente do sistema) que não altere as propriedades estatísticas esperadas do sistema, ou que as altere para uma forma conhecida. Além disso, a estratégia escolhida tampouco deve permitir que haja concorrência por memória entre *threads* distintos, situação chamada de *data racing*[2].

Um dos tipos de estratégias mais utilizados na paralelização de sistemas de rede, é a decomposição em domínios, que se baseia em encontrar uma forma de decomposição

[2] Situação onde mais de um *thread* tenta alterar a posição na memória simultaneamente. Resultados devidos à essa condição são imprevisíveis, uma vez que não se sabe qual *thread* acessará primeiro a memória (ver exemplos nas Ref [18, 19]).

espacial que garanta independência entre determinadas regiões. Exemplos desta, na rede quadrada, podem ser vistos na figura 2.1. Escolhendo a decomposição de maneira correta, garantimos que não haverá atualização mútua de vizinhos, além de evitarmos *data racing*. Este tipo de estratégia nos será útil para a paralelização dos modelos.

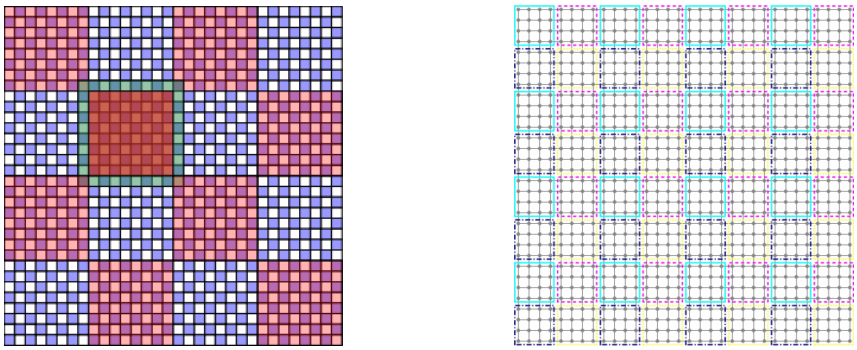

Figura 2.1. A figura da esquerda foi retirada da Ref. [18] e representa uma decomposição espacial chamada de *double checkerboard* utilizada no artigo citado para aplicação ao modelo de Ising. A direita, figura retirada da Ref. [20], que representa outro tipo de decomposição espacial chamado de *double tiling* utilizada no artigo para aplicação ao *public goods game*. Ambas as redes quadradas possuem tamanho $L = 32$.

2.4.2 Geradores de Números Aleatórios *Threadsafe*

Um problema surge na abordagem ao MC em paralelo: se a cada *thread* é atribuído um processo distinto, ao trabalharmos com estes de forma simultânea, esperamos que a sequência aleatória de números fornecida pelo gerador para cada um destes seja independente. A geradores aleatórios com esta característica, damos o nome de geradores *threadsafe*. Bibliotecas padrão de números aleatórios em C como, *"rand()"* e *"srand()"* não são geradores aleatórios que possuem esta característica. Isto ocorre pois threads em paralelo possuem a mesma *seed*, fazendo com que todos acessem a mesma sequência de números, além de alterarem o estado do gerador sem controle da ordem, fazendo com que as propriedades aleatórias deste possam ser alteradas. Outro problema relacionado aos geradores padrão é que, por serem uma função externa ao código, sua performance quando acessada simultaneamente por threads não é eficiente. Sua utilização sistemática em paralelo faz com que simulação fique mais lenta que a executada em série.

Para contornar tal problema, foram desenvolvidos incontáveis tipos de geradores

aleatórios, como por exemplo, os das referências [21, 22]. Nesta monografia, em específico, foi utilizado o gerador de números mais simples possível, um Gerador Congruente Linear (LCG em inglês), como uma função interna do código. Durante o laço de MC realizado na simulação, será dado a cada *thread* uma *seed* distinta, fazendo com que as sequências geradas sejam independentes, e que as variáveis de estado dos geradores não sejam compartilhadas.

2.5 Medidas

Nesta seção discutiremos algumas das medidas relevantes a serem realizadas sobre os sistemas estudados. Abordaremos brevemente como serão obtidos os resultados de interesse.

2.5.1 Tempos de Correlação

Como neste trabalho medimos valores esperados para observáveis de interesse assumindo que o sistema tenha alcançado um equilíbrio, aprofundar o conhecimento em relação a função de autocorrelação entre medidas de uma amostra é essencial uma vez que estas não são independentes, o que por sua vez afeta os estimadores. Esta grandeza nos dá informações sobre a quantidade de *MCS*, entre medidas, necessários para que estas possam ser consideradas independentes do ponto de vista estatístico.

A definição de função de autocorrelação para uma série temporal de um dado observável O, a um dado passo de tempo t desta série em relação a um tempo t' anterior, é

$$C_O(t) = \frac{\langle O(t'+t)\, O(t')\rangle - \langle O\rangle^2}{\langle O^2\rangle - \langle O\rangle^2} \quad . \tag{2.4}$$

Veremos na seção 3.4, que o comportamento desta função para uma série suficientemente grande pode ser dado pela seguinte equação,

$$C_O(t) \approx \exp\left(-\frac{t}{\tau_c}\right) \quad , \tag{2.5}$$

onde τ_c, chamado de tempo de autocorrelação, é o tempo (medido aqui em *MCS*) que a correlação entre medidas de um mesmo observável leva para cair a $1/e$ de seu valor inicial.

Normalmente é mais conveniente trabalhar com a função de autocorrelação integrada τ_{int} definida como,

$$\tau_{int}(t) = \frac{1}{2} + \sum_{t'=0}^{t'=t} c(t') \quad . \tag{2.6}$$

Isto ocorre devido ao fato de que quando $C_O(t) \to 0$, τ_{int} converge para um valor constante, o que facilita a sua obtenção numérica sem a necessidade de procedimentos de ajustes (*fitting*).

Pode ser mostrado que o número efetivo de amostras independentes, S_{eff}, entre as S realizadas é dado por

$$S_{eff} = \frac{S}{\tau_{int}} \quad , \tag{2.7}$$

onde S, nas simulações, é o tamanho da série temporal, que deve ser suficientemente grande para satisfazer a equação 2.5.

Um menor tempo de correlação implica em um aumento na eficiência estatística do algoritmo, já que medidas precisas podem ser realizadas em intervalos de tempos menores, diminuindo assim, o tempo de simulação necessário. Através da comparação entre simulações, tempos de correlação entre suas medidas, e seus respectivos tempos de execução, estimaremos a eficiência estatística e computacional dos códigos desenvolvidos em paralelo.

2.5.2 Deslocamento Quadrático Médio (*MSD*)

Outro conceito importante em nosso estudo será o de deslocamento quadrático médio (*MSD* em inglês). Em mecânica estatística, deslocamento quadrático é uma grandeza que dá ideia do quão deslocada de sua posição inicial uma partícula está. O *MSD* é definido como a média dos deslocamentos quadráticos sobre todas as N partículas do sistema em dado tempo (medido aqui em *MCS*). Ou, matematicamente,

$$MSD(t) = \frac{1}{N} \sum_{i=1}^{N} \left(\vec{r}_i(t) - \vec{r}_i(0)\right)^2 \quad , \tag{2.8}$$

onde $\vec{r}_i(t)$ é a posição da i-ésima partícula, na rede, no instante de tempo t.

Dois resultados exatos para esta grandeza nos permitem calcular o coeficiente de difusão: *(i)* o caminhante aleatório (*random walker*, *RW*) e *(ii)* equação de Langevin para

uma partícula livre. No primeiro caso, a distribuição de probabilidade[3] de encontramos uma partícula na posição (x, y) no tempo t dado que esta se encontrava na origem no tempo $t = 0$ é dada por [23]

$$\rho(x, y, t) = \frac{1}{4\pi Dt} \exp\left(-\frac{(x^2 + y^2)}{4Dt}\right) \quad , \tag{2.9}$$

onde D é o coeficiente de difusão e variância da distribuição é justamente o MSD. No segundo caso, depois de integradas as equações de movimento e realizadas as médias sobre o ruído térmico [23], obtemos o seguinte resultado para o MSD em duas dimensões:

$$\left\langle (\vec{r}(t) - \vec{r}(0))^2 \right\rangle = 4D\left(1 - \frac{1 - e^{-\gamma t}}{\gamma}\right) \quad , \tag{2.10}$$

onde D é o coeficiente de difusão e γ o coeficiente de amortecimento.

Em ambos os casos, no *RW* sempre e na equação de Langevin para tempos longos, obtemos que o comportamento do MSD com o tempo é dado pela relação

$$\lim_{t\to\infty} MSD(t) = 4Dt \quad . \tag{2.11}$$

Empiricamente, esperamos que as partículas que constituem um fluido apresentem uma dinâmica difusiva, similar ao comportamento de uma única partícula. Como esta é uma medida que depende do tempo, neste caso trabalharemos com médias desta sobre mais de uma amostra. Definiremos então, o *MSD* médio em um dado tempo, como

$$\langle MSD(t)\rangle = \frac{1}{K}\sum_{j=1}^{K} MSD_j(t) \quad , \tag{2.12}$$

onde K é o número de amostras. Através de medidas deste observável, teremos noção do comportamento difusivo, ou não, das partículas do sistema na rede.

[3] A distribuição de probabilidade do *RW* é solução da equação de difusão: $\partial_t \rho = D\partial_x^2 \rho$, onde D é o coeficiente de difusão.

Capítulo 3

Aplicação ao Modelo de Ising 2D

O Modelo de Ising é um modelo estatístico para um ímã real, que passa por uma transição de fase na região de temperatura $T = T_c \approx 2{,}269$ (para $k_B = J = 1$), onde para temperatura superior ($T > T_c$) o sistema se encontra em uma fase paramagnética e para temperaturas inferiores ($T < T_c$), em uma fase ferromagnética. A premissa por trás deste, e da maioria dos modelos magnéticos, é que o magnetismo de um material é formado pela combinação dos momentos de dipolo magnéticos de muitos spins atômicos dentro do material. O modelo postula uma rede com um spin s_i por sítio, que representa uma média sobre os spins atômicos na região do sítio. Neste, spins assumem a forma mais simples possível, $s_i = -1$ (spin *down*) ou $s_i = +1$ (spin *up*), como exemplifica a figura 3.4 à esquerda. Por simplicidade, neste capítulo, as simulações realizadas serão feitas sem a aplicação de campo externo ($B = 0$) e supondo interações entre primeiros vizinhos, todas estas com mesma intensidade J. Logo, o Hamiltoniano com o qual trabalharemos será,

$$\mathcal{H} \;=\; -J \sum_{<ij>} s_i \, s_j \;\; . \tag{3.1}$$

Para a aplicação direta do algoritmo de Metropolis, nos interessa a diferença de energia ΔE entre dois estados distintos do sistema, gerados pela alteração de um único spin da rede (*single spin flip).* Esta diferença de energia é dada por,

$$\Delta E = H^{\nu} - H^{\mu} \;\; , \tag{3.2}$$

onde ν é o novo estado (proposto) do sistema, e μ seu estado antigo (atual).

Trabalhando em uma rede quadrada de dimensões $L \times L$ com condições de contorno

periódicas (*PBC*, em inglês), podemos reescrever esta diferença como:

$$\Delta E = -s_k^{\nu}\, J \sum_{j=1}^{4} s_j - (-s_k^{\mu}\, J \sum_{j=1}^{4} s_j) \ , \tag{3.3}$$

onde s_k^{ν} é o spin do sítio escolhido no estado ν, e a soma é realizada sobre os primeiros vizinhos ao sítio k na rede. No modelo utilizado, o novo estado do spin alterado só depende de seu estado anterior, de tal modo que $s_k^{\nu} = -s_k^{\mu}$, enquanto que todos os seus vizinhos são mantidos fixos (por esta razão o índice μ foi omitido). Logo, a equação anterior, em termos do spin do sítio antes da tentativa de *flip*, se reduz a

$$\Delta E = 2 s_k^{\mu} J \sum_{j=1}^{4} s_j \ . \tag{3.4}$$

Esta diferença de energia[1] fará parte da aceitação da transição dos movimentos propostos no algoritmo (equação 2.3). Cada *MCS* nestas simulações será composto por L^2 tentativas de movimento via algoritmo de Metropolis, e será utilizado como unidade de tempo.

Com o objetivo de validar os resultados dos códigos de execução em paralelo que serão desenvolvidos neste trabalho, abordaremos primeiro a simulação em série, para então comparar os resultados obtidos pelas distintas formas de paralelização.

3.1 Abordagem Serial

Utilizando um código desenvolvido para este trabalho, realizamos simulações de MC para diferentes tamanhos de rede com o intuito de comparar os dados obtidos com os resultados conhecidos, tanto numéricos quanto exatos [24]. Para obter os resultados que seguem, tanto a constante de acoplamento entre os spins, quanto a constante de Boltzmann foram tomadas como unitárias ($J = 1, k_B = 1$).

Como primeiro teste do código, foram amostradas as séries temporais referentes à figura 3.1. O gráfico da energia nos mostra um comportamento centrado em um valor bem específico com algumas flutuações estatísticas. Nas medidas para magnetização

[1] Os únicos valores de ΔE possíveis na rede quadrada, para o modelo de Ising com *single spin flip* (segundo a equação 3.4), são $-8J, -4J, 0, 4J, 8J$.

($M = \sum s_i$), temos um comportamento oscilatório, que flutua entre um valor positivo bem determinado e este mesmo valor com sinal oposto. Este é um comportamento característico do modelo estudado, dado que nossa dinâmica favorece spins de mesmo sentido, em uma vizinhança, independente da orientação destes. Estes estados, tanto com magnetização positiva quanto negativa, possuem a mesma energia. Comportamento este, esperado, já que esta simulação foi gerada para $T < T_c$. Estes saltos nos valores de magnetização são o motivo de em medidas deste observável, estarmos interessados apenas em seu módulo [1, 25]. As diferenças nas escalas temporais aparecem pois as flutuações na energia ficam mais visíveis em períodos de tempo menores.

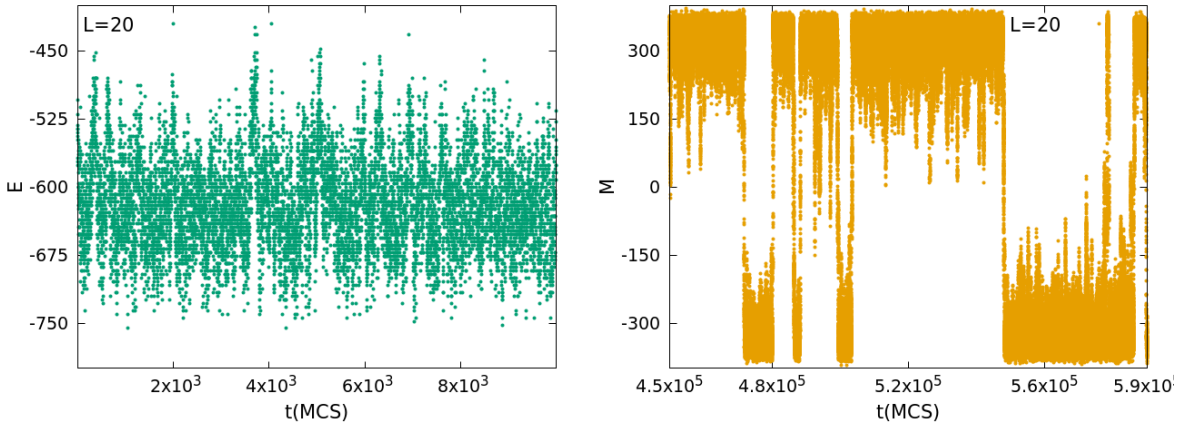

Figura 3.1. Séries temporais de energia e magnetização ao longo de uma simulação com 10^5 *MCS* para equilíbrio e 10^6 medidas consecutivas. Os trechos foram selecionados de forma a esclarecer o comportamento destes observáveis. A temperatura de banho térmico utilizada foi $T = 2{,}2$.

A figura 3.2, mostra a distribuição estatística de nossa amostra. Vemos que para energia, a distribuição é aproximadamente Gaussiana, característica do algoritmo de aceitação, que permite flutuações estatísticas em torno de um certo valor de energia (compatíveis com a temperatura T do banho térmico). À direita, vemos que a os valores para magnetização são bem localizados em dois extremos, padrão esperado para o modelo, dado o tamanho da rede [2].

Através da figura 3.3 podemos concluir que os resultados desejados são reproduzidos pelo código desenvolvido. Dada esta capacidade, podemos tomar o mesmo como referência

[2] Em redes maiores, não verifica-se o padrão. Isto ocorre pois a quantidade de tentativas de movimento necessárias para estes saltos aumenta com o volume da rede.

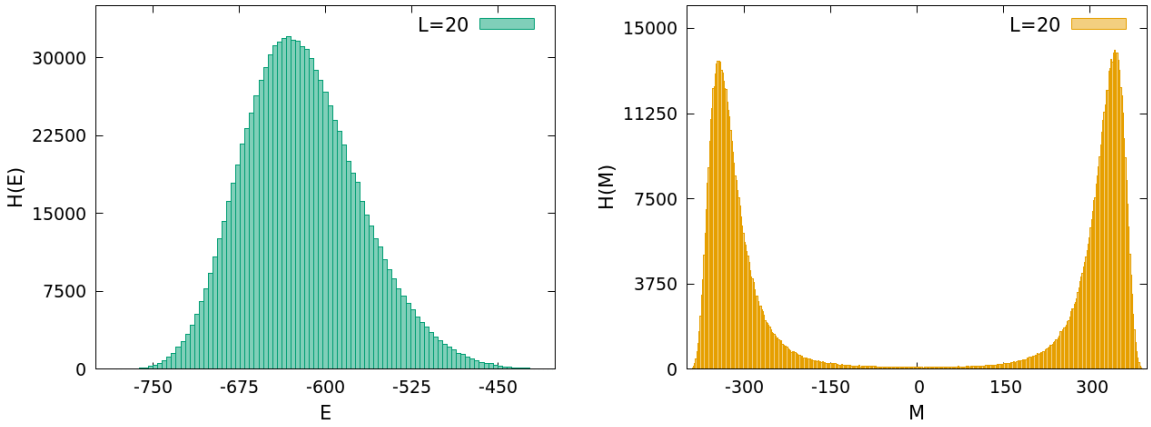

FIGURA 3.2. Histogramas para energia e magnetização gerados pelos dados mostrados na figura 3.1. Um comportamento similar a uma Gaussiana pode ser observado tanto à *esquerda* quanto à *direita*.

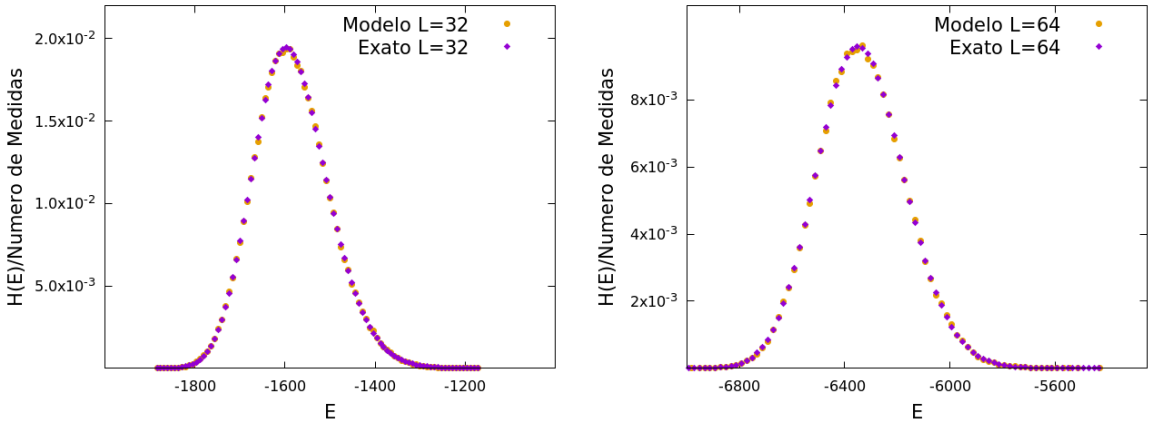

FIGURA 3.3. A figura apresenta comparação entre os resultados obtidos na execução com temperatura de banho térmico $T = 2{,}2$, e a solução exata proposta por [24] para dois tamanhos de rede. Podemos ver que os dados são reproduzidos com precisão, como esperado pelo modelo.

para comparação posterior com os resultados que serão obtidos através da paralelização.

3.2 Paralelização: Estrutura do tipo *Checkerboard*

Para evitar problemas como *data racing* e atualização de sítios vizinhos, devemos escolher com cuidado nossa estratégia de evolução. Utilizar a dinâmica atual, i.e. sortear um sítio aleatório entre L^2, independentemente, em múltiplos *threads*, implicaria não mais

em atualizarmos um sítio por vez, mas sim P sítios, onde P é o número de *threads* ativos. Como não foi adicionada restrição para seleção de sítios, eventualmente, em simulações extensas, como as de MC, dois ou mais *threads* podem sortear sítios vizinhos para evolução simultânea, ou até mesmo sortear o mesmo sítio para realizar a evolução. Ambos eventos devem ser evitados, devido a concorrência por memória e violação de balanço detalhado.

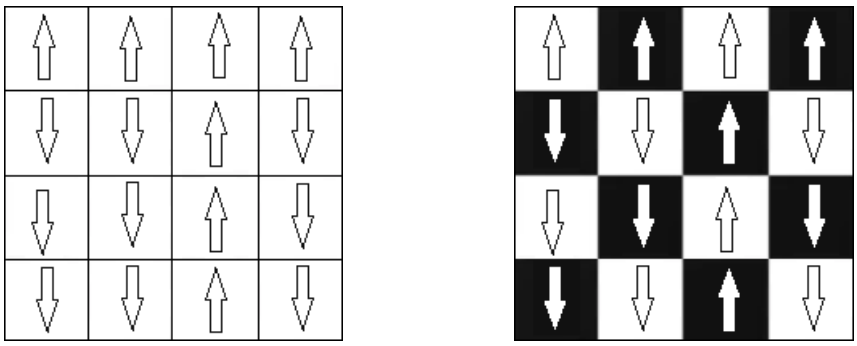

Figura 3.4. A aplicação do padrão *checkerboard* em uma rede 4×4 de Ising spins. À direita, fica claro que células de mesma cor podem sempre ser atualizadas simultaneamente, já que nunca são vizinhas.

Para evitar este tipo de problema, devemos pensar em uma estratégia de seleção de sítios que seja eficiente, prática e correta. Distintas estratégias de paralelização podem ser utilizadas em uma mesma dinâmica, tudo depende do sistema com o qual se vai trabalhar. A técnica de decomposição em domínios é uma das abordagens mais gerais utilizada em casos similares ao tratado neste trabalho [20]. Este tipo de decomposição também é comumente utilizado na paralelização através de *GPU's* ou *MPI*. Nesta seção, utilizaremos um tipo específico de decomposição em domínios, adequada ao nosso problema.

No algoritmo estudado, ao atualizarmos um sítio, não alteramos o estado de seus vizinhos[3]. Logo, mapeando a rede em uma estrutura similar a de um tabuleiro de damas (ver figura 3.4), e separando sítios em duas classes distintas, cores pretas ou brancas, temos a liberdade de utilizar uma dinâmica em paralelo. Nesta nova estrutura, sítios de mesma cor sempre poderão ser atualizados simultaneamente[4].

[3] Veremos, no capítulo posterior, que ao alterarmos o estado dos vizinhos, passamos a não poder mais utilizar de maneira eficiente a estrutura de *checkerboard*.

[4] Para o funcionamento correto da estratégia, a rede deve possuir lado par, já que para lado ímpar, células de mesma cor podem ser vizinhas.

Diferente da dinâmica do algoritmo de Metropolis serial, atualizaremos nesta seção $L^2/2$ sítios brancos aleatórios, em um primeiro laço, e em seguida $L^2/2$ sítios pretos aleatórios, evitando o acesso mútuo a vizinhos. Não foram incluídos critérios para a seleção de sítio durante a evolução, i.e. a cada passo (dentro de um único *MCS*), *threads* irão, em paralelo, selecionar um sítio aleatório dentre os $L^2/2$ sítios de mesma cor. O número de spins atualizados a cada *MCS* será conservado, logo, esperamos obter neste código paralelizado, resultados compatíveis com os obtidos em série, além de tempos de autocorrelação semelhantes entre as medidas.

	Tempo (segundos)				
Lado	Série	1 *thread*	2 *threads*	4 *threads*	8 *threads*
20	$5,5 \pm 0,5$	$6,2 \pm 0,5$	$4,0 \pm 0,5$	$2,7 \pm 0,5$	$2,2 \pm 0,5$
40	22 ± 1	24 ± 1	14 ± 1	8 ± 1	5 ± 1
100	144 ± 5	162 ± 5	92 ± 5	51 ± 5	29 ± 5
250	1032 ± 10	1195 ± 10	626 ± 10	330 ± 10	218 ± 10
500	4447 ± 50	5238 ± 50	2878 ± 50	1457 ± 50	900 ± 50

TABELA 3.1. Os resultados foram obtidos simulando 2×10^5 *MCS* em cada uma das redes. O aumento do tempo de simulação entre o algoritmo em série e em paralelo com 1 *thread* é esperada devido à necessidade da criação da região em paralelo que não é utilizada. A temperatura de banho térmico utilizada foi $T = 2{,}2$

A tabela 3.1 mostra os primeiros resultados obtidos para tempo de execução da simulação[5]. O erro utilizado nas medidas é referente ao desvio padrão da média sobre 10 repetições para tempos de execução das simulações com diferentes tamanhos de rede. Em redes maiores que $L = 20$, quando comparamos simulações em série com simulações com 8 *threads*, por exemplo, verifica-se uma redução de aproximadamente 80% no tempo de execução. Vemos também que em paralelo, uma rede de $L = 500$ executada em 8 *threads* pode ser simulada em $900 \pm 50s$, enquanto que uma execução em série para uma rede com metade do tamanho, leva $1032 \pm 10s$. Em redes menores, como $L = 20$, a melhoria também é evidente, porém em uma escala menor. Isto ocorre devido a um tempo mínimo de execução (dependente do processador com o qual se trabalha) responsável pela criação de regiões de execução em paralelo.

[5] Processador utilizado nas simulações: Intel® Core™ i7-3770 CPU @ 3.40GHz composto por 4 núcleos e com possibilidade de executar 8 *threads* simultâneas.

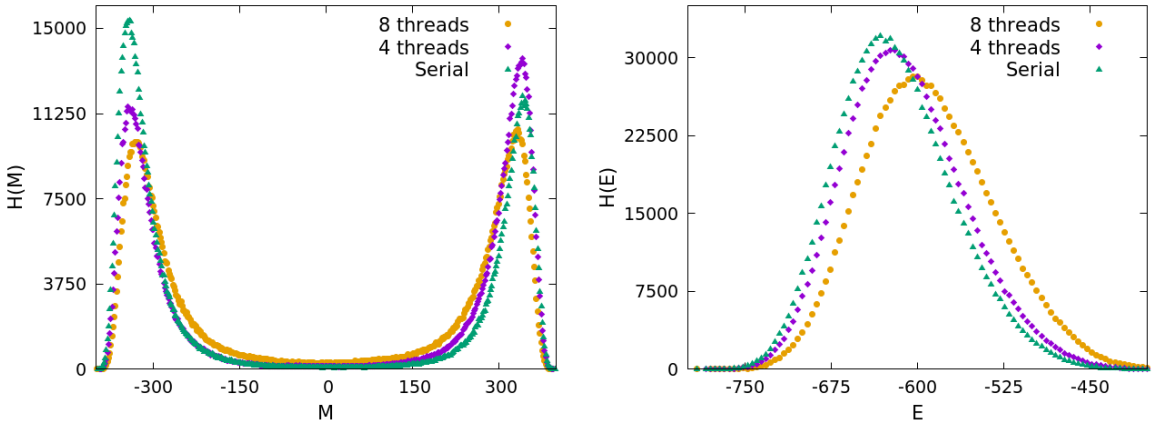

Figura 3.5. As figuras mostram histogramas para magnetização (M) e energia total (E) gerados com os resultados das simulações em paralelo usando 4 e 8 *threads* comparadas com o algoritmo em série para rede de lado $L = 20$. Foram utilizados 10^5 *MCS* até o equilíbrio e efetuadas 10^6 medidas por simulação à temperatura $T = 2{,}2$.

Na comparação quantitativa dos dados obtidos através desta primeira paralelização, com os dados gerados pela simulação em série, verificamos algumas discrepâncias entre os mesmos. Nos gráficos dispostos na figura 3.5, estas diferenças ficam mais explícitas. Olhando para o gráfico à direita, fica claro que há um deslocamento no pico da Gaussiana gerada pela simulação em paralelo.

Esta distinção nos resultados obtidos para os observáveis de interesse é consequência da condição, já discutida, de *data racing*. Até então, o método de seleção de sítios a serem atualizados, utilizado por cada *thread* era aleatório, permitindo que eventualmente o mesmo sítio fosse sorteado por dois ou mais *threads* ao mesmo tempo. Fica claro esta condição, quando passamos a pré-determinar a ordem de atualização de sítios (ver figura 3.6) para cada *thread*. Neste caso, os $L^2/2$ sítios de mesma cor foram dispostos em um laço ordenado sobre estes, de modo que cada passo consecutivo é entregue a um *thread* distinto.

Conforme vemos nas figuras 3.7, as fontes de erro relativas à implementação em paralelo, foram eliminadas quando alteramos o critério de seleção na atualização dos sítios. Com relação aos tempos de execução, podemos ver na tabela 3.2, que em relação ao código anterior (tabela 3.1), houve uma redução considerável, consequência do fato de não mais utilizarmos a cada passo um gerador de números aleatórios para a seleção de sítios a serem atualizados, e também por garantirmos que não haverá *data racing*.

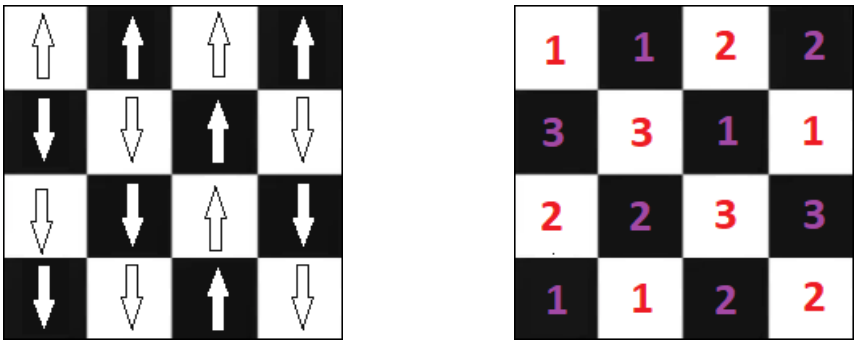

FIGURA 3.6. À direita, representação da pré-seleção da ordem de atualização dos sítios para 3 *threads* em uma rede 4×4. Os números indicam o *thread* responsável pelo sítio.

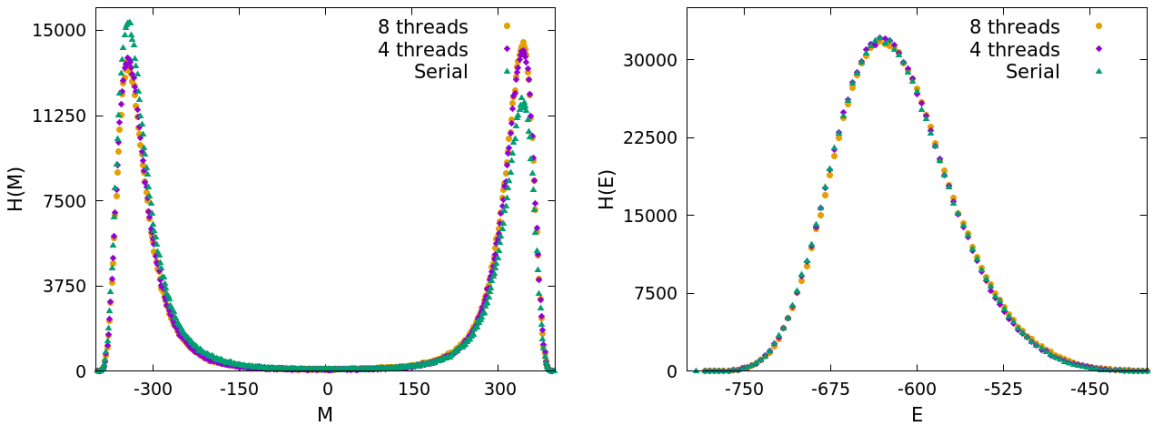

FIGURA 3.7. Histograma para simulações ($L = 20$ e $T = 2{,}2$) utilizando a decomposição em *checkerboard*, porém, selecionando previamente a ordem de atualização dos sítios da rede. A direita, vemos que a nova estratégia se mostra suficiente para reproduzir os resultados esperados para a simulação, enquanto a esquerda, devido a natureza da medida (saltos de M a $-M$), o comportamento do observável no histograma depende de em qual ponto da simulação paramos de realizar as medidas, e de sua dinâmica. Se o tempo de simulação fosse muito grande, $t \to \infty$, os picos teriam a mesma altura.

Com a proposta de aumentar a carga computacional em cada *thread*, criando um trecho grande o suficiente para diluir os efeitos de agendamento e carregamento, abordaremos uma dinâmica que não abre mão da aleatoriedade na seleção de sítios a serem atualizados.

	Tempo (segundos)			
Lado	2 *threads*	4 *threads*	8 *threads*	8 *th* (tab. 3.1)
20	$2,9 \pm 0,5$	$2,1 \pm 0,5$	$2,0 \pm 0,5$	$2,2 \pm 0,5$
40	10 ± 1	6 ± 1	4 ± 1	5 ± 1
100	58 ± 5	34 ± 5	23 ± 5	29 ± 5
250	366 ± 10	212 ± 10	141 ± 10	218 ± 10
500	1438 ± 50	860 ± 50	567 ± 50	900 ± 50

Tabela 3.2. As primeiras colunas mostram resultados para tempo de execução de simulações ($T = 2{,}2$) com 2×10^5 *MCS*, utilizando seleção prévia da ordem de atualização. Para comparação, a última coluna é referente à execução da tabela 3.1, para comparação.

3.3 Paralelização: Rearranjando a estrutura

A estratégia de *checkerboard* conseguiu resolver o problema de *threads* acessando sítios vizinhos simultaneamente. Porém, sem pré-ordenar a seleção de sítios, vemos que eventualmente, no grande número de *MCS* necessários para a simulação, dois ou mais *threads* sortearão o mesmo sítio para atualização.

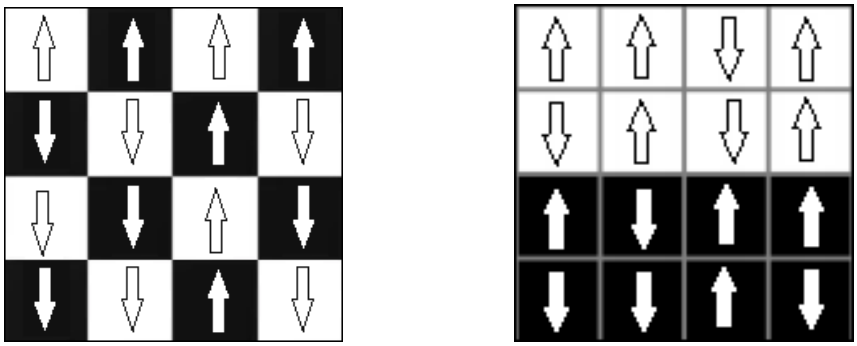

Figura 3.8. À direita, representação da reorganização da estrutura de *checkerboard*, onde a estrutura foi disposta em linhas compostas de sítios de mesma cor. Na dinâmica, cada sítio ficará responsável pela atualização de uma linha.

Com o objetivo de eliminar a condição de *data racing*, sem abrir mão da aleatoriedade da dinâmica, foi desenvolvida uma rede auxiliar de mesmo tamanho da original. Agora, além de classificarmos os sítios por cor, estes também foram rearranjados na nova rede de modo que em uma mesma linha, apenas sítios de mesma cor estejam presentes[6],

[6] Assim como na seção anterior, aqui devemos trabalhar com redes de lado par para garantir a

conforme mostra a figura 3.8.

Nesta nova estrutura, será utilizado um critério de seleção distinto para atualização dos sítios. Um *MCS*, nesta nova dinâmica, implicará em todas as L linhas desta nova matriz sofrerem L tentativas de atualização em suas colunas, por um mesmo thread. Em outras palavras, cada thread, será responsável pela atualização aleatória de uma linha em específico[7], sorteando a cada passo uma das L colunas desta. Este critério não permite mais que múltiplos *threads* sorteiem o mesmo sítio, já que cada sítio da rede original terá um único correspondente localizado em uma linha específica desta matriz auxiliar.

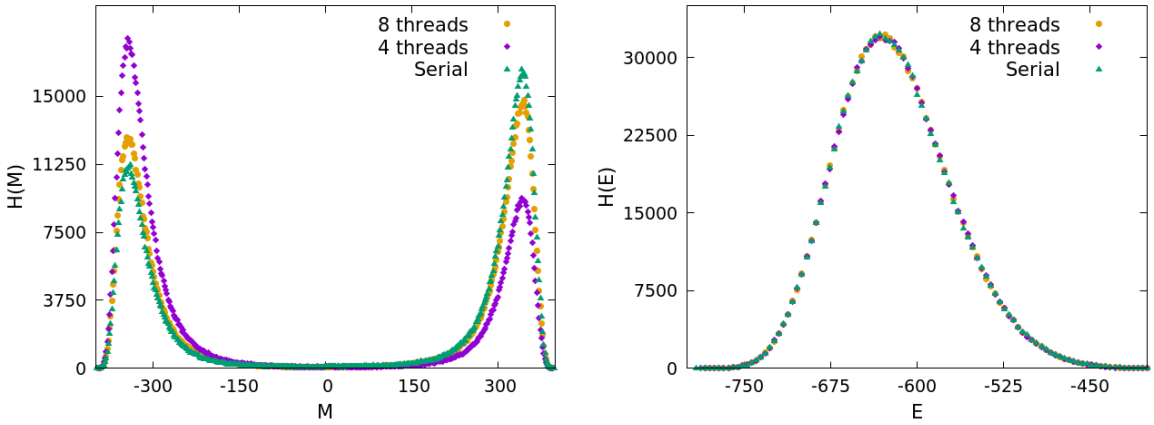

FIGURA 3.9. Histogramas referentes à simulações ($L = 20$ e $T = 2{,}2$) utilizando a nova estratégia de paralelização. A direita, verifica-se que a nova estratégia se fez suficiente para reproduzir os resultados esperados para a simulação. A esquerda, novamente não podemos tirar conclusões sobre eficiência devido à natureza da medida.

Os resultados obtidos para as simulações utilizando o código desenvolvido nesta seção estão dispostos na figura 3.9. Quando comparados com os resultados anteriores (figuras 3.2 e 3.7), vemos que esta nova abordagem reproduz os resultados esperados para os observáveis sem a necessidade de pré-ordenarmos a evolução dos sítios. Com relação aos tempos de execução, quando comparamos a tabela 3.3 com a tabela 3.2, fica claro que houve uma pequena perda de eficiência. Isto pode ocorrer devido ao fato de, no caso anterior, não necessitarmos de uma sequência de números aleatórios a cada passo, pois a ordem de atualização já era pré determinada.

validade da estratégia.

[7] Caso hajam mais linhas de mesma cor do que threads, um thread poderá ficar responsável por atualizar mais de uma linha.

	Tempo (segundos)				
Lado	2 *threads*	4 *threads*	8 *threads*	8 *th* (tab. 3.1)	8 *th* (tab. 3.2)
20	$3,7 \pm 0,5$	$2,5 \pm 0,5$	$2,3 \pm 0,5$	$2,2 \pm 0,5$	$2,0 \pm 0,5$
40	13 ± 1	7 ± 1	5 ± 1	5 ± 1	4 ± 1
100	82 ± 5	46 ± 5	29 ± 5	29 ± 5	23 ± 5
250	524 ± 10	281 ± 10	169 ± 10	218 ± 10	141 ± 10
500	2079 ± 50	1147 ± 50	672 ± 50	900 ± 50	567 ± 50

Tabela 3.3. Nas primeiras colunas, resultados para tempo de execução de simulações (T = 2,2) com 2×10^5 *MCS*. Para realização destas simulações foi utilizada a estratégia de reorganização da estrutura de *checkerboard*. Última colunas referentes as tabelas anteriores, para comparação.

Vale ressaltar que através da comparação do código desenvolvido, com um código semelhante publicado na rede[8] verificamos que nossas execuções tomavam muito mais tempo computacional. Investigando a fundo, o custo excessivo do uso explícito da função "exp()" na linguagem C, mostrou-se de suma importância na perda de eficiência. Devido ao número limitado de valores para diferenças de energia ΔE possíveis, existem alternativas eficientes para contornar este problema.

	Tempo (segundos)					
Lado	2 *threads*	4 *threads*	8 *threads*	8 *th* (tab. 3.1)	8 *th* (tab. 3.2)	8 *th* (tab. 3.3)
100	11 ± 1	6 ± 1	4 ± 1	29 ± 5	23 ± 5	29 ± 5
250	66 ± 2	34 ± 1	23 ± 1	218 ± 10	141 ± 10	169 ± 10
500	262 ± 5	141 ± 5	98 ± 5	900 ± 10	567 ± 50	672 ± 50

Tabela 3.4. Nas primeiras colunas, resultados para tempo de execução de simulações (T = 2,2) com 2×10^5 *MCS* utilizando o código da seção 3.2, cujos tempos de execução são os menores, utilizando uma tabela para obtenção das exponenciais das diferenças de energia. Última colunas referentes as tabelas anteriores, para comparação.

Ao criar uma tabela para os valores energia e suas respectivas probabilidades de aceitação, o tempo de execução das simulações foi reduzido em até $\approx 85\%$ em relação aos dados da tabela 3.2, que possui os menores tempos de execução, como mostra a tabela 3.4.

[8] `https://www.imsc.res.in/~sbhtta/ising_omp.c`

Até então, mostramos que a estratégia de paralelização escolhida é de suma importância não só na reprodução das distribuições estatísticas esperadas, mas também no tempo de execução das simulações. Porém, estes dois fatores não se mostram suficientes para decidirmos sobre a eficiência estatística da simulação. Além destes, devemos nos preocupar com os tempo de autocorrelação entre as medidas dos observáveis de interesse para os diferentes tipos de dinâmicas.

3.4 Tempos de Autocorrelação

Utilizando um procedimento baseado na Ref. [26] (assim como o erro associado a estas grandezas) e as séries temporais obtidas nas seções anteriores deste mesmo capítulo, determinaremos a função de autocorrelação (equação 2.4) entre as medidas para energia, $C_E(t)$, e magnetização, $C_M(t)$, nas diferentes execuções. Em seguida, obteremos a função de autocorrelação integrada τ_{int} (equação 2.6) para cada um destes observáveis.

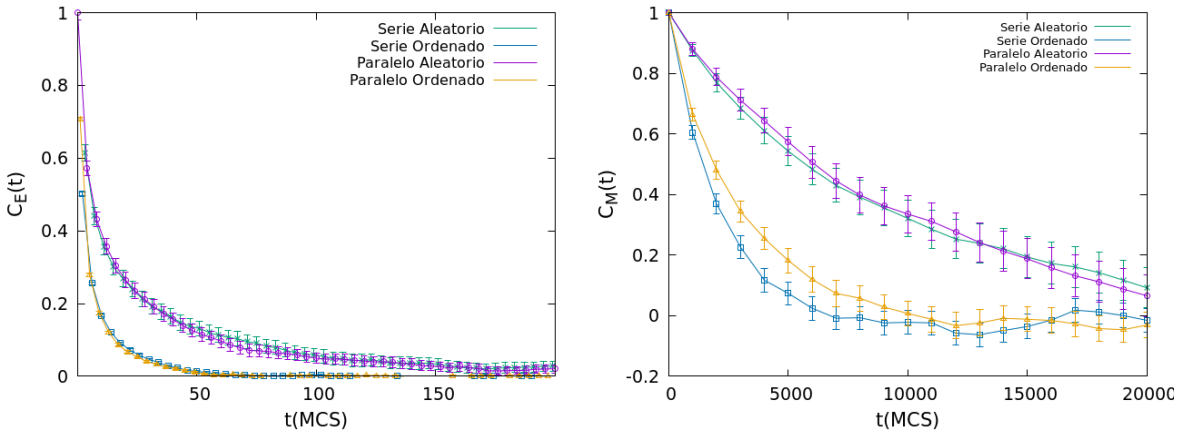

FIGURA 3.10. Funções de autocorrelação em função do tempo para as séries ($L = 20$, $T = 2{,}2$) de energia e magnetização obtidas na seção 3.1, com seleção de sítios pré-ordenada e aleatória, e nas seções 3.2 e 3.8 cujas seleções são pré-ordenada e aleatória, respectivamente. A diferença entre as escalas de tempo, similar às da figura 3.1, dos gráficos é devida a magnetização ser uma medida que leva muito mais tempo para descorrelacionar.

Analisando a figura 3.10, podemos ver que a função de autocorrelação, tanto para energia quanto para magnetização depende mais do modo como é feita a seleção de sítios atualizados, do que do ambiente computacional na qual a simulação é executada (série

ou paralelo). A seleção de sítios pré-ordenada faz com que as medidas se descorrelacionem mais rápido, tornando os métodos que utilizam este tipo de seleção, mais eficientes estatisticamente, segundo a equação 2.7.

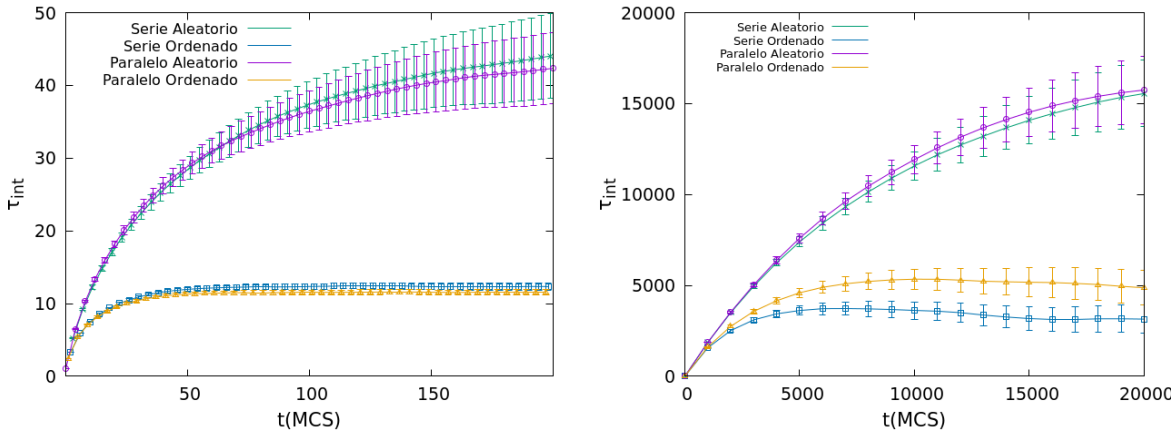

Figura 3.11. A figura mostra as funções de autocorrelação integrada (obtidas através dos dados da figura 3.10) em função do tempo para os diferentes observáveis em cada uma das execuções. Nota-se que a convergência para diferentes tipos de seleção de sítios se dá em tempos diferentes e para distintos valores de τ_{int}.

Para corroborar a análise anterior, utilizamos a função de autocorrelação integrada. Através dos gráficos dispostos na figura 3.11, podemos confirmar que o tempo de autocorrelação nas medidas realizadas pelas execuções com seleção pré-ordenada, se mostra bem menor em relação as outras execuções. Enquanto, nas medidas de energia em simulações pré-ordenadas, a correlação integrada converge a um valor aproximadamente constante em $t = \tau_c \approx 10$ *MCS*, nas outras execuções este tempo chega a $t = \tau_c \approx 40$ *MCS*. Uma redução de quase 75% nesta grandeza nos permite realizar medidas em intervalos menores de amostras, melhorando a estatística para mesmo tempo computacional. Com relação às medidas de magnetização os resultados são similares: enquanto o tempo de autocorrelação na execução pré-ordenada é da ordem de $\tau_c \approx 3000$ *MCS*, nas simulações com seleção aleatória este tempo é aproximadamente $\tau_c \approx 15000$ *MCS*, cerca de $5\times$ maior. Vale apontar que quando comparamos as execuções com seleção pré-ordenada nos diferentes ambientes, vemos que o tempo de correlação da simulação em série ainda é ligeiramente menor do que a execução em paralelo, porém com as barras de erros superpostas.

As séries temporais dispostas na figura 3.12 ilustram o resultado obtido para tempos

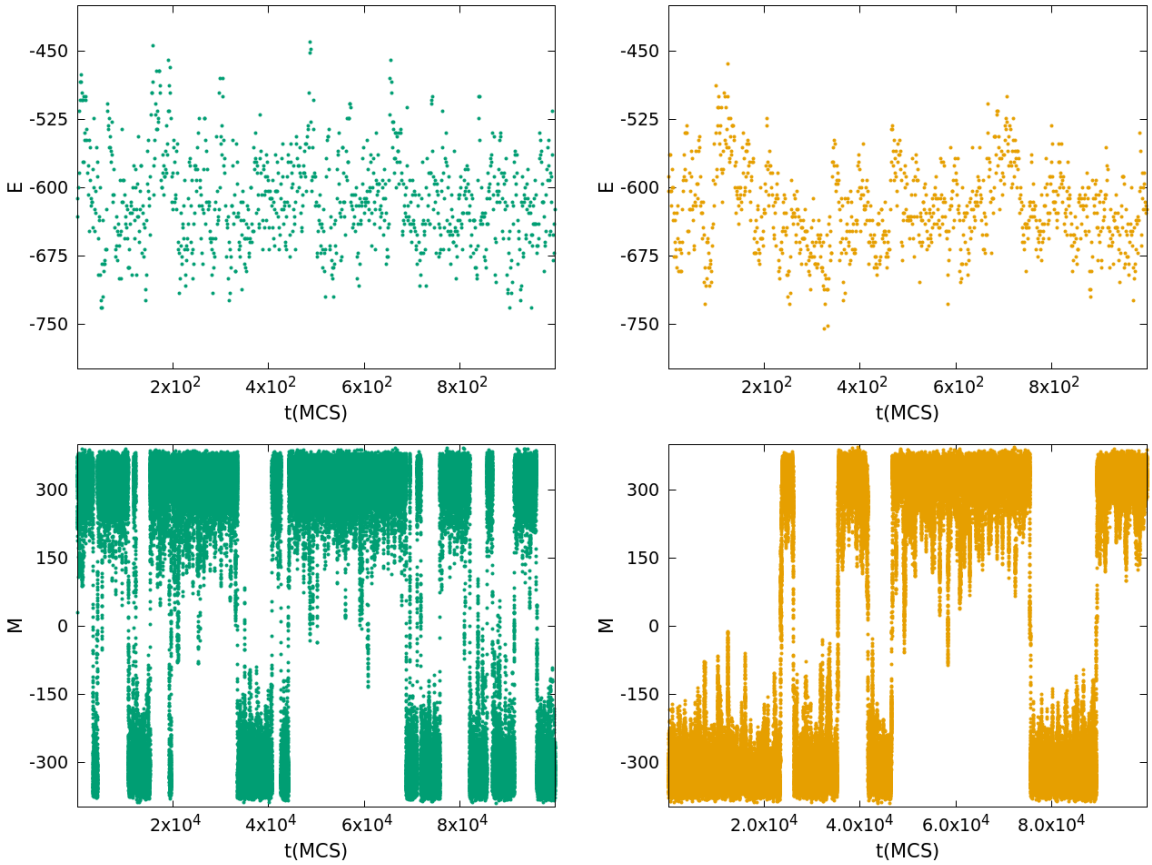

FIGURA 3.12. Trechos de séries temporais (10^6 *MCS*) para energia e magnetização ao longo de uma simulação ($L = 20$, $T = 2{,}2$) com 10^5 *MCS* para equilíbrio. A esquerda, séries referentes ao código em paralelo com atualização pré-ordenada. A direita, séries para execução em paralelo, porém com seleção de sítios realizada de forma aleatória. Em ambas as simulações o estado inicial do sistema era aleatório com mesma *seed*. Para energia, medidas significantes são aquelas próximas a $E = 600$, portanto, simulações com flutuações mais próximas a este valor, devem possuir tempo de correlação menor. No caso da magnetização, a média desta, sobre a simulação é $M = 0$, logo quanto maior a frequência das flutuações menor o tempo de correlação.

de autocorrelação. Com relação as séries para energia, observamos que no código com menor tempo de correlação, as medidas permanecem menos tempo nas regiões foram da média, permitindo um número maior de medidas independentes em um mesmo intervalo de tempo. No caso das medidas para magnetização, fica claro que os códigos com atualização pré-ordenada deixam de se correlacionar em um número de passos bem menor que nos outro caso, já que os saltos nos valores de M são muito mais frequentes neste. A figura 3.13 mostra o impacto desta diferença entre os tempos de correlação, do ponto de vista estatístico.

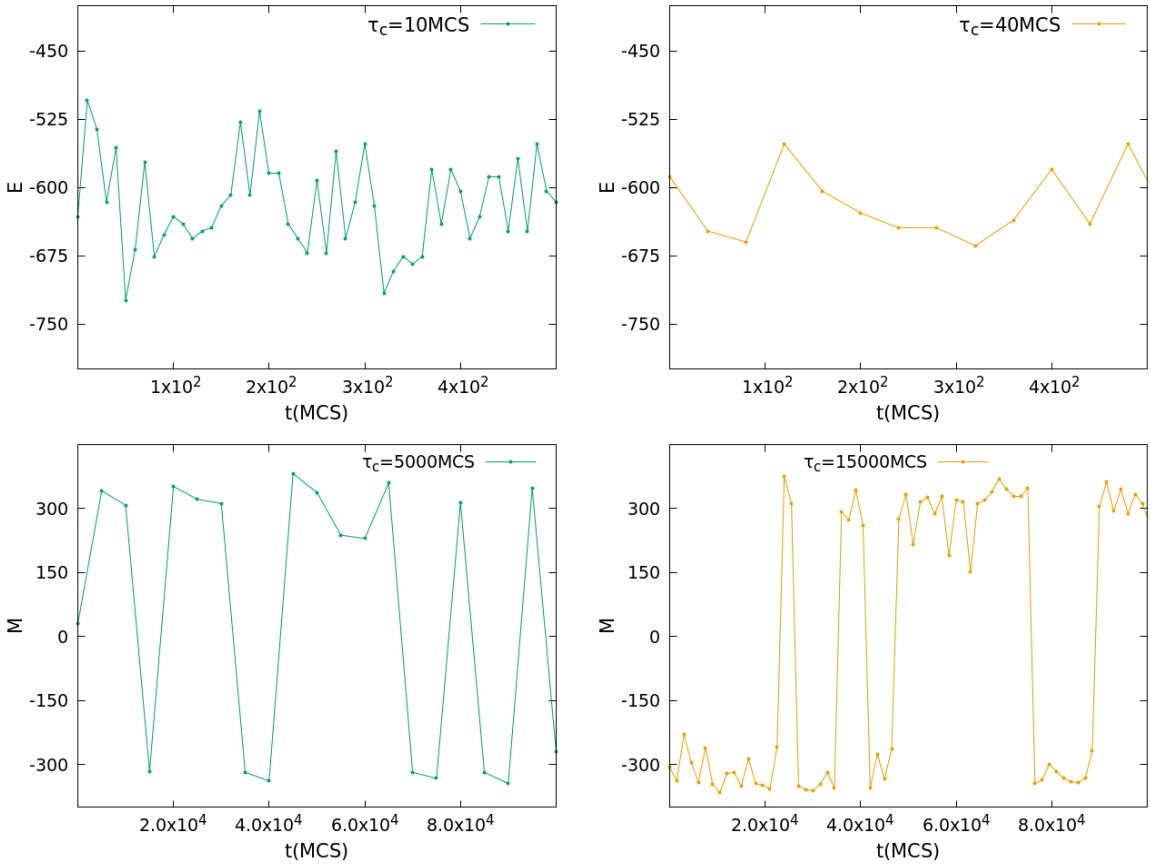

Figura 3.13. Séries temporais idênticas às dispostas na figura 3.12, porém, utilizando um intervalo de tempo $\Delta t = \tau_c$ entre as medidas, para cada um dos observáveis. Nota-se que as flutuações relevantes do ponto de vista estatístico ficam muito mais nítidas na execução à esquerda, onde a seleção de sítios é feita de forma pré-ordenada.

A figura 3.14 mostra que ao trabalharmos com módulo das medidas de magnetização, reduzimos o tempo de correlação entre as medidas consideravelmente, aproximando este dos valores obtidos para energia. Isto ocorre pois ao trabalharmos com módulo deste observável, deixamos de considerar os saltos entre valores positivos e negativos, o que fazia com que as medidas acabassem muito distantes do valor esperado.

Para continuarmos fiéis ao critério de eficiência que utilizamos, analisaremos os tempos de simulação referentes a códigos com o mesmo tipo de seleção. A tabela 3.5 mostra a comparação entre os tempos de execução para os códigos otimizados com seleção pré-ordenada de sítios, tanto em paralelo quanto em série, de onde estima-se que a paralelização pode fazer com o que o algoritmo seja executado até 10× mais rápido[9],

[9] Este é um resultado numérico válido para o ambiente computacional em que trabalhamos. O

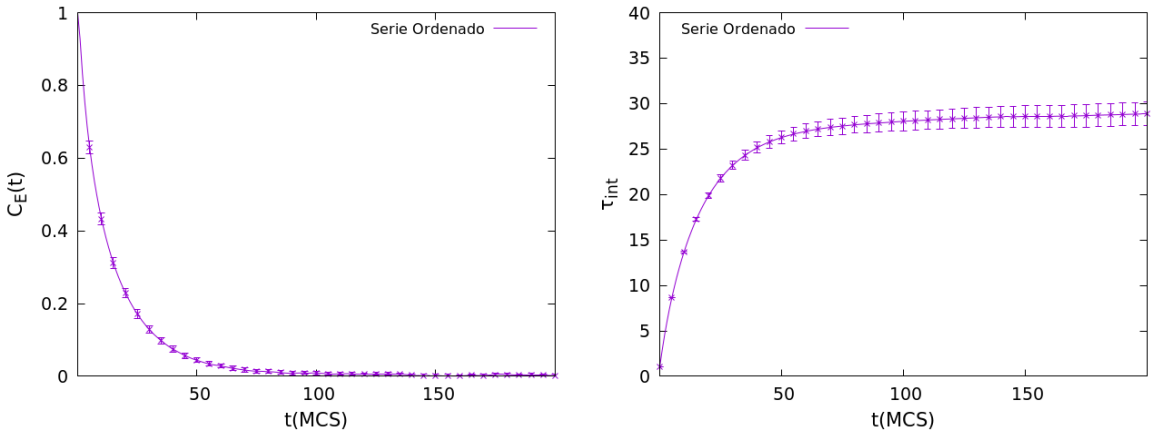

FIGURA 3.14. A figura mostra as funções de autocorrelação e autocorrelação integrada em função do tempo para o módulo da Magnetização do sistema. Nota-se que em comparação com os resultados das figuras 3.10 e 3.11, o tempo de correlação τ_c foi reduzido em aproximadamente 99%.

	Tempo (segundos)			
Lado	Série	2 *threads*	4 *threads*	8 *threads*
100	40 ± 1	11 ± 1	6 ± 1	4 ± 1
250	134 ± 1	66 ± 2	34 ± 1	23 ± 1
500	1112 ± 5	262 ± 5	141 ± 5	98 ± 5

TABELA 3.5. Resultados para tempo de execução de simulações ($T = 2{,}2$) com 2×10^5 *MCS* utilizando, em todos os casos, a seleção de sítios pré-ordenada e valores das probabilidades de transição guardados na memória (otimização da seção 3.3)

dependendo do tamanho da rede e do número de *threads* utilizados.

O parâmetro de eficiência (equação 2.7) para as execuções com seleção pré-ordenada, em relação a magnetização, é da ordem de $S_{eff} \approx 333$, enquanto para as com seleção aleatória este valor chega a $S_{eff} \approx 66$. Estes resultados nos mostram que em séries temporais de mesmo tamanho ($10^6 MCS$ no caso), algoritmos com seleção pré-ordenada podem realizar aproximadamente $5\times$ mais medidas independentes. Dito isto, pode-se inferir, que execuções com seleção pré-ordenada são mais eficientes, estatisticamente, do que códigos que utilizam seleção aleatória.

Através destes resultados, concluímos que, devido ao tempo de execução, e a efici-

aumento na eficiência depende do número de *threads* e da eficiência da CPU, permitindo que esta melhora na performance possa ser ainda maior.)

ência estatística, o código desenvolvido na seção 3.2, representa a simulação desenvolvida mais eficiente, tanto computacional quanto estatisticamente.

Capítulo 4

Aplicação ao Gás de Rede

Diferente do modelo de Ising, onde todos os sítios da rede eram ocupados por partículas com spin $+1$ ou -1, no caso do gás de rede, apenas uma parte dos sítios será responsável pela dinâmica do sistema. Aqui, os sítios assumem valor 0 caso vazio, e 1 caso ocupado. Pela similaridade entre ambos os modelos, pode-se facilmente realizar um mapeamento entre estes. Mostra-se através deste, que a constante de interação entre partículas ϵ, no gás de rede, deve valer um quarto da intensidade de interação entre spins J, no modelo de Ising, tal que $\epsilon = J/4$. Isto faz com que o modelo possua a característica de passar por uma transição de fase na região de temperatura $T_c = (T_c)_{Ising}/4 \simeq 0,567$ (para $k_B = \epsilon = 1$). Para temperaturas acima desta região ($T > T_c$) o modelo deve se encontrar em uma fase gasosa, onde a interação entre as partículas não é forte o suficiente para alterar o movimento aleatório. Nestes regimes, esperamos que, em baixas densidades, o comportamento do *MSD* seja similar ao de um única partícula, como visto na seção 2.5.2. Para temperaturas abaixo da temperatura crítica ($T < T_c$) o sistema se encontra em uma fase onde há coexistência entre estado líquido e gasoso, com separação de fase. Nesta situação, o comportamento do *MSD* deve mudar (ver seção 4.2).

Assim como no modelo previamente estudado, trabalharemos aqui com o caso mais simples, para facilitar o entendimento. O Hamiltoniano que rege a dinâmica do modelo trabalhado é,

$$\mathcal{H} = -\epsilon \sum_{<ij>} \sigma_i \sigma_j \ , \tag{4.1}$$

onde ϵ é a energia de interação entre vizinhos e os σ_i as variáveis de ocupação dos sítios. Definindo $\epsilon \geq 0$, notamos que a energia de interação é atrativa, formando, em baixas temperaturas, regiões de aglomerados de partículas. Neste caso, a diferença de energia de

um estado para outro, devido à movimentação de uma única partícula da rede será dada por:

$$\Delta E = -\epsilon\, \sigma_k^{\mu}\Big(\sum_{<k',i\neq k>} \sigma_i - \sum_{<k,j\neq k'>} \sigma_j\Big) \ , \tag{4.2}$$

onde k é sítio selecionado para atualização, k' o sítio vizinho de destino da partícula e a soma sobre os vizinhos diferentes de k e k'. Trabalharemos neste capítulo também na rede quadrada, portanto L^2 é o número de sítios da rede, enquanto que $N = \rho L^2$ é o número de partículas ocupando a rede, e ρ a densidade. N é mantido fixo ao longo das simulações, ou seja, trabalhamos com o ensemble canônico. Como fica claro ao analisar a equação 4.2, a variação na energia depende da diferença no número de vizinhos ocupados entre os sítios de ocupação no estado atual e no estado proposto da partícula[1]. Caso o número de sítios vizinhos ocupados seja maior na posição futura da partícula, segundo o algoritmo de Metropolis (eq. 2.3), esta diferença de energia apresenta valor negativo, e como consequência a transição será sempre aceita.

No gás de rede, além de alterarmos o valor local do sítio sorteado durante a evolução (como no caso do modelo anterior), há a necessidade de alterarmos o valor de um de seus vizinhos também, já que estamos simulando o movimento de uma partícula. Novas formas de implementação em paralelo devem ser pensadas para nos adaptarmos ao novo modelo. Uma das propriedades dinâmicas importantes de sistemas como este é a difusão. O presente capítulo mostra o impacto da paralelização nesta grandeza.

4.1 Caso $\epsilon = 0$

Por questões de simplicidade, no primeiro momento trataremos o modelo de gás de rede sem interações entre as partículas vizinhas, a condição de ocupação de um sítio por uma única partícula continua válida. Neste caso, como podemos ver na equação 4.1 fazendo $\epsilon = 0$, temos um Hamiltoniano que não depende do estado atual do sistema, ou seja, trabalharemos com um sistema onde todos os estados possuem igual energia e probabilidade. A evolução de um sistema como este, no limite de baixa densidade, onde as interações por bloqueio podem ser desconsideradas, é dado simplesmente pela dinâ-

[1] Em uma rede quadrada, para o gás de rede, os valores possíveis de ΔE são $-3\epsilon, -2\epsilon, -\epsilon, 0, \epsilon, 2\epsilon, 3\epsilon$.

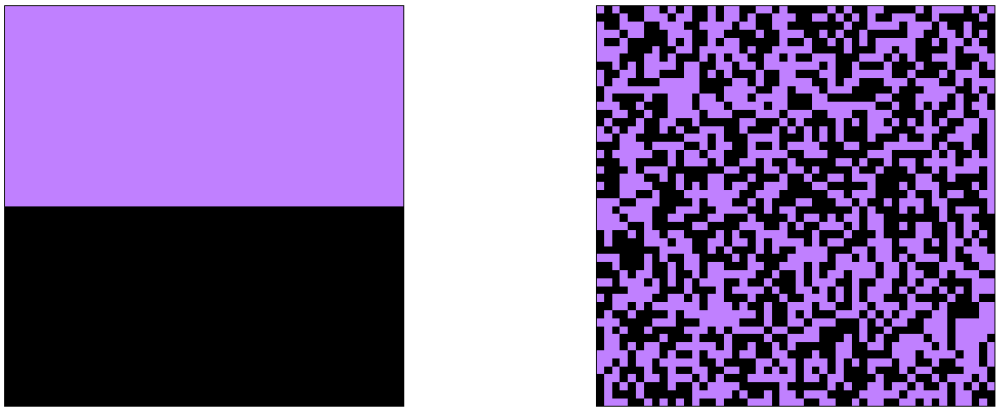

Figura 4.1. Snapshot de uma rede de tamanho $L = 50$, com densidade $\rho = 0{,}5$ em diferentes momentos da evolução do sistema. À esquerda, uma snapshot da rede antes de qualquer tentativa de evolução. À direita, sistema após 10^5 *MCS*, onde a distribuição de partículas é praticamente aleatória.

mica de um *RW* em 2 dimensões. Nestas condições, é esperado que o sistema apresente comportamento difusivo, dado que este é a solução exata do *RW*.

Assim como no modelo de Ising, antes de trabalharmos com algoritmos de execução em paralelo, devemos primeiro abordar as simulações em série, com o objetivo de termos uma base comparativa. Similar a abordagem inicial ao modelo de Ising, a evolução de nossa rede se dará de forma aleatória. Um *MCS* aqui, significará que houveram N tentativas de movimento em sítios ocupados aleatórios. Como critério de movimentação, damos liberdade para o modelo selecionar uma das 4 direções possíveis. Caso vizinho selecionado esteja desocupado, a transição será sempre aceita. A figura 4.1 nos dá uma ideia da dinâmica do sistema.

Uma das medidas relevantes a serem feitas sobre sistemas como este, é o *MSD*, obtido através das equações 2.8 e 2.12. Esta grandeza, mensurável em laboratório, nos dá noção do deslocamento médio, em relação a origem, das partículas de um sistema. Análises mais profundas sobre esta, nos levam a maior compreensão de comportamentos dinâmicos do sistema, sejam eles balísticos ou difusivos. Como esta é uma medida que, para tempos grandes, neste caso, deve apresentar comportamento linear em relação ao tempo, utilizaremos também a escala logarítmica, de modo que o comportamento desejado será dado por,

$$\log\ MSD(t) = \log(t) + \log(4D)\ . \tag{4.3}$$

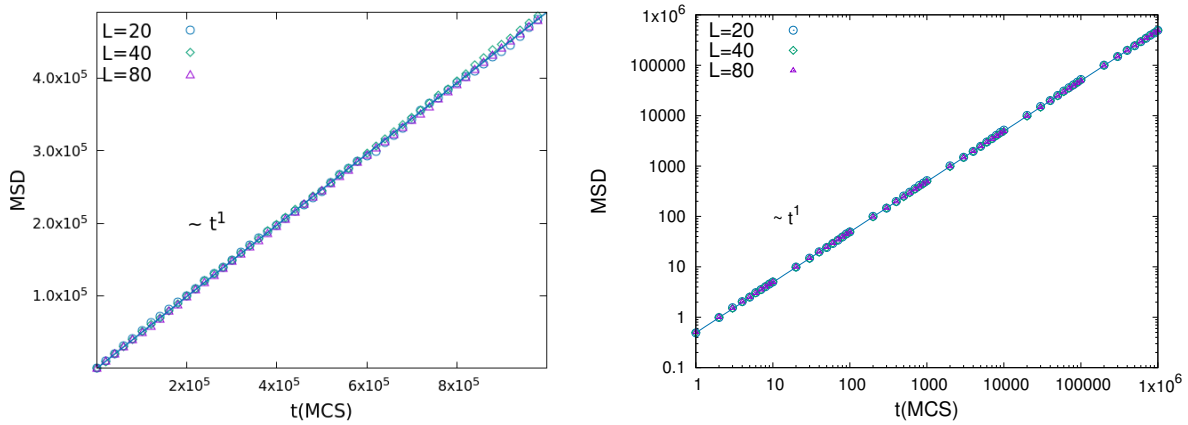

FIGURA 4.2. As figura mostra resultado obtido para as simulações executadas em série. Utilizando o modelo sem energia de interação entre as partículas, o comportamento difusivo é verificado. Foram utilizados densidade $\rho = 0{,}5$ e diferentes tamanhos de rede. No eixo das abscissas, a contagem de *MCS* é feita a partir do equilíbrio (10^5 *MCS*). A direita, ambos os eixos se encontram em escala logarítmica. A reta representa o ajuste linear realizado sobre as medidas, a inclinação da reta é $4D = 0,5337 \pm 9,4 \times 10^{-3}$.

Portanto, em escala logarítmica, esperamos que as medidas obtidas sejam ajustáveis a uma reta com inclinação 1. Olhando para a figura 4.2, podemos ver que o modelo executado em série, apresenta comportamento difusivo. Com o código base para comparação em funcionamento, passamos ao desenvolvimento de estratégias para a paralelização do algoritmo.

4.1.1 Paralelização: Rede Auxiliar

Analogamente ao modelo de Ising, onde utilizamos a decomposição espacial em *checkerboard*, foi realizada primeiramente uma tentativa de decomposição em domínios como estratégia de paralelização. Diferentemente, neste caso em específico, alterações no estado dos sítios vizinhos poderão nos causar *data racing*, como mostra a figura 4.3 à esquerda. Para contornar este problema, foi utilizada uma matriz auxiliar para destino das partículas na rede. Ao final de um *MCS*, a matriz destino será copiada sobre a rede original, de modo que cada sítio (destino) irá sortear apenas uma das partículas que se destinam a este. Só então o movimento é de fato realizado, como mostra a figura 4.3 a direita.

A rede foi decomposta em *checkerboard* e rearranjada em uma matriz de acesso

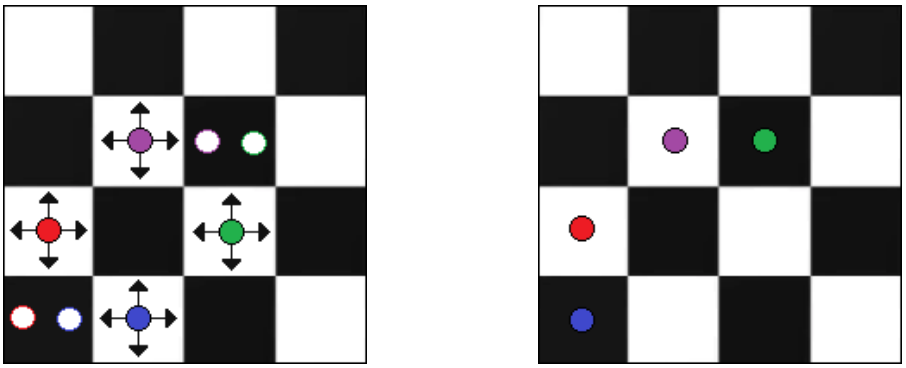

Figura 4.3. A figura mostra exemplo de concorrência por memória quando dois *threads* tentam acessar o mesmo sítio vizinho para troca. Círculos sem preenchimento representam a posição proposta das respectivas partículas. A direita, a rede auxiliar faz com que o sítio preto escolha, ao final de cada *MCS*, qual partícula irá realizar o movimento.

para os threads, assim como na seção 3.3. Diferente do capítulo anterior, cada *thread* irá atualizar N' vezes a linha de sítios de mesma cor que lhe foi dada, onde N' é o número de partículas nesta. Cada *MCS* nesta dinâmica, assim como na anterior, será contado como a tentativa de atualização de N sítios ocupados da rede.

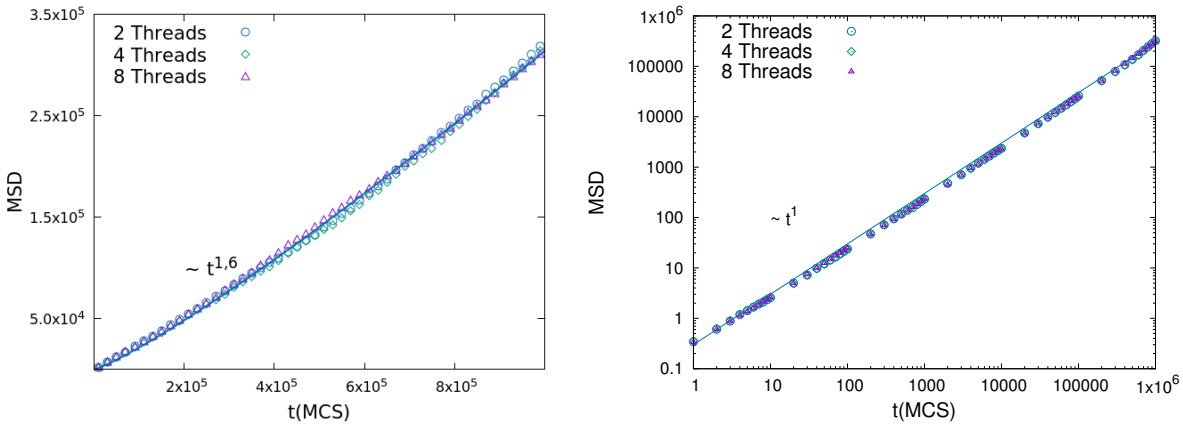

Figura 4.4. A figura mostra resultados obtidos para média do *MSD* sobre 10 simulações ($\rho = 0,5$ e $L = 20$) utilizando rede auxiliar para cada ambiente. A direita, ambos os eixos se encontram em escala logarítmica. A reta representa o ajuste linear realizado sobre as medidas, a inclinação da reta é $4D = 0,3000 \pm 1,2 \times 10^{-3}$

Os resultados obtidos utilizando esta abordagem, estão dispostos na figura 4.4. Analisando a figura da direita, vemos que o comportamento não difusivo, que leva o expoente do ajuste a ser $1,6$, só aparece para tempos muito grandes. Nesta região, devido

a estarmos realizando médias apenas sobre 10 amostras, a incerteza nas medidas deve ser muito grande. Por esta razão, o comportamento desta execução será considerado aqui como difusivo. Nota-se entretanto, que o coeficiente de difusão neste caso é menor do que no caso em série. Portanto, um indivíduo interessado em estudar sistemas com um impacto menor da difusão, poderia optar por usar este código.

4.1.2 Paralelização: Dinâmica Multidimensional

Para evitar a necessidade de uma rede auxiliar, uma nova estratégia de paralelização foi proposta. Devido à simetria do problema em 2D, uma abordagem com dinâmica multidimensional pode ser elaborada. Nesta, hipoteticamente, o movimento de uma partícula será decomposto em dois movimentos independentes. Em um primeiro momento, cada linha da rede será entregue a um *thread*. Este *thread* ficará responsável por contar o número de partículas N' nesta e em seguida, realizar N' tentativas de movimentos horizontais com as mesmas, sorteando aleatoriamente um sentido para movimento, com igual probabilidade. Deste modo, garantimos que não haverá competição por memória, já que cada linha é atualizada de forma serial.

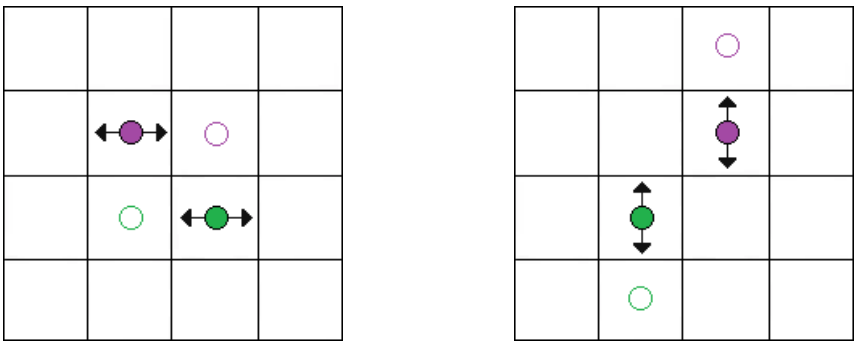

FIGURA 4.5. A figura exemplifica a independência entre as evoluções na horizontal e na vertical desta estratégia. Em cores distintas, partículas sendo atualizadas simultaneamente por diferentes *threads*.. Os pontos vazados indicam o estado proposto para a partícula.

Em um segundo momento, o mesmo processo será realizado para os movimentos na vertical, neste caso, a cada *thread* será entregue uma coluna da matriz, seguindo analogamente como no caso anterior. Um exemplo desta dinâmica[2] pode ser visto na

[2] Vale ressaltar que diferentemente das abordagens de *checkerboard* utilizadas anteriormente, esta

figura 4.5. Nesta dinâmica, um *MCS* contará como $2N$ tentativas de movimento, metade em cada direção.

Os resultados referentes às simulações utilizando a nova estratégia estão dispostos na figura 4.6. Como podemos ver, em comparação com as execuções anteriores (figuras 4.4 e 4.2) esta possui o maior coeficiente de difusão. Em escala logarítmica, podemos ver que o comportamento do *MSD* pode ser ajustado por uma reta com inclinação 1.

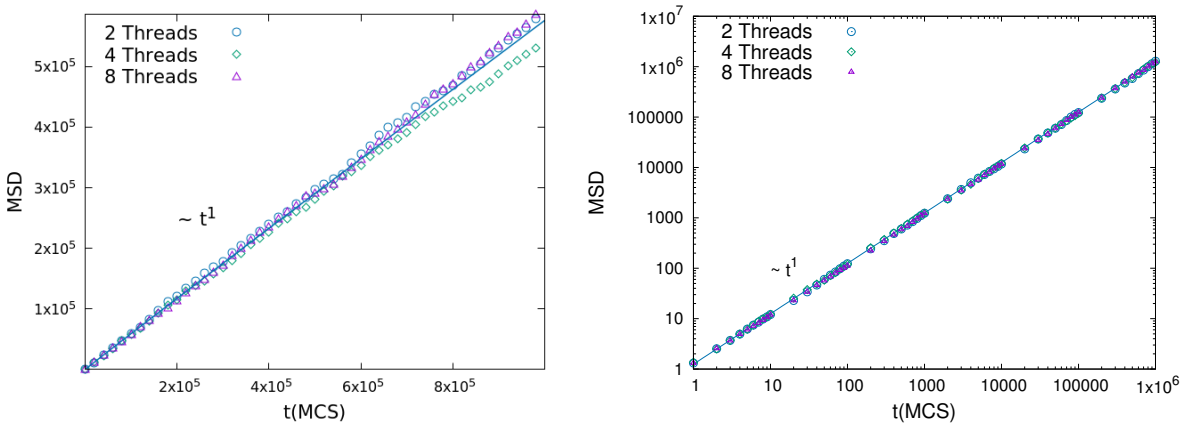

Figura 4.6. Resultado obtido para simulações ($\rho = 0,5$ e $L = 20$) executadas em diferentes ambientes, utilizando o algoritmo com independência entre os graus de liberdade do sistema. A direita, ambos os eixos se encontram em escala logarítmica. A reta representa o ajuste linear realizado sobre as medidas, a inclinação da reta é $4D = 0,5944 \pm 6 \times 10^{-4}$

No algoritmo desenvolvido em série, cada partícula selecionada, sorteava com igual probabilidade, um de seus 4 vizinhos para sua evolução. Isto significa que cada uma das transições possuía 25% de chance de ser a escolhida. Para garantir que não alteremos a dinâmica do sistema, satisfazendo as condições de balanço global e detalhado, devemos repensar as probabilidades de transição sob a qual a partícula é submetida.

Pensando em cumprir as condições acima citadas, repesamos as probabilidades de transição para cada partícula. Nas primeiras tentativas de evolução de uma partícula, incluímos a partir de agora, uma chance de 50% da partícula permanecer no estado em que se encontra, enquanto os outros 50% são distribuídos igualmente entre os movimentos em ambos os sentidos.

não depende da paridade dos lados da rede.

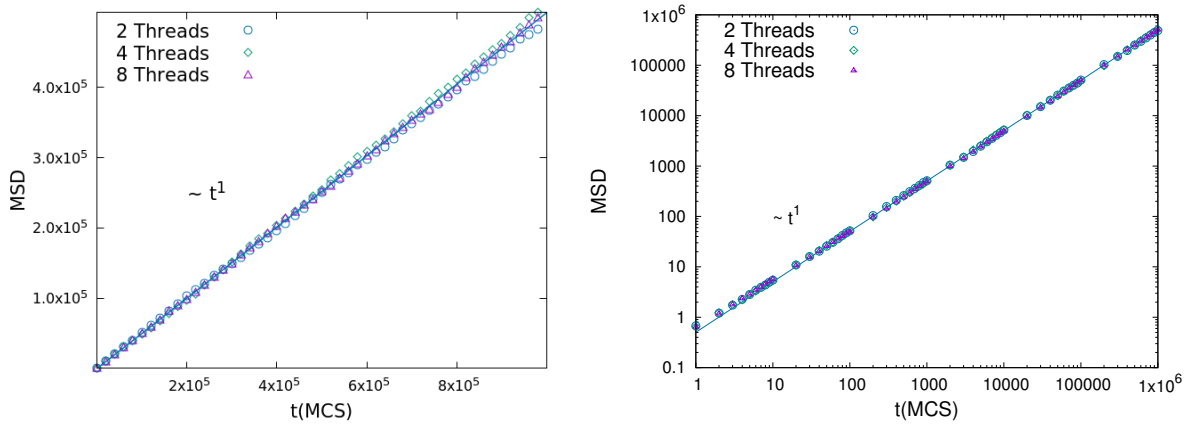

FIGURA 4.7. A figura mostra resultados obtidos para simulações ($\rho = 0,5$ e $L = 20$) executadas em diferentes ambientes, utilizando o algoritmo com independência entre os graus de liberdade do sistema com repesagem das probabilidade. A direita, ambos os eixos se encontram em escala logarítmica. A reta representa o ajuste linear realizado sobre as medidas, a inclinação da reta é $4D = 0,5231 \pm 6 \times 10^{-4}$.

	Tempo (segundos)			
Lado	Série	2 *threads*	4 *threads*	8 *threads*
128	$23,5 \pm 0,5$	$28,0 \pm 0,5$	$16,5 \pm 0,5$	$10,0 \pm 0,5$
256	109 ± 1	113 ± 1	68 ± 1	37 ± 1
512	568 ± 2	462 ± 2	278 ± 2	165 ± 2

TABELA 4.1. Resultados para tempo de execução de simulações ($\rho = 0,4$) com 10^5 *MCS* para diferentes tamanhos de rede. O algoritmo em paralelo utilizado para comparação foi a versão multidimensional com alteração nos pesos das transições. Nota-se novamente que em redes menores a melhora na eficiência não é tão explícita.

Os resultados referentes as simulações utilizando esta nova estratégia de paralelização estão dispostos na figura 4.7. Comparando com os resultados anteriores (figuras 4.2, 4.4 e 4.6) para coeficientes de difusão D, vemos que esta é a única estratégia em que os resultados obtidos na simulação são semelhantes aos obtidos em série. A partir destas observações, vemos que em casos onde a preservação da dinâmica difusiva seja relevante, esta seria a dinâmica em paralelo indicada. Analisando a tabela 4.1, nota-se que em redes maiores a redução no tempo de simulação, na comparação com a execução serial, chega a $\approx 70\%$ quando utilizamos 8 *threads*. Além disso, vemos que a simulação de uma rede de tamanho $L = 512$ pode ser realizada com 8 *threads* em paralelo em um período de tempo próximo a uma de lado $L = 256$ executada em série.

4.1.3 Paralelização: *Double Tiling*

Nesta seção, abordaremos uma nova estratégia de decomposição em domínios, chamada de *double tiling* (figura 2.1). A ideia principal por trás desta, assim como das outras decomposições espaciais, é encontrar independência entre determinadas regiões da rede, de modo que estas, possam sempre ser atualizadas simultaneamente. Na estratégia proposta, separando a rede em 4 classes distintas vemos que regiões tracejadas por mesma cor são sempre independentes. Isto implica que sítios em distintas regiões de mesma cor, podem ser sempre atualizados em paralelo sem concorrência por memória.

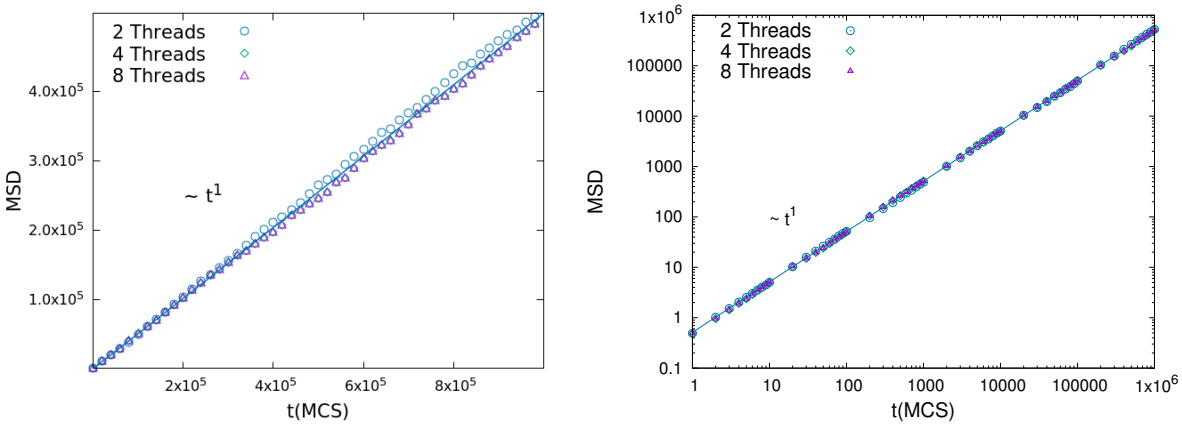

Figura 4.8. A figura mostra resultados obtidos para simulações ($\rho = 0,5$ e $L = 32$) executadas em diferentes ambientes, utilizando o algoritmo com decomposição em *double tiling*. A direita, ambos os eixos se encontram em escala logarítmica. A reta representa o ajuste linear realizado sobre as medidas, a inclinação da reta é $4D = 0,5045 \pm 5 \times 10^{-4}$

Utilizando regiões de tamanho 4×4, vemos que $P = L^2/64$ é o número de regiões de mesma cor[3] em uma rede quadrada de lado L. Nesta estratégia, cada *MCS* será dividido em 4 laços sobre regiões de mesma cor. Dentro de um laço, cada uma das distintas regiões será entregue a um *thread*, de modo que P é também o número máximo de *threads* com o qual podemos trabalhar. Dentro de cada região, a atualização será feita de forma aleatória, somente sobre os sítios ocupados, com N' tentativas de movimento por região, onde N' aqui é o número de partículas dentro da região. De modo que ao final de um *MCS*, terão sido realizadas N tentativas de movimento. Logo, como as regiões são independentes, e dentro de cada região somente um *thread* trabalha, garantimos que não

[3] Esta estratégia só é válida para redes com lado múltiplo de 8, $L = 8,\ 16,\ 32,\ 64,\ \cdots$. Por este motivo, nesta seção trabalharemos com redes como estas.

haverá *data racing*.

Vemos que em relação as figura 4.2 e 4.7, a figura 4.8 mostra resultados semelhantes. Este resultado nos motiva a investigar os tempos de execução deste código, que estão dispostos na tabela 4.2.

	Tempo (segundos)				
Lado	Série	2 *threads*	4 *threads*	8 *threads*	8 *th* (tab 4.1)
128	$23,5 \pm 0,5$	$19,5 \pm 0,5$	$11,5 \pm 0,5$	$7,0 \pm 0,5$	$10,0 \pm 0,5$
256	109 ± 1	71 ± 1	43 ± 1	27 ± 1	37 ± 1
512	568 ± 2	300 ± 2	180 ± 2	105 ± 2	165 ± 2

TABELA 4.2. A tabela mostra tempos de execução em diferentes ambientes computacionais. Os resultados foram obtidos simulando 10^5 *MCS* ($\rho = 0,4$ e $T = 0,4$) em cada uma das redes.

Em comparação aos resultados da tabela 4.1, a tabela 4.2 mostra uma redução de quase 40% no tempo de execução. Isto decorre do fato de na seção 4.1.2 realizarmos $2\times$ mais passos a cada *MCS*. Apesar desta redução no tempo de execução, optamos por não trabalhar com este, pela dificuldade em sua implementação quando passamos ao caso com interações. A figura 4.9 e a tabela 4.3 mostram a comparação final entre os resultados obtidos durante esta seção.

Método	D	Tempo de Execução (s)
Série	$0,1334 \pm 2,3 \times 10^{-3}$	568 ± 2
Checkerboard 4.1.1 (8 *Th.*)	$0,0750 \pm 3 \times 10^{-4}$	244 ± 2
Mult. Sem Repesagem 4.1.2 (8 *Th.*)	$0,1486 \pm 1 \times 10^{-4}$	200 ± 2
Mult. Com Repesagem 4.1.2 (8 *Th.*)	$0,1307 \pm 1 \times 10^{-4}$	165 ± 2
Double Tiling 4.1.3 (8 *Th.*)	$0,1261 \pm 1 \times 10^{-4}$	105 ± 2

TABELA 4.3. A tabela sintetiza os resultados relevantes obtidos nesta seção. Os coeficientes de inclinação D foram obtidos através do ajuste linear das medidas para cada execução. Os tempos de execução aqui são referentes à execuções com $\rho = 0,4$, $L = 512$ e simulada durante $10^5 MCS$.

Mostramos então, que a estratégia de paralelização utilizada, apesar de satisfazer as condições estatísticas do modelo, pode alterar o comportamento das medidas de *MSD*

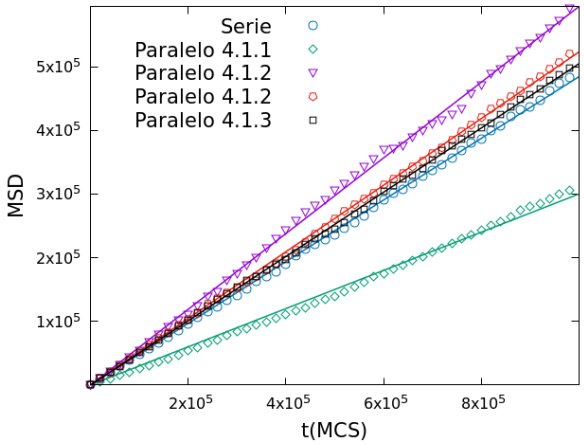

Figura 4.9. Resultados obtidos durante esta seção para *MSD* em função do tempo. As séries temporais foram as mesmas utilizadas durante seção ($L = 32$ e $\rho = 0,5$), e portanto os coeficientes de difusão são os mesmos obtidos nas figuras anteriores. Como discutido anteriormente, nota-se uma grande diferença no comportamento difusivo do sistema, principalmente, nos códigos desenvolvidos nas seções 4.1.1 e 4.1.2.

de um sistema. Alguém interessado em evitar o impacto da difusão em dado sistema, pode optar por utilizar o algoritmo desenvolvido na seção 4.1.1. Enquanto que alguém interessado em um sistema com uma dinâmica menos difusiva pode optar por utilizar o algoritmo desenvolvido na primeira parte da seção 4.1.2. Estes resultados devem ser verificados antes da aplicação.

4.2 Caso $\epsilon > 0$

Quando passamos a permitir interações (atrativas) entre as partículas do sistema, de acordo com a equação 4.1, este passa a se tornar um modelo para gases reais[4]. Através da execução em série discutiremos os resultados desejados pela paralelização desta. Nas medidas que prosseguem, tanto a constante de Boltzmann quanto a constante de interação foram tomadas como unitárias ($\epsilon = 1, k_B = 1$).

A figura 4.10 mostra retratos do sistema em diferentes pontos de sua evolução. Podemos notar a formação de regiões de mais alta densidade, que consideramos como áreas onde o sistema já condensou e se encontra na fase líquida. Nestes regimes, analisaremos

[4]No caso $\epsilon = 0$ o sistema possui solução exata e não apresenta a transição de fase gás-líquido.

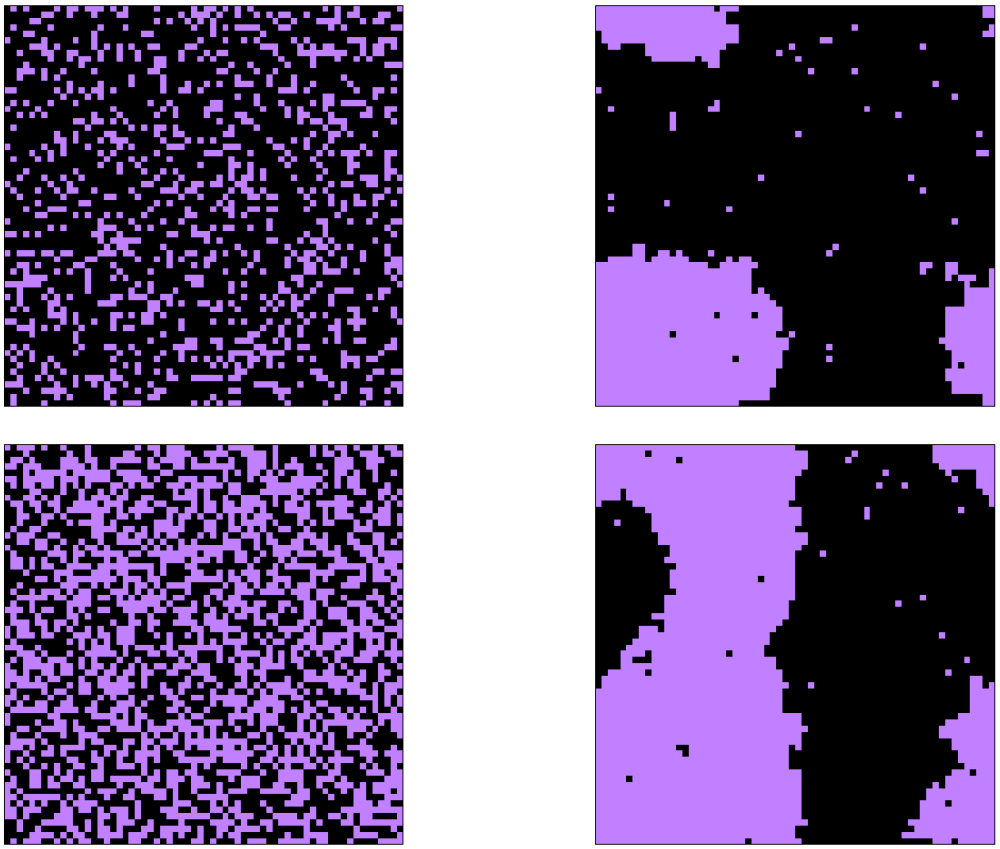

FIGURA 4.10. Imagem dos instantes inicial ($t = 0$) e de equilíbrio ($t = 5 \times 10^5 MCS$), respectivamente, para rede quadrada de lado $L = 64$, executada em série, com $\rho =$ 0,25(cima) e $0,5$(baixo), à temperatura de banho térmico $T = 0{,}4$. Neste estado, como esperado do modelo, percebe-se coexistência entre as fases líquida e gasosa do sistema.

apenas a distribuição de energias geradas pelas simulações.

Na figura 4.11, podemos ver a série temporal obtida nas simulação do modelo em série, e a distribuição estatística desta. Como esperado, a distribuição gerada, é a de Boltzmann. A figura 4.12 exemplifica o porque de não medirmos difusão nestes regimes. Para tempos mais longos, os movimentos coletivos (de *clusters*, ou gotas[5]) passa a ser mais relevante que os movimentos individuais, que só são realizados por partículas na superfície destas formações.

A partir de agora, trabalharemos com o modelo em regimes de alta temperatura ($T > T_C$), onde esperamos encontrar comportamentos difusivos nas medidas de *MSD*.

[5] Conjuntos de partículas que por interação, tendem a ficar juntas apesar do movimento aleatório individual.

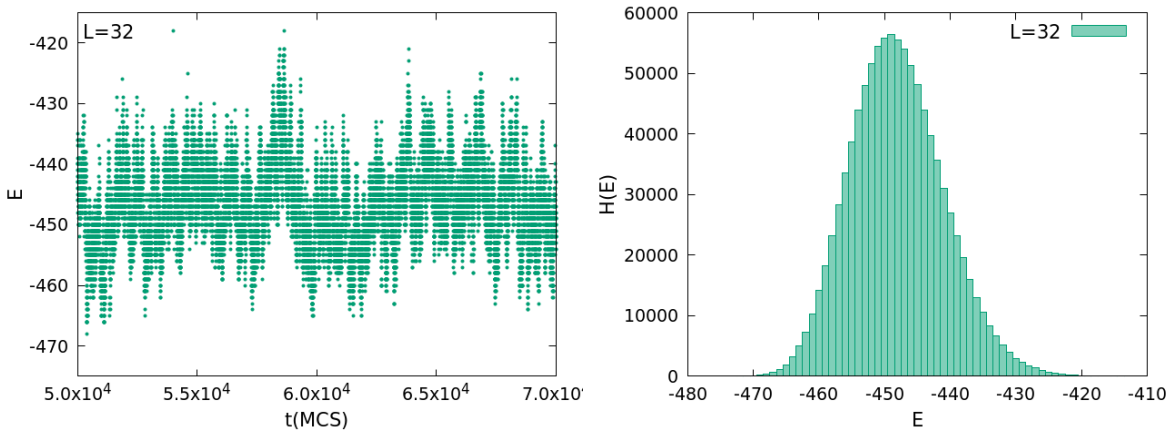

Figura 4.11. Série temporal para energia ($\rho = 0,25$ e $T = 0,4$), e seu respectivo histograma. Foram realizados $2,5 \times 10^5 MCS$ até o equilíbrio. Vale apontar que a semelhança nas distribuições entre este, e o modelo de Ising é consequência do algoritmo de Metropolis.

Nestes regimes, temperatura de banho térmico deve ser muito alta em relação à força de interação entre as partículas, favorecendo o movimento aleatório e não a formação de gotas.

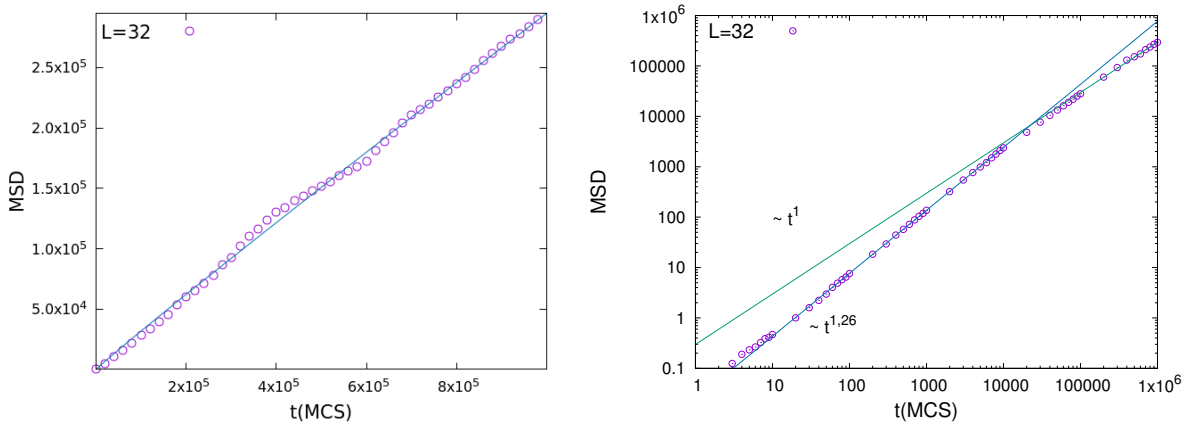

Figura 4.12. Deslocamento quadrático médio em função do tempo para as simulações da figura 4.11. As retas, ajustadas por diferentes pontos da curva, mostram que o comportamento desta grandeza não é linear.

Assim como no caso sem interação, as medidas de *MSD* não dependem do tamanho da rede (devido às *PBC*), mas sim da densidade ρ e da temperatura T. Portanto, na comparação com os códigos que serão desenvolvidos, utilizaremos sempre a rede de lado $L = 32$. A figura 4.13 mostra que nos regimes trabalhados o comportamento difusivo

é recuperado apesar das interações. Nota-se também a similaridade com o caso sem interações estudado na seção 4.1. No limite para $T >> T_c$ e baixas densidades, este comportamento deve ser idêntico ao previamente citado.

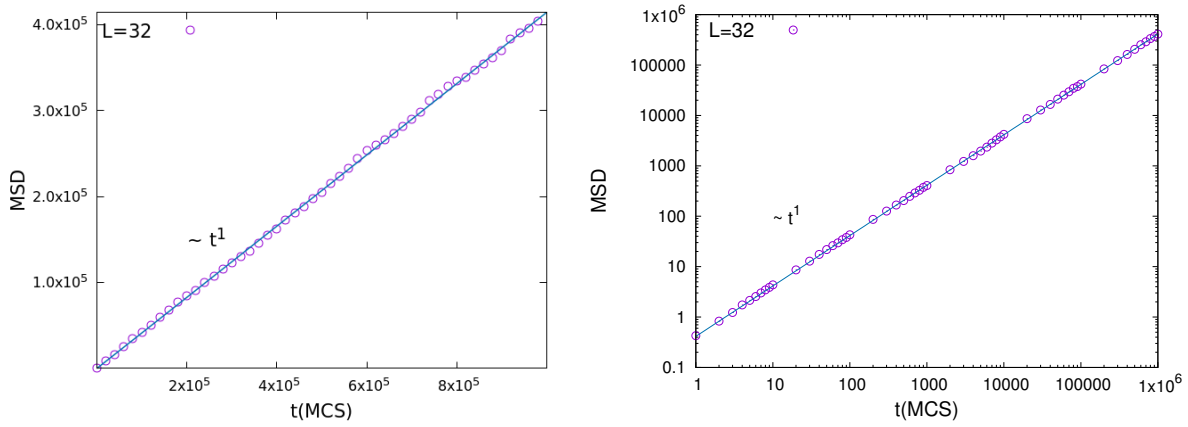

FIGURA 4.13. Comportamento do *MSD* com densidade $\rho = 0{,}5$ e temperatura de banho térmico $T = 1$. Nestas condições, notamos que o sistema apresenta comportamento linear com relação ao *MSD*, como esperado para esta temperatura e densidade.

A partir de agora, para corroborar os resultados obtidos sem interação, passaremos à aplicação das estratégias desenvolvidas ao gás de rede com interação.

4.2.1 Paralelização: Dinâmica Multidimensional

Utilizando o código desenvolvido na seção 4.1.2 e aplicação direta da equação 4.2 ao algoritmo de Metropolis (eq. 2.3), adaptamos a estratégia ao modelo com interações. Primeiramente, analisamos a distribuição de energias gerada pelos códigos desenvolvidos nos regimes de baixa densidade e temperatura.

Os resultados obtidos para estas execuções, em comparação com a simulação em série, estão dispostos na figura 4.14. Podemos notar que as medidas de energia para ambas as execuções são semelhantes, enfatizando a eficiência do código para a simulação. A figura 4.15 mostra que em regimes de alta temperatura, a estratégia de paralelização da seção 4.1.2 continua reproduzindo um comportamento difusivo idêntico à execução em série, o que pode vir a ser de interesse.

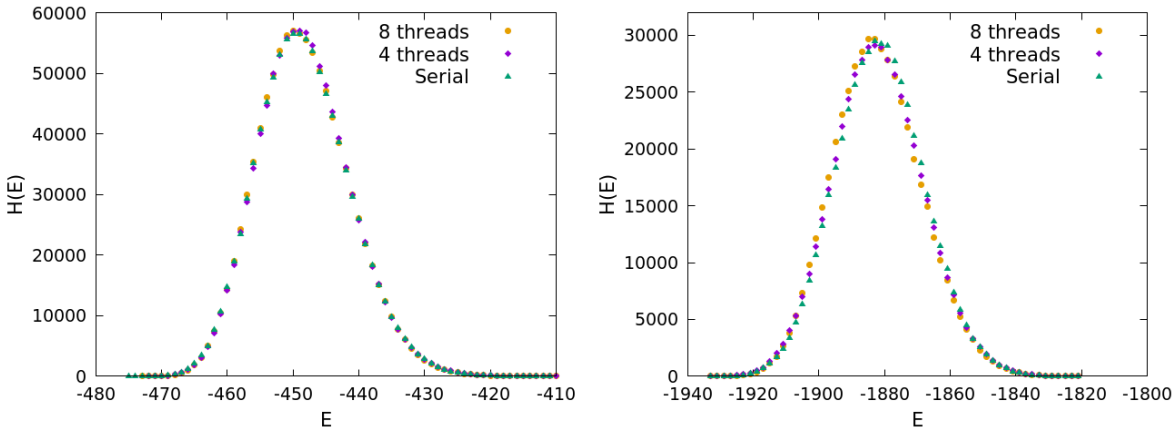

Figura 4.14. As figuras mostram histogramas para energia em redes de lado $L = 32$ e $L = 64$, respectivamente, gerados pelas execuções em paralelo usando 4 e 8 *threads* comparados com o algoritmo em série. A densidade de partículas utilizada foi $\rho = 0,25$

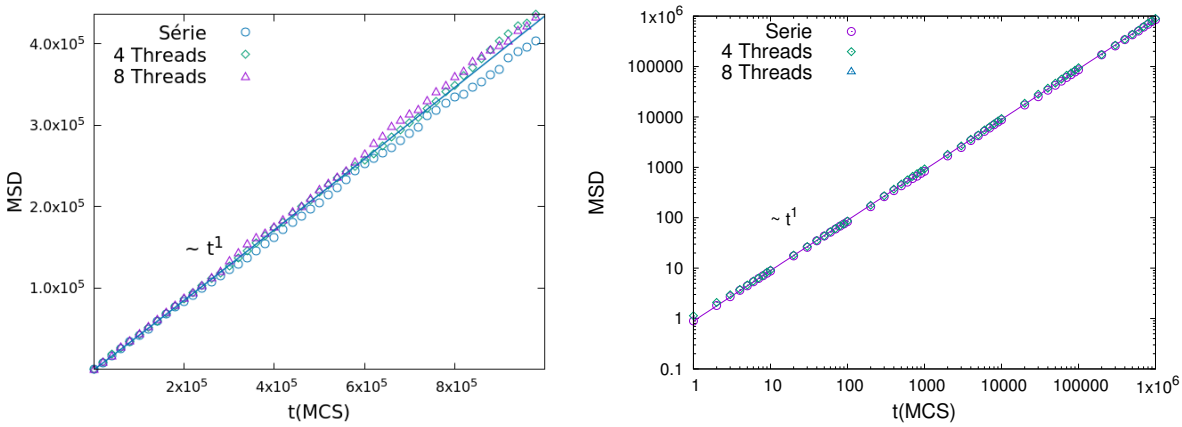

Figura 4.15. Comportamento do *MSD* com densidade $\rho = 0,5$ e temperatura de banho térmico $T = 1$. Nestas condições, notamos que o sistema apresenta comportamento linear com relação ao *MSD*, como esperado para esta temperatura e densidade.

Com relação aos tempos de execução, a tabela 4.4 mostra uma redução explícita de até $\approx 70\%$ em relação à execução em série. Assim como nos casos anteriores esta melhora na eficiência depende do tamanho da rede e do ambiente computacional.

Para garantir não só a eficiência computacional, como estatística do algoritmo, iremos novamente analisar os tempos de correlação entre as medidas de energia obtidas nos distintos ambientes computacionais, assim garantindo a eficiência do código desenvolvido.

	Tempo (segundos)			
Lado	Série	2 *threads*	4 *threads*	8 *threads*
128	$20 \pm 0,5$	$18 \pm 0,5$	$10 \pm 0,5$	$5,5 \pm 0,5$
256	100 ± 1	71 ± 1	40 ± 1	24 ± 1
512	510 ± 2	320 ± 2	178 ± 2	106 ± 2

TABELA 4.4. A tabela mostra tempos de execução em diferentes ambientes computacionais. Os resultados foram obtidos simulando 10^5 *MCS* ($\rho = 0,25$ e $T = 0,4$) em cada uma das redes.

4.2.2 Tempos de Autocorrelação

Nesta seção tomaremos procedimento análogo ao anterior (seção 3.4). Primeiramente analisaremos as funções de autocorrelação $C_E(t)$ e autocorrelação integrada τ_{int}, e depois estimaremos um parâmetro de eficiência (2.7) baseado no tempo de autocorrelação entre medidas de uma simulação.

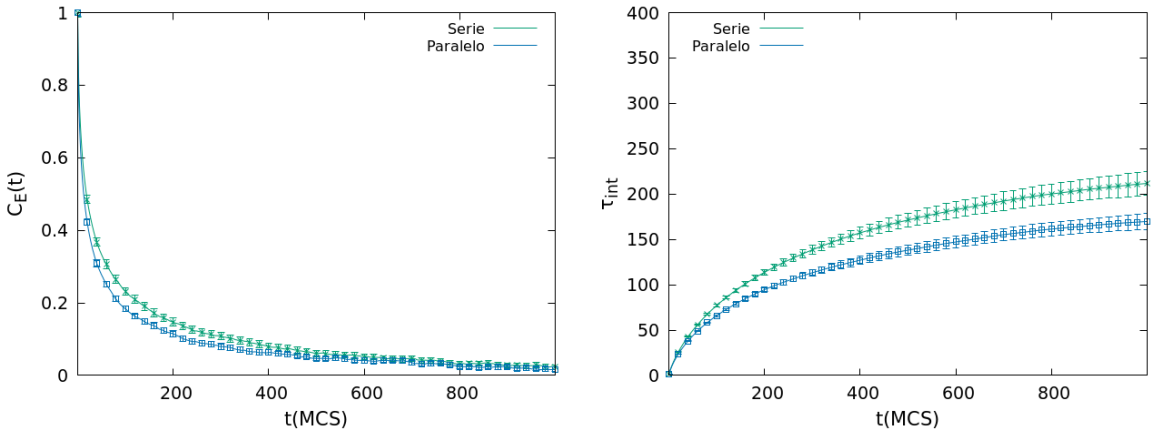

FIGURA 4.16. Função de autocorrelação e autocorrelação integrada, em função do tempo para as séries ($L = 32$, $\rho = 0,25$ $T = 0,8$) de energia obtidas na seção 4.2, em série, e na seção 4.2.1 em paralelo. Podemos notar que as medidas se descorrelacionam ligeiramente mais rápido no algoritmo executado em paralelo.

A figura 4.16 mostra as funções de autocorrelação e autocorrelação integrada para as diferentes execuções. Notamos que em ambas as simulações o tempo que a correlação entre medidas leva para cair a zero é aproximadamente o mesmo. À direita, podemos ver que enquanto o valor de convergência de τ_{int} em série é $t = \tau_c \approx 200MCS$, em paralelo este

valor é $t = \tau_c \approx 150 MCS$. Isto nos diz que além de computacionalmente mais eficiente, a estratégia desenvolvida na seção 4.1.2 é também estatisticamente mais eficiente.

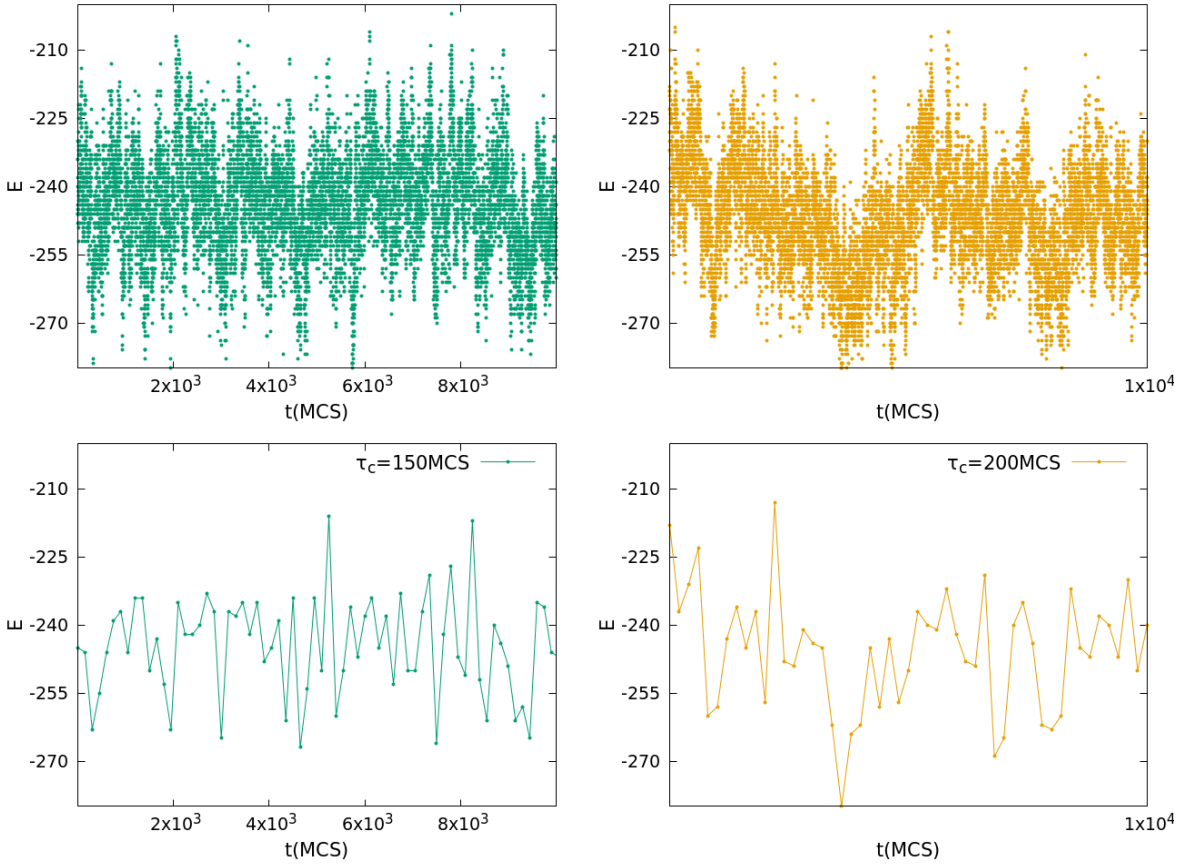

Figura 4.17. Séries temporais com e sem a utilização de um intervalo de tempo $\Delta t = \tau_c$ entre as medidas. Nota-se que as flutuações relevantes do ponto de vista estatístico parecem muito semelhantes nos dois casos, corroborando o resultado de que ambas as execuções possuem mesmo tempo de autocorrelação.

As séries temporais dispostas na figura 4.17 foram geradas pelos algoritmos supracitados. Tanto as figuras de cima, quanto as figuras de baixo geram a mesma distribuição estatística, entretanto, nas de baixo, o número de medidas realizadas é $\approx 200\times$ menor tornando a eficiência estatística das simulações maior. O fato de o tempo de correlação da execução em paralelo ser menor, juntamente com o com o desempenho computacional mostrado nas tabelas desta seção, indicam que o programa desenvolvido na seção 4.1.2 é de fato mais eficiente que o código executado em série.

Capítulo 5

Conclusões e Perspectivas

5.1 Conclusões

Primeiramente foram desenvolvidos conhecimentos essenciais em relação a algoritmos executados em paralelo e suas linguagens. Por meio destes, reproduzimos os resultados desejados para o modelo de Ising, em paralelo, com uma redução explícita no tempo de execução de até 90% (segundo a tabela 3.5). Corroboramos, também, o resultado conhecido de que algoritmos com seleção pré-ordenada de sítios possuem um tempo de correlação menor, entre medidas de um mesmo observável, tornando o algoritmo desenvolvido na seção 3.2 o mais eficiente, do ponto de vista estatístico e computacional, na aplicação ao modelo.

Em um segundo momento, as técnicas de paralelização desenvolvidas foram aplicadas, com sucesso, ao gás de rede sem interações, obtendo diferentes coeficientes de difusão para distintas estratégias de paralelização. Também foi proposta uma nova estratégia de paralelização, similar a utilizada em *fiber algotithms*[27, 28] e *lifting algorithms*[28, 29][1]. Na aplicação da estratégia proposta ao gás de rede com interações, os resultados conhecidos foram reproduzidos, em paralelo, com uma redução explícita no tempo de execução de até 70% (segundo a tabela 4.4). Concluímos que as estratégias elaboradas apresentam comportamento difusivo, alterando, em alguns casos, o coeficiente de difusão esperado para a execução em série. Este resultado permite a utilização destas estratégias em sistemas grandes sem o comprometimento deste fenômeno. Com relação aos tempos de

[1]Como o interesse do trabalho era que as distintas estratégias resultassem em movimento difusivo, não foi utilizada a ideia de persistência dos movimentos nas diferentes direções que estão presentes nestas referências.

correlação, notamos que ambas as execuções possuem tempos de correlação compatíveis, tornando o código desenvolvido na seção 4.1.2 o mais eficiente do ponto de vista computacional.

5.2 Perspectivas

A versatilidade dos algoritmos implementados neste trabalho permite a aplicação destas estratégias de paralelização a sistemas massivamente paralelos, através do uso de *GPU's.* e/ou a processadores mais modernos como, por exemplo, o *Threadripper 3* que possui 64 núcleos físicos e um total de 128 *threads*.

Casos onde as interações entre as partículas do Gás de Rede sejam repulsivas, como as da Ref. [30] podem ser interessantes, assim como casos onde as partículas ocupem regiões extensas na rede [30, 31]. Estes sistemas possivelmente podem ser tratados por meio de adaptações simples da decomposição de domínios.

Outra possibilidade seria uma abordagem teórica a partir da ideia do algoritmo com independência dos movimentos entre os graus de liberdade. Em um primeiro momento, escrever uma Equação Mestra para a dinâmica da distribuição de probabilidades dos estados do sistema, esta seria implementada e resolvida numericamente, nos moldes da Ref. [32]. Por meio desta abordagem, pode ser possível calcular a distribuição estacionária e os autovalores. Deste modo, podemos comparar a distribuição obtida com a esperada (a distribuição de Boltzmann) para validação da abordagem.

Referências Bibliográficas

[1] M. E. J. Newman and G. T. Barkema, *Monte Carlo methods in statistical physics*. Oxford: Clarendon Press, 1999.

[2] D. Frenkel and B. Smit, *Understanding Molecular Simulation*. Orlando, FL, USA: Academic Press, Inc., 2nd ed., 2001.

[3] S. R. Salinas, "Einstein e a teoria do movimento browniano," *Revista Brasileira de Ensino de Física*, vol. 27, no. 2, pp. 263–269, 2005.

[4] S. Chapman, T. G. Cowling, and D. Burnett, *The mathematical theory of non-uniform gases: an account of the kinetic theory of viscosity, thermal conduction and diffusion in gases*. Cambridge university press, 1990.

[5] A. Einstein, *Investigations on the Theory of the Brownian Movement*. Courier Corporation, 1956.

[6] M. P. Allen and D. J. Tildesley, *Computer simulation of liquids*. Oxford university press, 2017.

[7] G. Moore, "Moore's law," *Electronics Magazine*, vol. 38, no. 8, p. 114, 1965.

[8] T. Mattson and L. Meadows, "A "hands-on" introduction to openmp," *Intel Corporation*, 2014. `https://www.openmp.org/wp-content/uploads/Intro_To_OpenMP_Mattson.pdf`. Acesso: 23/12/2019.

[9] L. Onsager, "Crystal statistics. i. a two-dimensional model with an order-disorder transition," *Physical Review*, vol. 65, no. 3-4, p. 117, 1944.

[10] W. Janke, "Statistical analysis of simulations: Data correlations and error estimation," *Quantum Simulations of Complex Many-Body Systems: From Theory to Algorithms*, vol. 10, pp. 423–445, 2002.

[11] D. W. Heermann and A. N. Burkitt, "Parallelization of the ising model and its performance evaluation," *Parallel Computing*, vol. 13, no. 3, pp. 345–357, 1990.

[12] G. Altekar, S. Dwarkadas, J. P. Huelsenbeck, and F. Ronquist, "Parallel metropolis coupled markov chain monte carlo for bayesian phylogenetic inference," *Bioinformatics*, vol. 20, no. 3, pp. 407–415, 2004.

[13] E. Alerstam, T. Svensson, and S. Andersson-Engels, "Parallel computing with graphics processing units for high-speed monte carlo simulation of photon migration," *Journal of biomedical optics*, vol. 13, no. 6, p. 060504, 2008.

[14] J. S. Rosenthal, "Parallel computing and monte carlo algorithms," *Far east journal of theoretical statistics*, vol. 4, no. 2, pp. 207–236, 2000.

[15] L. Boltzmann, *Lectures on gas theory*. Courier Corporation, 2012.

[16] I. Beichl and F. Sullivan, "The metropolis algorithm," *Computing in Science & Engineering*, vol. 2, no. 1, p. 65, 2000.

[17] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *The journal of chemical physics*, vol. 21, no. 6, pp. 1087–1092, 1953.

[18] M. Weigel, "Simulating spin models on gpu," *Computer Physics Communications*, vol. 182, no. 9, pp. 1833–1836, 2011.

[19] K. B. Daly, J. B. Benziger, P. G. Debenedetti, and A. Z. Panagiotopoulos, "Massively parallel chemical potential calculation on graphics processing units," *Computer Physics Communications*, vol. 183, no. 10, pp. 2054–2062, 2012.

[20] M. Perc, "High-performance parallel computing in the classroom using the public goods game as an example," *European Journal of Physics*, vol. 38, no. 4, p. 045801, 2017.

[21] X. Li, A. B. Cohen, T. E. Murphy, and R. Roy, "Scalable parallel physical random number generator based on a superluminescent led," *Optics letters*, vol. 36, no. 6, pp. 1020–1022, 2011.

[22] M. S. Guskova, L. Y. Barash, and L. N. Shchur, "Rngavxlib: Program library for random number generation, avx realization," *Computer Physics Communications*, vol. 200, pp. 402–405, 2016.

[23] T. Tomé and M. de Oliveira, *Stochastic Dynamics and Irreversibility*. Graduate Texts in Physics, Springer International Publishing, 2014.

[24] P. D. Beale, "Exact distribution of energies in the two-dimensional ising model," *Phys. Rev. Lett.*, vol. 76, pp. 78–81, Jan 1996.

[25] D. P. Landau and K. Binder, *A guide to Monte Carlo simulations in statistical physics*. Cambridge university press, 2014.

[26] B. A. Berg, *Markov chain Monte Carlo simulations and their statistical analysis: with web-based Fortran code*. World Scientific Publishing Company, 2004.

[27] R. M. Neal, "Improving asymptotic variance of mcmc estimators: Non-reversible chains are better," *arXiv preprint math/0407281*, 2004.

[28] P. Diaconis, S. Holmes, and R. M. Neal, "Analysis of a nonreversible markov chain sampler," *Annals of Applied Probability*, pp. 726–752, 2000.

[29] H. C. Fernandes and M. Weigel, "Non-reversible monte carlo simulations of spin models," *Computer Physics Communications*, vol. 182, no. 9, pp. 1856–1859, 2011.

[30] H. C. M. Fernandes, J. J. Arenzon, and Y. Levin, "Monte carlo simulations of two-dimensional hard core lattice gases," *The Journal of chemical physics*, vol. 126, no. 11, p. 114508, 2007.

[31] H. C. M. Fernandes, Y. Levin, and J. J. Arenzon, "Equation of state for hard-square lattice gases," *Physical Review E*, vol. 75, no. 5, p. 052101, 2007.

[32] R. Ren and G. Orkoulas, "Acceleration of markov chain monte carlo simulations through sequential updating," *The Journal of Chemical Physics*, vol. 124, no. 6, p. 064109, 2006.