Fernando Piccini Cercato

Um Algoritmo de Alto Desempenho para Evoluir o Modelo de Potts Celular

Fernando Piccini Cercato

Um Algoritmo de Alto Desempenho para Evoluir o Modelo de Potts Celular

Dissertação apresentada à Universidade do Vale do Rio dos Sinos (UNISINOS) como requisito parcial para obtenção do grau de Mestre em Computação Aplicada.

Orientador: Prof. Dr. José Carlos Merino Mombach

São Leopoldo

2005

Ficha catalográfica elaborada pela Biblioteca da
Universidade do Vale do Rio dos Sinos

C412a Cercato, Fernando Piccini
Um algoritmo de alto desempenho para evoluir o modelo de Potts celular / por Fernando Piccini Cercato. - 2005.
83 f. : il. ; 29cm.

Dissertação (mestrado) - Universidade do Vale do Rio dos Sinos, Programa de Pós-Graduação em Computação aplicada, 2005.
"Orientação: Prof. Dr. José Carlos Merino Mombach, Ciências Exatas e Tecnológicas".

1. Algoritmo - Número aleatório. 2. Modelo de Potts. I. Título.

CDU 004.421.5

Catalogação na Publicação:
Bibliotecária Eliete Mari Doncato Brasil - CRB 10/1184

UNIVERSIDADE DO VALE DO RIO DOS SINOS

PROGRAMA INTERDISCIPLINAR DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO APLICADA

A COMISSÃO EXAMINADORA, ABAIXO ASSINADA, APROVA A DISSERTAÇÃO

UM ALGORITMO DE ALTO DESEMPENHO PARA EVOLUIR O MODELO DE POTTS CELULAR

ELABORADA POR

FERNANDO PICCINI CERCATO

COMO REQUISITO PARCIAL PARA A OBTENÇÃO DO TÍTULO DE MESTRE EM COMPUTAÇÃO APLICADA

Comissão examinadora:

Prof. Dr. José Carlos Merino Mombach
Universidade do Vale do Rio dos Sinos

Prof. Dr. Ney Lemke
Universidade do Vale do Rio dos Sinos

Prof^{a}. Dr^{a}. Maria Emília Machado Telles Walter
Universidade de Brasília

São Leopoldo, 10 de janeiro de 2006

Esta dissertação é dedicada
à Camila, minha filha.

AGRADECIMENTOS

Agradeço aos meus pais e à minha esposa pelo apoio e compreensão em todos os momentos para a realização do mestrado. Ao José Carlos, meu orientador e um grande amigo que esteve sempre presente e disposto a ajudar. Ao Gerson Cavalheiro co-orientador deste trabalho, um amigo muito querido, agradeço a tua disponibilidade. Aos amigos Alex Sandro Garzão, Daniel de Costa Paiva, Daniela Saccol Peranconi e demais colegas de mestrado. A Rejane Weissheimer, secretária do PIPCA, pela atenção. Agradeço também a HP pela bolsa de estudos e incentivo na realização deste trabalho.

Tudo que restou
Dos sonhos de guerreiros —
Capim de verão
Matsuo Bansho

RESUMO

A simulação de sistemas celulares tem recebido grande interesse nos últimos anos. Em particular, o modelo de Potts celular é o mais utilizado na área dada a sua precisão em representar estes sistemas. Este modelo, na sua forma convencional, possui uma série de operações e cálculos que são executados de maneira pouco eficiente, o que impossibilita sua utilização em simulações grandes e que exigem considerável tempo e memória para sua conclusão. Com base nisso propomos um novo algoritmo de maior desempenho que permite obter resultados aproximados dos obtidos com o algoritmo Monte Carlo em tempo bem menor. Técnicas de execução concorrente e comunicação foram introduzidas no algoritmo através do uso de processos leves para execução em computadores com memória compartilhada e usando aglomerados de computadores, respectivamente, buscando reduzir o tempo de processamento e viabilizando a execução de simulações de grande porte. Os resultados obtidos de simulações de segregação celular e evolução de espumas mostram um ganho de velocidade de no mínimo 6 vezes no tempo total de simulação, quando comparado ao algoritmo baseado em um processo Monte Carlo padrão. A execução de simulações grandes em aglomerados de computadores, não realizáveis em um computador único, tornou possível a obtenção de ganhos de velocidade da ordem de 9 vezes.

Palavras Chaves: Agregados celulares; modelo de Potts celular; alto desempenho.

ABSTRACT

The simulation of cellular systems has received great interest in the last years. In particular, the cellular Potts model is widely used in the area given its precision in representing these systems. This model, in its standard form, takes a series of operations and calculations that are executed in an inefficient way, what disables its use in large scale simulations that demand considerable time and memory for conclusion. Based on that, we propose a new algorithm of higher performance that allows to obtain results close from those obtained with the Monte Carlo algorithm in much shorter time. Techniques of concurrent execution and communication have been introduced in the algorithm, through the use of light processes for execution in computers with shared memory and using clusters of computers, respectively, aiming to reduce the processing time and making possible the execution of large scale simulations. The results presented obtained from simulation of cellular segregation and foam evolution show a minimum speedup of 6 times in the total simulation time when compared to the algorithm based on a standard Monte Carlo process. The execution of large scale simulations in computer clusters, not feasible in a single computer, make possible to reach speedups of the order of 9 times.

Keywords: Cellular aggregates; celular Potts model; high performance.

LISTA DE FIGURAS

1 Segregação celular. 16

2 Espumas de sabão. 16

3 Rede cúbica 10 x 10 x 10. Nesta imagem ilustrativa o eixo Z, perpendicular à página, exibe apenas 4 dos 10 planos. 20

4 Os 26 vizinhos mais próximos de um rótulo. 23

5 Rótulos das bordas destacados em vermelho, os demais rótulos são internos. 24

6 Estrutura básica de um aglomerado de computadores. 28

7 Diagrama do funcionamento do protocolo UDP. 40

8 Funcionamento do protocolo TCP. 41

9 Separando a rede de dados em sub-regiões para processamento paralelo. . . 45

10 Detecção e conexão, com linhas, das coordenadas iniciais e finais de uma célula. 50

11 Interface do programa visualizador. 51

12 Imagens capturadas pelo programa visualizador. 52

13 Simulação de segregação celular. 52

14 Simulação de espumas de sabão. 53

15 Fluxograma do algoritmo de CA's. 56

16 Estado inicial do agregado celular composto por 43 células vermelho claro e 69 células azul escuro no seu estado inicial. 57

17 Estado final do agregado celular composto por 43 células vermelho claro e 69 células azul escuro no seu estado final. 58

18 Estados da simulação de espumas de sabão 2d. 59

19 Resultados das simulações de espumas 2d. 60

20 Resultados das simulações de espumas 3d. 61

21 Fluxograma do algoritmo de CA's paralelo. 64

22 Formas de segmentação. 66

23 Comunicação existente em um bloco gerado através de um corte paralelo a um eixo qualquer. 67

24 Comunicação existente em um bloco gerado através de dois ou mais cortes paralelos a um eixo qualquer. 67

25 Um bloco de dados e suas regiões fantasmas. 69
26 Modelos de comunicação. 70
27 Fluxograma do algoritmo de CA's distribuído. 71
28 Resultados das simulações de espumas 3d. 74

LISTA DE TABELAS

1 Parâmetros utilizados na simulação de segregação celular. 57

2 Média de tempo em segundos gastos para cada algoritmo na simulação de segregação celular, em computador mono-processado Intel Xeon 2.66 GHz. 58

3 Parâmetros utilizados na simulação de bolhas de sabão. 59

4 Média de tempo em segundos gastos para cada algoritmo na simulação de segregação celular, em um computador com 2 processadores Intel Xeon 2.66 GHz. 62

5 Média de tempo em segundos gastos para cada algoritmo na simulação de segregação celular, em um computador 4 processadores Intel Xeon 1.6 GHz. 63

6 Quantidade de dados descartados na execução de uma caminhada de aproximadamente 100.000.000 rótulos 63

7 Tempo de execução seqüencial. 72

8 Tempo de execução dos algoritmos Caminhantes Aleatórios e Monte Carlo em um aglomerado de computadores (1 *thread* por nó). 73

9 Tempo de execução do algoritmo CA distribuído em um aglomerado de computadores (2 *threads* por nó). 73

10 Tempo de execução do algoritmo de CA em um aglomerado de computadores (2 *threads* por nó) com uma rede de dados grande contendo 900^2x800 rótulos. 74

LISTA DE ABREVIATURAS

2D - *Two Dimensional*
3D - *Three Dimensional*
BSP - *Bulk Synchronous Parallel*
CA - Caminhante Aleatório
CAM - *Cell Adhesion Molecule*
DAH - *Differential Adhesion Hypothesis*
IP - *Internet Protocol*
MA - Mensagens Ativas
MC - Monte Carlo
MPI - *Message Passing Interface*
PCA - Passo de Caminhante Aleatório
PMC - Passo de Monte Carlo
PRAM - *Parallel Random-access Machine*
RPC - Chamanda Remota de Procedimento
SCI - *Scalable Coherent Interface*
SMP - *Symmetric Multiprocessor*
TCP - *Transmission Control Protocol*
UDP - *User Datagram Protocol*

SUMÁRIO

1 Introdução **15**

2 Introdução ao modelo de Potts **19**

2.1 Modelo de Potts 19

2.2 Hipótese de Adesão Diferenciada 21

2.3 Modelo de Potts celular 22

2.4 Desempenho do algoritmo 24

2.5 Considerações 25

3 Introdução aos conceitos de concorrência e processos leves **26**

3.1 Aglomerado de computadores 26

3.2 Programação concorrente 28

3.3 Modelos de programação concorrente 31

3.4 *Threads*: processos leves 32

3.4.1 Mecanismos de sincronização 34

3.4.2 Formas de implementar utilizando *threads* 35

3.5 Comunicação em aglomerados de computadores 36

3.5.1 Troca de mensagens 37

3.5.2 Chamada Remota de Procedimento 38

3.5.3 Mensagens Ativas 38

3.6 Bibliotecas para comunicação em aglomerados de computadores 39

3.6.1 Sockets 39

3.6.2 MPI 41

3.6.3 ANAHY 42

3.7 Considerações 43

4 Técnicas para otimizar o modelo de Potts celular **44**

4.1 Paralelização do algoritmo convencional 44

4.2 Algoritmo N-Fold Way 45
4.3 Algoritmo Masking 46
4.4 Considerações 47

5 Visualizador de Agregados 3d **49**
5.1 Algoritmo para visualizar agregados 49
5.2 Resultados 50
5.3 Considerações 53

6 Caminhantes Aleatórios **54**
6.1 Algoritmo de Caminhantes Aleatórios 55
6.1.1 Resultados 57
6.2 Algoritmo de Caminhantes Aleatórios concorrente 61
6.2.1 Paralelizando o algoritmo de CA's 61
6.2.1.1 Resultados 65
6.2.2 Distribuindo o algoritmo de CA's 65
6.2.2.1 Resultados 72
6.3 Discussão dos algoritmos de CA's 74

7 Considerações Finais **76**

Referências **79**

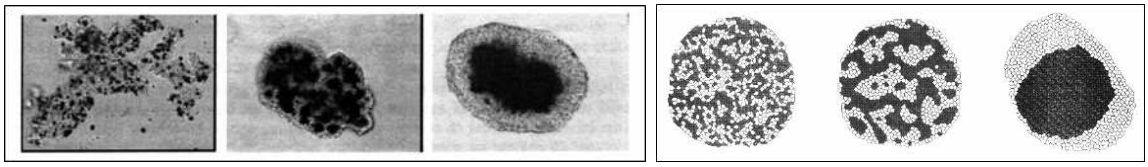
1 INTRODUÇÃO

A simulação é um conjunto de técnicas utilizadas em computadores para imitar, ou simular, operações de vários tipos de fenômenos ou processos do mundo real. O fenômeno ou processo de interesse é usualmente chamado de sistema, e para estudar cientificamente um sistema deve-se fazer algumas suposições de como este sistema funciona. Estas suposições tomam a forma de relações matemáticas ou lógicas relacionais que constituem um modelo que é usado para tentar ganhar algum conhecimento de como um sistema se comporta. Se a lógica relacional que compõe um modelo é simples, pode ser possível usar métodos matemáticos, tais como álgebra, cálculo, ou teoria das probabilidades para obter a informação exata a respeito das questões que se tem interesse. Isto é chamado de solução analítica. No entanto, os sistemas são muito complexos em geral para permitir que modelos realistas sejam avaliados pela forma analítica, e estes modelos devem ser estudados por meio de simulações. Numa simulação é usado um computador para evoluir um modelo numericamente, e os dados são agrupados de forma a juntar as características verdadeiramente desejadas de um modelo (LAW; KELTON, 1991).

Além disto, se um modelo de simulação não contém nenhum componente probabilístico, como um gerador de números aleatórios, é chamado de modelo determinístico. Nos sistemas determinísticos as saídas são "determinadas" uma vez que o conjunto de dados de entrada e os relacionamentos no modelo foram especificados. Muitos sistemas, no entanto, devem ser modelados tendo no mínimo um componente aleatório, e neste caso dá origem ao modelo de simulação estocástico. Os modelos estocásticos produzem uma saída de dados que por si só é aleatória e deve ser considerada apenas como uma estimativa das verdadeiras características do modelo. Esta é uma das principais desvantagens da

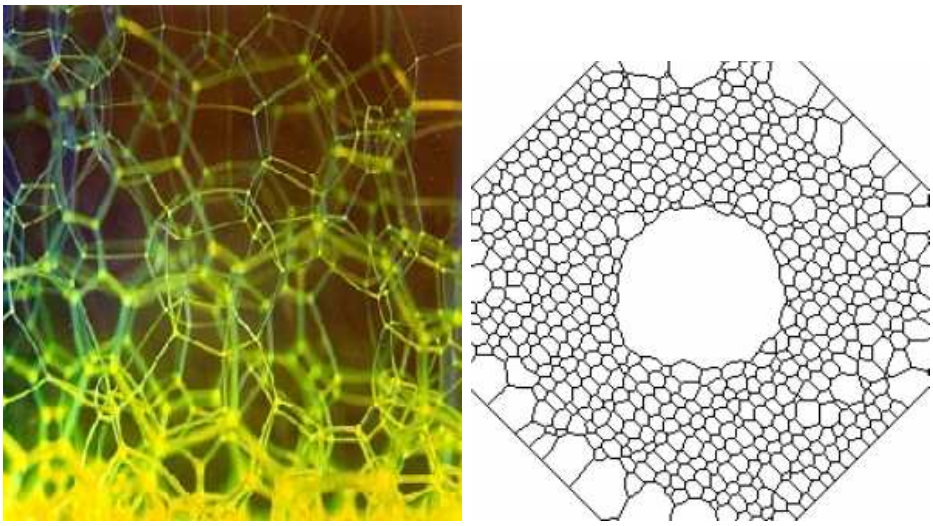
simulação (LAW; KELTON, 1991).

O modelo de Potts (GLAZIER; WEAIRE, 1992) é um modelo computacional estocástico similar ao modelo de Ising referenciado em (MOMBACH, 1997) utilizado para simular as propriedades magnéticas de materiais e grãos de um policristal (GLAZIER; WEAIRE, 1992). Uma evolução deste modelo é o modelo de Potts celular, que foi desenvolvido por Glazier e Graner em 1992 com a finalidade de simular a reorganização celular (Figura 1) (GRANER; GLAZIER, 1992). Atualmente este modelo vem sendo largamente utilizado para realizar diversos tipos de simulações, dentre eles o crescimento de tumores (KNEWITZ; MOMBACH, 2005; STOTT et al., 1999; KNEWITZ, 2002), desenvolvimento dos membros de aves (IZAGUIRRE; CHATURVEDI; HUANG, 2004), desenvolvimento do organismo completo *Dictyostelium discoideum* (MARÉE; HOGEWEG, 2001) e espumas de sabão (Figura 2) (GLAZIER; WEAIRE, 1992).



(a) Experimento real (b) Simulação 2D

Figura 1: Segregação celular.



(a) Experimento real (b) Simulação 2D

Figura 2: Espumas de sabão.

O algoritmo padrão utilizado para simulações do modelo de Potts apresenta uma série de cálculos de atualização de dados que são bastante ineficientes e têm alto custo computacional. Dependendo do tipo e do tamanho da simulação a ser realizada, a quantidade de memória exigida pelo algoritmo impossibilita a execução da simulação nas arquiteturas de computadores atuais. Propõe-se nesta dissertação o desenvolvimento de um algoritmo para execução do modelo de Potts buscando reduzir o tempo de processamento e possibilitar a execução de simulações grandes que possuem alto custo de memória utilizando conceitos de processamento paralelo e distribuído, empregando técnicas que minimizam o desperdício com cálculos desnecessários que não interferem na evolução de uma simulação qualquer. O uso de conceitos e técnicas de processamento paralelo e distribuído permitirá dividir uma simulação de grande porte em vários blocos, possibilitando sua execução em aglomerados de computadores. Para isto, serão investigados tópicos onde existe a possibilidade de execução concorrente no algoritmo de Potts celular, utilizando bibliotecas de comunicação em sistemas distribuídos, permitindo distribuir o mesmo código do algoritmo entre os nós que compõem um aglomerado de computadores fazendo com que dados sejam trabalhados em paralelo. Isto possibilitará uma divisão dos dados necessários à execução de uma simulação qualquer, permitindo desenvolver simulações de maiores proporções e com maior número de elementos, visando um maior realismo, uma vez que o uso de aglomerados de computadores tornou-se uma alternativa atraente para a solução de problemas que demandam grande quantidade de recursos computacionais por serem economicamente viáveis e de fácil manutenção. Para o desenvolvimento desta dissertação serão apresentados conceitos a respeito do modelo de Potts e da computação paralela, ítens necessários para se compreender este trabalho.

O objetivo deste trabalho é desenvolver um algoritmo eficiente para realizar simulações do modelo de Potts celular, possibilitando a execução de simulações de grandes proporções anteriormente limitadas pelo tempo e capacidade de memória. O algoritmo que simula o modelo de Potts celular será acrescido com a capacidade de execução concorrente, podendo explorar paralelismo em computadores com mais de um processador.

Por fim, este algoritmo será estendido fazendo uso da biblioteca de comunicação padrão Sockets que permite a comunicação necessária para a execução deste algoritmo em aglomerados de computadores, possibilitando segmentar a rede de dados utilizado na simulação, tornando possível a execução de simulação grandes que muitas vezes não podem ser executadas em apenas um computador devido a limitações de recursos de memória. Para facilitar a validação e permitir visualizar uma simulação qualquer, foi desenvolvido um programa visualizador que utiliza um algoritmo para construção de imagens.

Esta dissertação está organizada em sete capítulos. No Capítulo 2 é apresentado o modelo de Potts celular na sua forma convencional. No Capítulo 3 serão colocados os conceitos necessários para compreender e implementar concorrência. No Capítulo 4 serão demonstradas técnicas de otimização de código desta simulação, fundamento para desenvolvimento do novo algoritmo paralelo e distribuído. O Capítulo 5 apresenta um sistema que foi desenvolvido neste trabalho com a finalidade de permitir visualizar resultados de simulações do modelo de Potts celular. A seguir, no Capítulo 6, o algoritmo proposto neste trabalho é detalhado, apresentando a idéia por trás do novo algoritmo, que busca otimizar as simulações do modelo de Potts celular, possibilitando a sua execução de forma paralela e distribuída. Finalmente no Capítulo 7 são apresentadas algumas considerações finais a respeito dos resultados obtidos e os trabalhos futuros.

2 INTRODUÇÃO AO MODELO DE POTTS

Este capítulo apresenta o modelo de Potts celular, apresentando as principais finalidades deste modelo, destacando suas características de execução, desempenho e custo computacional.

2.1 Modelo de Potts

O modelo de Potts celular proposto por Glazier e Graner em 1992 (GRANER; GLAZIER, 1992) representa as propriedades físicas de células biológicas, sendo utilizado quando se pretende simular a reorganização celular que se dá pela minimização de energia livre de adesão celular, de acordo com a Hipótese da Adesão Diferenciada (IZAGUIRRE; CHATURVEDI; HUANG, 2004). Este modelo tem sido validado na simulação de sistemas em que a reorganização celular de um organismo completo se dá através da Adesão Celular Diferenciada (Seção 2.2). Neste modelo uma estrutura celular, como por exemplo um agregado de células, é representado em um espaço discreto. Neste caso uma célula é constituída por um grupo de rótulos espacialmente conexos e de mesmo valor numérico. Na Figura 3 é possível observar uma representação tridimensional esquemática baseada em uma rede cúbica de 10 elementos para cada eixo, totalizando 1000 rótulos (10^3). No segmento da rede que é possível ser observado na Figura 3, existem sete células distintas definidas pelo conjunto conexo de elementos da rede com mesmo rótulo σ. As células de rótulo 1, 2 e 6 são de um tipo celular τ e as demais de outro tipo. Se, por exemplo, consideramos que a célula $\sigma = 5$ possui a mesma forma em todos os 10 segmentos do eixo Z, então esta célula

possui volume V(σ) = 160 e superfície S(σ) = 200, que é a superfície dos elementos que compõem a borda da célula e estão em contato com as demais. Este sistema pode conter condições de contorno periódicas, não representada.

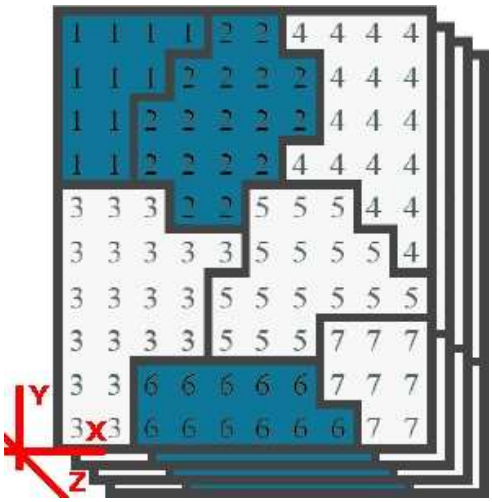


Figura 3: Rede cúbica 10 x 10 x 10. Nesta imagem ilustrativa o eixo Z, perpendicular à página, exibe apenas 4 dos 10 planos.

O modelo é evoluído por um processo Monte Carlo com o algoritmo de Metropolis. O método Monte Carlo faz parte da maior e mais importante classe de métodos numéricos para solução de problemas, e tem seu funcionamento a partir da simulação de um experimento com a finalidade de determinar propriedades probabilísticas de uma população a partir de uma nova amostragem aleatória dos componentes dessa população (NEWMAN; BARKEMA, 1999). Isto permite apresentar soluções aproximadas para uma grande variedade de problemas físicos e matemáticos por meio de experimentos numéricos que utilizam números aleatórios ou pseudo-aleatórios. Esta técnica pode ser aplicada tanto a problemas de natureza probabilística quanto a problemas determinísticos. O nome do método é inspirado na famosa cidade de Monte Carlo no principado de Mônaco e nas roletas dos cassinos, que são um exemplo de geradores de números aleatórios.

O algoritmo de Metropolis, introduzido por Nicolas Metropolis em 1953, é o mais utilizado em simulações Monte Carlo. Pode-se considerar o algoritmo de Metropolis um caso especial de amostragem de importância que gera estados com a probabilidade de

Boltzmann, responsável pela dinâmica da simulação, determinando as taxas de transição entre os níveis de energia em sistemas naturais. Este algoritmo trabalha gerando um novo estado ν, e então aceitando-o ou rejeitando-o de acordo com uma probabilidade de aceitação que depende da temperatura. À temperatura igual a zero, o sistema é determinístico. Se o novo estado é aceito, o sistema é alterado para este estado. Caso contrário, permanece inalterado. O algoritmo de Metropolis caracteriza-se por usar uma probabilidade de transição de um estado μ para ν, $P\ (\mu \rightarrow \nu)$, na seguinte forma:

$$P(\mu \rightarrow \nu) = \begin{cases} \exp(-\frac{\Delta E}{T})\ , & \text{if } \Delta E > 0\ , \\ \\ 1\ , & \text{if } \Delta E \leq 0\ , \end{cases} \tag{2.1}$$

onde o ΔE é a diferença de energia do sistema devido à mudança no estado e T é a temperatura. No caso do modelo de Potts celular, um parâmetro controla a mobilidade celular.

2.2 Hipótese de Adesão Diferenciada

Baseado em observações de que existe similaridade entre o comportamento de tecidos celulares e o de líquidos imiscíveis (por exemplo, água e óleo), foi proposta a Hipótese de Adesão Diferenciada (Differential Adhesion Hypothesis - DAH) (GRANER, 1993). Trata-se de um modelo termodinâmico, segundo o qual a interação entre as células envolve uma energia de adesão superficial que depende das moléculas de adesão nas membranas celulares. A minimização desta energia ou tensão superficial, analogamente ao que ocorre com líquidos, guia a evolução do sistema. Tensões superficiais representam diferenças de energia por unidade de área numa interface e determinam a configuração com o mínimo de energia global. Em agregados celulares, devido às Moléculas de Adesão Celular (Cell Adhesion Molecules - CAM's) (GRANER, 1993), diferentes energias estão associadas às interfaces entre células de tipos iguais e diferentes. Macroscopicamente estas energias se manifestam como tensões superficiais entre agregados de células e estas determinam se ti-

pos celulares diferentes se misturam ou se segregam. Tensões superficiais negativas levam à mistura de células de tipos diferentes e, de modo contrário, tensões superficiais positivas conduzem à segregação. Graner (1993) definiu, matematicamente, as tensões superficiais entre agregados celulares, onde uma mistura de dois tipos celulares hipotéticos, células claras e escuras, estão envolvidos. As energias de adesão superficial por unidade de área de contato, associadas com essas interfaces, tomam os valores e_{dd}, e_{dl}, e_{ll}, e_{dM} e e_{lM}, onde (l) refere-se a células claras, (d) a escuras e (M) ao meio extracelular. As tensões superficiais são definidas por (GRANER, 1993):

$$\gamma_{dl} \equiv e_{dl} - \frac{e_{dd} + e_{ll}}{2} \tag{2.2}$$

$$\gamma_{dM} \equiv e_{dM} - \frac{e_{dd}}{2} \tag{2.3}$$

$$\gamma_{lM} \equiv e_{lM} - \frac{e_{ll}}{2}. \tag{2.4}$$

2.3 Modelo de Potts celular

O modelo de Potts celular tem por finalidade simular a reorganização celular baseada na Hipótese da Adesão Celular Diferenciada (IZAGUIRRE; CHATURVEDI; HUANG, 2004). Ele é capaz de simular diversos fenômenos biológicos, dentre eles o crescimento de tumores. Neste modelo a dinâmica da reorganização celular se dá pela minimização da energia de adesão celular. Modelos baseados neste princípio têm obtido sucesso na simulação de sistemas em que até mesmo a reorganização celular de um organismo completo, como o *Dictyostelium discoideum* se dá através da Adesão Celular Diferenciada (MARÉE; HOGEWEG, 2001). As tensões superficiais no modelo são calculadas a partir da definição das energias de adesão superficial usando as equações (2.2), (2.3) e (2.4). A energia total de configuração do sistema é definida por (GRANER; GLAZIER, 1992; IZAGUIRRE; CHATURVEDI; HUANG, 2004):

$$E = \sum_{i,j,k} \sum_{i',j',k'} \epsilon_{\tau(\sigma),\tau(\sigma')}(1 - \delta_{\sigma,\sigma'}) + \lambda \sum_{\sigma} [V(\tau(\sigma)) - v(\sigma)]^2 \tag{2.5}$$

onde $\tau(\sigma)$ é o tipo associado com a célula σ e $\epsilon_{\tau(\sigma),\tau(\sigma')}$ é a energia de adesão superficial entre elementos do tipo σ e σ'. λ é um parâmetro associado à compressibilidade celular e $V(\tau(\sigma))$ é o "volume alvo" das células de tipo τ. O volume da célula σ é $v(\sigma)$. O primeiro somatório é executado sobre todos os rótulos da rede e o segundo sobre os vizinhos mais próximos de cada rótulo. (1-$\delta_{\sigma,\sigma'}$) identifica interfaces entre células distintas (δ=1 se σ=σ' e δ=0, caso contrário), fazendo com que sejam contabilizadas apenas as energias de adesão superficial de rótulos pertencentes às bordas (superfície) das células. O terceiro somatório é um termo de energia elástica que serve para regular o volume da célula e sua compressibilidade.

A simulação é evoluída através do algoritmo de Monte Carlo, onde para cada passo da simulação são realizadas tentativas de atualizações de rótulos até um número igual ao tamanho da rede, o que define a unidade de tempo, o Passo de Monte Carlo (PMC). As atualizações dos rótulos são feitas de acordo com algoritmo de Metropolis e envolvem a interação entre o rótulo selecionado e seus 26 vizinhos mais próximos (Figura 4) que são, respectivamente, os oito rótulos a volta do rótulo escolhido, seguido dos 9 rótulos pertencentes ao segmento imediatamente acima do rótulo escolhido e dos 9 rótulos pertencentes ao segmento imediatamente abaixo do rótulo escolhido.

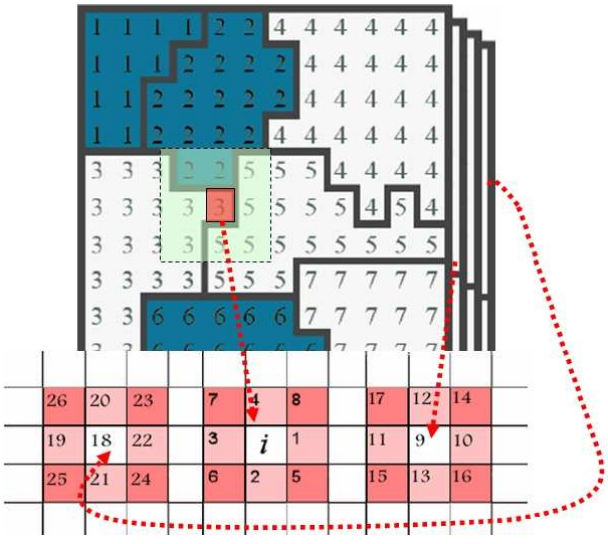


Figura 4: Os 26 vizinhos mais próximos de um rótulo.

2.4 Desempenho do algoritmo

O algoritmo que implementa o modelo de Potts celular, na sua forma original, é muito ineficiente computacionalmente (WRIGHT et al., 1997), porque, o uso do processo Monte Carlo para selecionar aleatoriamente os rótulos dos elementos que fazem parte da rede pode levar a muitas seleções que não permitem trocas que visam minimizar a energia do sistema, no caso de rótulos internos das células (Figura 5). O número de seleções aleatórias, n, do algoritmo é proporcional à dimensionalidade (d) do problema investigado $O(n^d)$, determinando que a complexidade de tempo deste problema seja polinomial. A maior ineficiência do algoritmo convencional se dá na tentativa de efetuar uma troca de rótulo numa região da rede onde todos os rótulos possuem o mesmo valor, ou seja, esta região pertence ao interior de uma célula. Esta tentativa de troca nesta configuração é muito comum e gera um desperdício de tempo em trocas que nunca poderão ocorrer.

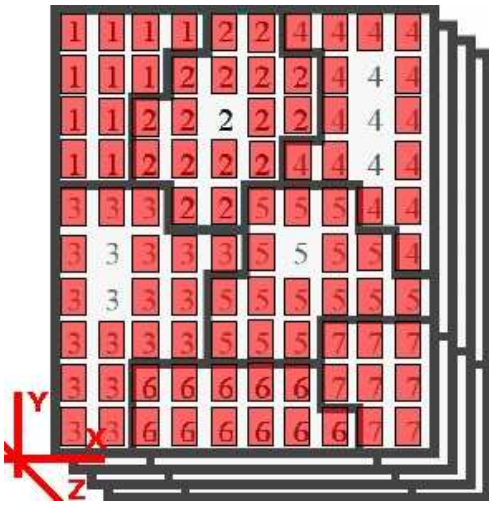


Figura 5: Rótulos das bordas destacados em vermelho, os demais rótulos são internos.

Com relação à memória, o processo convencional do algoritmo de Potts celular permite desenvolver simulações de pequeno porte nas arquiteturas de computadores atuais, porque a quantidade de memória necessária para armazenar cada rótulo pode variar de 2 a 8 bytes conforme a precisão exigida e, dependendo da simulação, o tamanho da rede cúbica pode exigir um grande espaço de armazenamento. O espaço onde é realizada a

simulação (rede de dados) depende diretamente da capacidade de memória do computador onde a simulação for executada, e exige um tempo de processamento com crescimento polinomial em relação ao tamanho da rede simulada. Por exemplo, pode-se imaginar uma rede cúbica contendo 1000^3 rótulos com um tamanho individual de 4 bytes cada um, item necessário para execução de simulações próximas da realidade obtida em experimentos em laboratório (WRIGHT et al., 1997; MOMBACH, 1997). Para este caso, efetuando os cálculos, a memória exigida para esta simulação é de 4 gigabytes.

2.5 Considerações

Neste capítulo foi apresentado o funcionamento do modelo de Potts celular, através do qual é possível a simulação de diversos tipos de fenômenos celulares. Este modelo apresenta alto custo computacional, uma vez que o método de Monte Carlo, neste caso, consome muito tempo ao realizar um grande número de sorteios em posições que não permitem trocas. O modelo de Potts trabalha com a dinâmica de Metropolis em redes de dados de grandes proporções que de alguma forma podem ser executadas em mais de um processador simultaneamente, porém devem ser levados em consideração os custos necessários para sincronizar os dados utilizados nos cálculos para efetuar a troca de rótulos.

3 INTRODUÇÃO AOS CONCEITOS DE CONCORRÊNCIA E PROCESSOS LEVES

Neste capítulo são apresentados conceitos de concorrência e técnicas de desenvolvimento de processos leves para a execução em computadores com memória compartilhada e em aglomerados de computadores. Estes assuntos são apresentados para que leitores menos familiarizados com esta área de computação possam compreender alguns dos conceitos envolvidos neste trabalho.

3.1 Aglomerado de computadores

Neste trabalho foi utilizado um aglomerado de computadores, solução de *hardware* muito empregada para resolver problemas grandes e complexos, tornando possível a utilização de uma arquitetura dotada de múltiplos recursos de processamento. Um aglomerado de computadores é constituído por um conjunto de computadores completos, os nós, com seus periféricos, que são interconectados fisicamente por uma rede de alto desempenho ou uma rede local (LAN) (TANENBAUM, 1999). Tipicamente, cada nó consiste em uma estação de trabalho, um servidor multiprocessado, ou um computador pessoal. O mais importante em um aglomerado de computadores é que todos os nós sejam capazes de trabalhar em conjunto, comportando-se como um único dispositivo integrado (HWANG; XU, 1998; BAKER R. BUYYA, 1999; VRENIOS, 2002). Este tipo de arquitetura é eficiente na obtenção de alto desempenho e disponibilidade de memória e, por serem construídos com peças comuns, encontradas com facilidade no mercado, a relação custo-benefício de um

aglomerado é mais baixa que a encontrada em supercomputadores. Na Figura 6 é possível observar a arquitetura conceitual de um aglomerado de computadores. Um aglomerado de computadores pode ser apresentado na forma de quatro conceitos básicos:

- **Nó:** Cada nó é um computador completo. Isto implica que cada nó tem seus processador(es), memória, discos de armazenamento e alguns adaptadores de E/S. Também reside no nó o sistema operacional.

- **Compartilhamento de recursos:** Um aglomerado de computadores possui um conjunto de ferramentas que permitem o agrupamento dos recursos de cada nó de forma a facilitar a operação deste tipo de arquitetura.

- **Comunicação entre-nós:** Cada nó de um aglomerado é conectado por algum tipo de interface que permita a comunicação entre os nós, tal interface pode ser, Ethernet, FDDI, fibra óptica e switches ATM. Além disto sempre se busca utilizar algum tipo de protocolo de comunicação padronizado, visando transparência na comunicação entre-nós.

- **Melhor desempenho:** Um aglomerado de computadores deve oferecer alto desempenho em vários aspectos. Uma destes aspectos consiste na atuação do aglomerado como um servidor de grande porte, onde cada n-nó do aglomerado pode servir m-clientes. Neste caso o aglomerado como um todo pode disponibilizar serviços para $m*n$ clientes simultaneamente. Um outro aspecto é usar o aglomerado para minimizar o tempo para executar uma única aplicação através do processamento paralelo distribuído. Este último é o aspecto usado neste trabalho.

Além disto os nós que compõem um aglomerado podem ser de diversas arquiteturas, dentre elas a arquitetura de multi-processador simétrico, ou, Symmetric Multiprocessing (SMP), uma arquitetura composta de vários processadores independentes que compartilham uma mesma memória. Neste caso cada computador pode ler ou escrever em qualquer parte da memória, surgindo a necessidade de que as atividades dos vários processadores

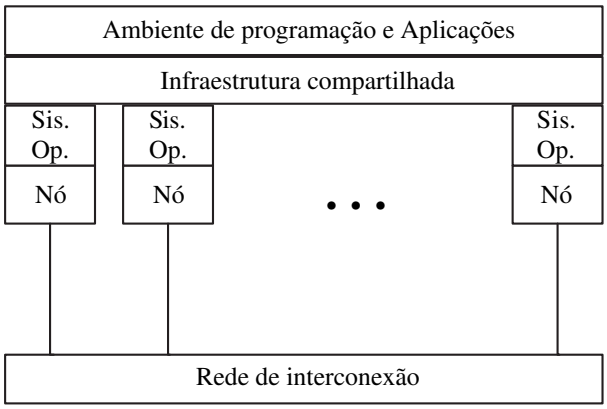


Figura 6: Estrutura básica de um aglomerado de computadores.

sejam coordenadas (por software), para evitar que a atividade de um processador interfira na atividade de outro (TANENBAUM, 1999). Esta arquitetura provê grande desempenho disponibilizando vários processadores para execução de processos individuais simultaneamente (multiprocessamento, Seção 3.4). Ao contrário do processamento assimétrico, qualquer processador que estiver ocioso pode ser utilizado para executar qualquer tarefa, e processadores adicionais podem ser adicionados para aumentar o desempenho relativo ao aumento de carga (HWANG; XU, 1998). Uma variedade de sistemas operacionais especializados e tipos de hardware estão disponíveis para execução SMP. Aplicações específicas podem se beneficiar desta arquitetura se o código permitir execução de threads (Seção 3.4).

3.2 Programação concorrente

Programação concorrente envolve a execução de dois ou mais fluxos de execução que realizam uma tarefa em comum (ANDREWS, 1991, 2000) e surgiu com a possibilidade de compartilhar recursos de computadores, tais como processador, memória, dispositivos de entrada/saída (E/S), o que permite a execução de dois ou mais programas independentes (TANENBAUM, 1992; SILBERSCHATZ; GALVIN, 1997). Com isto tornou-se possível explorar a concorrência das aplicações para buscar eficiência na exploração dos recursos computacionais. Para obter esta eficiência e ganhos de desempenho, os programas devem

utilizar o processador buscando utilizá-lo da melhor maneira possível. Quando se utilizam vários fluxos de execução para realizar uma tarefa em comum se obtém um melhor uso do processador, porque mesmo havendo a necessidade de utilizar recursos de dispositivos de comunicação (E/S) existem vários fluxos que estão disputando o uso do processador (VALIANT, 1990) visando eficiência. A eficiência é uma característica que busca explorar ao máximo o uso do processador e dos demais recursos disponíveis na arquitetura utilizada, buscando executar o maior número possível de processos em um certo intervalo de tempo (NAVAUX, 2001).

A programação concorrente é vantajosa em relação à computação seqüencial para o processamento de alto desempenho mas, devido ao fato de não existir um modelo universal para computação paralela, tal como o modelo para computação seqüencial de von Neumann, a tarefa de paralelizar programas se torna complicada (GOLDMAN, 2003). De acordo com o modelo de von Neumann, um computador consiste de um processador, uma unidade aritmética e uma área de memória. Neste modelo, as instruções são executadas seqüencialmente, na ordem especificada pelo programa fonte dentro de um fluxo de execução único, existindo um controle implícito que não permite que uma instrução execute antes que a anterior termine, garantindo que dados produzidos pela execução de instruções estejam disponíveis para leitura na memória. Na programação concorrente existem diversos fluxos de execução, cada um com sua própria seqüência de instruções, sem um controle implícito das relações de ordem de execução entre os diferentes fluxos, portanto sem mecanismos implícitos de controle a dados em memória.

Os programas concorrentes podem atuar de duas formas:

- **Visando competição:** Nesta forma os programas em execução são independentes e sem qualquer forma de colaboração entre si. Neste caso os fluxos de execução dividem o acesso aos recursos de E/S e competem pelo uso do processador, havendo a necessidade de empregar mecanismos para o controle de acesso aos recursos para evitar que um programa possa interferir na execução de outros.

- **Visando colaboração:** Neste caso uma única aplicação é decomposta em diversos fluxos de execução (WILKINSON; ALLEN, 1999), cada um responsável pela execução de uma parte de um problema maior, existindo a necessidade de troca de informações entre si para compor a solução final, surgindo uma pequena dependência. Fica a cargo do desenvolvedor determinar quais dados devem ser compartilhados usando mecanismos de sincronização (SEBESTA, 2000), quais atividades possuem restrições temporais e quais não possuem. As atividades que não possuem restrições podem concorrer por recursos de processamento.

Um programa concorrente, portanto, é composto por um conjunto de fluxos de execução que trocam informações de forma coordenada, garantida pelos mecanismos de sincronismo, que possibilita apresentar um resultado final da mesma maneira como seria feito na programação seqüencial tradicional (ANDREWS, 1991).

Dependendo da arquitetura utilizada podem ser apresentados vários níveis de concorrência. Para o caso de um aglomerado de computadores podem ser exploradas a concorrência intra e entre-nós. Estes casos são apresentados seguir:

- **Concorrência intra-nós:** É a concorrência que ocorre entre atividades executadas em um mesmo nó de um aglomerado ou em um único computador. Dependendo do número de fluxos de execução ativos em um dado momento e do número de processadores disponíveis no nó, este tipo de concorrência pode ser classificado como real ou temporal. A concorrência real ou paralelismo ocorre quando o número de fluxos de execução for menor ou igual ao número de processadores ativos em um determinado instante. A concorrência temporal ocorre quando existem mais fluxos de execução do que processadores disponíveis. Para este caso todos os fluxos de execução compartilham um mesmo espaço de endereçamento, e toda troca de dados entre processos pode ser feita através deste espaço comum.

- **Concorrência entre-nós:** Quando um aglomerado de computadores é utilizado, cada nó possui recursos próprios de memória e processamento, implicado que pode

ser executado em cada nó um fluxo de execução. Isto possibilita explorar paralelismo da arquitetura caso as tarefas a serem executadas sejam independentes e possam executar ao mesmo tempo. Neste caso se houver a necessidade de comunicação entre processos, é necessário utilizar algum tipo de mecanismo de comunicação, tal como a biblioteca Sockets (Seção 3.6.1).

Ambas as formas de concorrência podem ser exploradas ao usar um aglomerado de computadores. Desta forma os recursos de processamento podem ser utilizados de modo mais eficiente sobrepondo os gastos gerados pela comunicação por cálculo útil (VALIANT, 1990).

3.3 Modelos de programação concorrente

Esta seção apresenta alguns modelos utilizados para desenvolvimento de programas concorrentes. Fica a critério do programador definir o conjunto de recursos de programação que suporta a sua implementação. São apresentados a seguir os modelos clássicos de programação concorrente (CAVALHEIRO, 2004).

- **Compartilhamento de memória:** As tarefas em execução compartilham um mesmo espaço de memória. Toda comunicação entre tarefas pode ser feita através do acesso a esta memória. Pode-se efetuar comunicação utilizando variáveis compartilhadas.

- **Troca de mensagens:** Quando não existe um espaço de endereçamento comum, a troca de dados entre tarefas é realizada através da troca de mensagens de forma explícita. Neste caso uma rede de interconexão é explorada para o envio e recebimento de mensagens.

- **BSP:** Leslie Valient da universidade de Harvard propôs um modelo de programação concorrente, o bulk synchronous parallel (BSP), onde cada processo pode executar diferentes instruções simultaneamente. Pode-se compará-lo com o modelo parallel

random-access machine (PRAM), onde existem n processos, cada um residindo em um processador, e cada processador executa uma instrução deste processo em um ciclo (HWANG; XU, 1998). A característica principal do modelo BSP é a execução pelos fluxos de execução, de super-passo: cada super-passo consiste em uma fase de cálculo, na qual todos os processadores efetuam suas operações (as quais não têm a mesma duração), seguida de uma fase de comunicação, podendo esta também ser executada por todos os fluxos (VALIANT, 1990). Um super-passo termina com o sincronismo entre todos os fluxos.

- **SPMD:** Single Program Multiple Data (SPMD) é um tipo de arquitetura homogênea, onde uma mesma porção de código é executada por vários processos em diferentes dados. Esta arquitetura funciona como a arquitetura Multiple-Instructions-Multiple-Data (MIMD), onde diferentes instruções podem ser executadas por diferentes processos ao mesmo tempo (HWANG; XU, 1998).

Em aglomerados, é comum o uso de modelos mistos tais como: troca de mensagens e memória compartilhada.

3.4 *Threads*: processos leves

A programação de aplicações concorrentes é um recurso suportado pelos sistemas operacionais modernos, onde se faz uso de técnicas de multiprogramação leve (*multithreading*), ou seja, em um mesmo processo são criados vários fluxos de execução que são conhecidos como *threads* (ANDREWS, 1991). As *threads* são também denominadas processos leves, uma vez que os recursos de processamento contidos nos processos tais como contadores e controles de execução, normalmente onerosos no que diz respeito a sua manipulação pelo sistema operacional, são compartilhados por todas as *threads* criadas no processo (VAHALIA, 1976). Isto evita que cada *thread* necessite guardar sua própria descrição de recursos, tornando-a menos custosa ao sistema operacional (ANDREWS, 2000).

Um programa essencialmente *multithreading* é aquele que pode ser descrito como

efetuando diversas atividades diferentes ao mesmo tempo, que é o cenário ideal para o uso de *threads*. Programas típicos que são tipicamente *multithreading* incluem:

- **Atividades independentes:** Um exemplo é um depurador. Um depurador deve executar e monitorar um programa, mantendo a interface ativa, e deve mostrar e permitir a interação com o verificador de dados, com a pilha e com o monitor de desempenho. Como cada uma destas atividades deve ter acesso ao mesmo espaço de endereçamento é natural que se use *threads*.

- **Servidores:** Um exemplo é uma aplicação cliente/servidor. Tipicamente um servidor deve atender a vários pedidos (por exemplo: descarregamento de arquivos, acesso a bases de dados e outras operações com E/S intensivos). Usar um servidor com uma só *thread* significa que cada pedido tem que esperar que o anterior termine. Isto é lento e ineficiente. O uso de um servidor *multithreading* permite que vários pedidos sejam atendidos simultaneamente.

- **Tarefas repetitivas:** Um exemplo é um simulador que precisa apresentar as interações de diversos elementos diferentes a operarem simultaneamente. Diversas pequenas partes do problema são modeladas e depois os resultados são combinados para produzir o resultado final. Cada uma dessas pequenas partes é apropriada para ser escrita na sua *thread*.

Dentre os recursos compartilhados pelas *threads*, a memória alocada ao processo desempenha um papel especial: é através dela que as *threads* compartilham dados e se comunicam. O acesso à memória se dá pela simples execução de instruções de escrita e leitura. O problema é garantir o correto acesso às informações pelas *threads*, a solução emprega algum mecanismo de sincronização, abordado a seguir.

3.4.1 Mecanismos de sincronização

Existem certas condições em que um mesmo processo possui várias tarefas, que podem compartilhar recursos (memória, arquivo e etc). Dependendo do caso, duas ou mais tarefas podem tentar acessar estes recursos em um mesmo instante gerando erro nos cálculos (TANENBAUM, 1992). Para este tipo de problema estão à disposição do desenvolvedor mecanismos de sincronismo, recursos que servem para controlar a ordem de execução das tarefas (SEBESTA, 2000), determinando a partir de qual momento da execução uma tarefa pode passar a acessar os dados gerados por outra. Esta sincronização entre produção e consumo de dados entre as tarefas coordena a forma como elas comunicam-se (trocam dados). Com estes mecanismos torna-se possível exercer controle nos processos de forma a não permitir o acesso simultâneo dos dados, evitando erros nestes. Estes mecanismos são responsáveis por garantir que as tarefas seram executadas segundo a ordem definida pelo programa, durante a sua execução. Seguem abaixo as duas formas de implementar a sincronização nos programas concorrentes (ANDREWS, 1991, 2000).

- **Mutex:** O termo é originário do inglês *mutual exclusion* e designa um recurso que possibilita aos programadores coordenar o acesso a instruções em seções críticas compartilhadas por fluxos de execução concorrentes. O mutex trabalha com dois estados possíveis, aberto e fechado, empregando instruções que realizam a leitura e escrita de uma posição de memória de forma atômica, onde os dados são escritos e acessados em uma única operação. Com este recurso é feito o controle que possibilita a um processo a exclusividade de acesso a uma área de dados compartilhados, garantindo a execução correta de uma atividade concorrente.

- **Variável de condição:** É um mecanismo que garante que um processo irá aguardar, se necessário, até uma dada condição ser verdadeira. Um exemplo deste mecanismo é o semáforo, que consiste no uso de uma variável inteira (semáforo) para contar o número de sinais armazenados. De acordo com o valor desta variável, o semáforo pode realizar duas operações, *DOWN* e *UP* (generalizações de *SLEEP* e

WAKEUP, respectivamente (TANENBAUM, 1992)). A operação *DOWN* verifica se o valor do semáforo é maior que zero. Se for, seu valor é decrementado (ou seja, um sinal armazenado é gasto) e o processo simplesmente continua sua progressão. Se o valor do semáforo for zero, o processo que executou a operação *DOWN* é posto para dormir. A verificação do valor do semáforo, a modificação de seu valor, e eventualmente a colocação do processo para dormir, são ações únicas e indivisíveis. A operação *UP* incrementa o valor do semáforo. Se um ou mais processos estiverem dormindo neste semáforo, impedidos de completar uma operação *DOWN*, um deles vai ser escolhido pelo sistema, sendo-lhe então permitido completar a operação *DOWN*.

A sincronização no acesso à memória é necessária para evitar o não-determinismo na execução de programas concorrentes. Neste caso, a função da sincronização é controlar a execução de conjuntos de instruções que acessam uma área de dados compartilhados.

3.4.2 Formas de implementar utilizando *threads*

Pode-se caracterizar uma aplicação como paralela quando esta executa funções e utiliza recursos do sistema de forma concorrente a partir de várias *threads*. Dependendo da forma como forem implementadas, os programas podem ser caracterizados como:

- ***Thread-safe*:** Os procedimentos implementados no programa de forma a suportar reentrância, que determina o grau para o qual uma função de uma biblioteca ou rotina permite possuir várias instâncias de si mesma em progresso ao mesmo tempo (NICHOLS; BUTTLAR; FARREL, 1996). O comportamento reentrante não varia, independentemente de uma função ou rotina estar sendo executada em uma ou várias instâncias ao mesmo tempo. Para que a execução de várias chamadas simultâneas da função executem corretamente, a função não pode escrever em dados estáticos. Se fizer, cria uma condição de concorrência pelo acesso aos dados, e corre risco de obter resultados inválidos (NICHOLS; BUTTLAR; FARREL, 1996).

- ***Thread-aware*:** *Threads* onde são implementadas funções não-reentrantes, ou que utilizem dados globais, ou ainda que acessem recursos do sistema, a coerência pode ser obtida em alguns casos pelo emprego de mecanismos de sincronismo (NICHOLS; BUTTLAR; FARREL, 1996). O grande inconveniente desta solução é que uma operação bloqueante que estiver dentro da seção crítica poderá impedir por um tempo indeterminado a execução de outras *threads* e, dependendo do modelo de *threads*, bloquear uma *thread* significa bloquear todo o processo e por conseqüência todas as *threads* associadas ao processo, independentemente da tarefa que estas executam. Para este caso deve-se implementar os procedimentos de forma que estes não bloqueiem a execução das outras *threads*.

A implementação *thread-safe* permite a execução de várias tarefas de forma simultânea, uma vez que neste tipo de implementação não existem dependências de dados, o que faz com que esta técnica obtenha melhor desempenho.

3.5 Comunicação em aglomerados de computadores

Para funcionar corretamente em um aglomerado de computadores, uma aplicação deve ser capaz de realizar comunicação entre os nós que compõem o aglomerado, a fim de dividir o problema da aplicação em várias tarefas distribuídas entre os nós do aglomerado. No decorrer da execução da aplicação, dados produzidos por uma tarefa podem ser consumidos por outra tarefa que se encontra em um nó diferente, surgindo a necessidade de haver uma comunicação. A comunicação é uma das operações de maior custo neste tipo de arquitetura. Como existe a necessidade de processamento de alto desempenho, busca-se reduzir o tempo entre a produção de um dado e seu consumo (VALIANT, 1990).

Dessa forma, um bom desempenho da aplicação está diretamente ligado ao custo introduzido pela troca de dados entre os nós. Com base nessa situação, mecanismos de comunicação mais eficientes (MA e MPI, seções 3.5.3 e 3.6.2) e *hardware* para comunicação (SCI (ROSE et al., 2001) e MYRINET (BODEN et al., 1995)) com melhor desempenho têm

sido desenvolvidos, buscando explorar com maior eficiência os recursos oferecidos pelas redes de comunicação e reduzir o custo de comunicações.

O restante desta seção apresenta alguns métodos de comunicação que podem ser utilizadas para a troca de dados entre os nós de um aglomerado.

3.5.1 Troca de mensagens

Através das diretivas *send()* e *receive()*, um conjunto de dados pode ser enviado de um nó para outro, esta é uma estratégia tradicionalmente empregada para o compartilhamento de dados entre os nós de um aglomerado. Existem duas formas de utilizar esta estratégia:

- **Operação síncrona:** as diretivas *send()* e *receive()* nesta estratégia são operações bloqueantes, ou seja, uma chamada *send()* permanece bloqueada até que o *receive()* correspondente seja executado. Neste caso, o transmissor somente enviará os dados após receber do receptor uma confirmação de que este encontra-se pronto para a recepção. Claramente, utilizando-se a troca de mensagens síncrona, não é possível realizar processamento e comunicação ao mesmo tempo (EICKEN et al., 1992).

- **Operação assíncrona:** permite que parte do tempo gasto com comunicações seja sobreposto por cálculo efetivo. Isto é possível pois o transmissor de uma mensagem não necessita esperar nem pelo requerimento da mensagem por parte do receptor nem pela confirmação de recebimento desta mensagem. Com isso, após enviar uma mensagem, o transmissor está livre para novas transmissões ou para a realização de cálculo efetivo. O custo do uso desta técnica reflete-se em uma maior complexidade de controle da evolução do programa e da respectiva troca de dados.

Fazendo uso de operações assíncronas é possível obter melhor desempenho, neste caso dados podem ser processados no decorrer da transmissão de mensagens. Porém a implementação em software para este mecanismo é bastante complexa e custosa (CAVALHEIRO, 2001).

3.5.2 Chamada Remota de Procedimento

A chamada remota de procedimento (RPC) é uma estratégia para comunicação em aglomerados de computadores que oferece um nível de abstração ao programador (BIRRELL; NELSON, 1984). O funcionamento do RPC é semelhante a chamada de procedimento comum, onde a passagem de argumentos serve para enviar um conjunto de dados que permite executar de forma transparente ao desenvolvedor um serviço em um nó qualquer.

Esta técnica funciona com um servidor que disponibiliza serviços, um cliente que invoca os serviços oferecidos pelo servidor e um *stub*, executado junto ao cliente, que fica responsável pela captura das invocações do serviço remoto. A estratégia do RPC obteve popularidade devido à transparência que este serviço oferece ao programador, tornando a execução remota uma simples chamada de execução de procedimentos comum. E, além disto, este modelo opera de forma síncrona, onde o cliente espera sincronamente pelo servidor de forma semelhante à encontrada na execução de procedimentos seqüencial (BARCELLOS; GASPARY, 2003).

3.5.3 Mensagens Ativas

Permite realizar comunicação sem introduzir muito custo extra de execução na aplicação (WALLACH et al., 1995), apresentando-se como uma solução clássica para os problemas de comunicação em ambientes paralelos (LUMETTA; MAINWARING; CULLER, 1997; PERANCONI, 2005; PERANCONI; CAVALHEIRO, 2005). As mensagens ativas possuem um mecanismo de comunicação com mensagens assíncronas que buscam explorar toda capacidade das redes de computadores atuais (EICKEN et al., 1992). Quando uma mensagem é recebida, um procedimento é acionado para recuperar a mensagem da rede e inseri-la no processamento em andamento no nó, de forma que o serviço requisitado seja executado rapidamente. As mensagens deste serviço têm como funcionalidade enviar através do cabeçalho o endereço de um serviço a ser executado no receptor e o corpo da mensagem

armazena dados que compõem os argumentos do serviço (EICKEN et al., 1992).

Apesar das semelhanças entre Mensagens Ativas e RPC, cabe destacar que as primeiras foram desenvolvidas com o objetivo de obterem alto desempenho em arquiteturas distribuídas, enquanto as outras preocupam-se apenas com a comunicação em arquiteturas de aglomerados de computadores.

Quando suportadas em software (PERANCONI; CAVALHEIRO, 2005), Mensagens Ativas permitem sobrepor *overheads* de comunicação por cálculo efetivo.

3.6 Bibliotecas para comunicação em aglomerados de computadores

Desenvolver aplicações capazes de funcionar em arquiteturas de aglomerado de computadores consiste em uma tarefa bastante complexa, e envolve problemas que normalmente inexistem nas arquiteturas de computadores seqüenciais, tais como sincronia e comunicação de dados. Geralmente estes problemas são resolvidos com o uso de bibliotecas que visam facilitar o desenvolvimento de programas para este tipo de arquitetura, tornando a tarefa de implementação até certo ponto transparente para o desenvolvedor. Nesta seção é apresentada a biblioteca Sockets que é muito utilizado na implementação de rotinas voltadas para troca de mensagens e envio de dados na programação distribuída, e que também será utilizada neste trabalho. São abordadas também bibliotecas dotadas de recursos para facilitar a etapa de implementação e distribuição dos dados necessários para a execução do algoritmo de Caminhantes Aleatórios (Capítulo 6), uma vez que estas bibliotecas adotam técnicas de programação distribuída.

3.6.1 Sockets

Consiste num conjunto de sub-rotinas que permitem obter uma solução específica para uma plataforma. *Sockets* fornece uma abstração dos protocolos de comunicação em rede entre programas que necessitam realizar algum tipo de troca de mensagens ou dados,

além de oferecer portabilidade na troca de dados entre a maioria dos sistemas operacionais atuais (COMER; STEVENS, 2001).

Este conjunto de sub-rotinas trabalha sobre o protocolo IP de rede existindo a possibilidade de se optar pelo uso do protocolo TCP ou UDP. O seu uso fica a critério do desenvolvedor do programa, que deve observar as principais vantagens e desvantagens de cada um destes protocolos.

O protocolo UDP caracteriza-se por não ser orientado à conexão, isto é, não estabelece uma sessão direta entre cliente e servidor (Figura 7). Isto faz com que o servidor não guarde o estado dos pedidos, o que gera uma limitação no que diz respeito à capacidade de envio de pacotes de dados a apenas um por vez, o que também obriga a que a resposta do pedido seja realizada na mesma etapa.

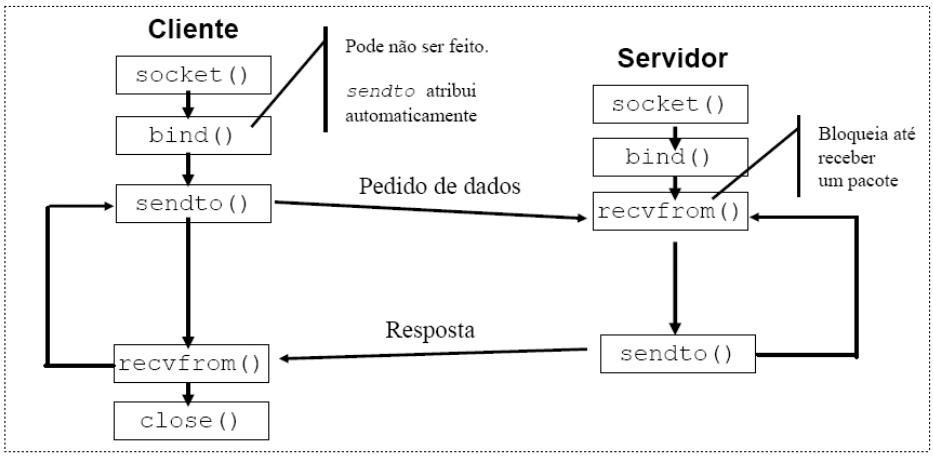


Figura 7: Diagrama do funcionamento do protocolo UDP.

Este protocolo exige por parte do programador a implementação de rotinas para detecção e correção de erros, tais como perda de pacotes de dados e recebimento de pacotes fora de ordem. Por este motivo o protocolo UDP é mais adequado para servidores onde poucos dados devem ser transportados.

Diferentemente do protocolo UDP, o protocolo TCP estabelece uma sessão entre cliente-servidor, fornecendo um canal de comunicação bidirecional dedicado, onde é possí-

vel enviar/receber dados de forma simultânea em ambos os lados (cliente-servidor). Além disso, este protocolo permite que sejam estabelecidas várias conexões simultaneamente, cada uma sendo controlada individualmente, pois neste protocolo cada cliente que consegue estabelecer uma conexão com o servidor recebe um *socket* independente do original utilizado para receber clientes.

Uma vez estabelecida a conexão cliente-servidor, passa a existir um canal dedicado para comunicação exclusiva do servidor para com o cliente e vice-versa, (Figura 8).

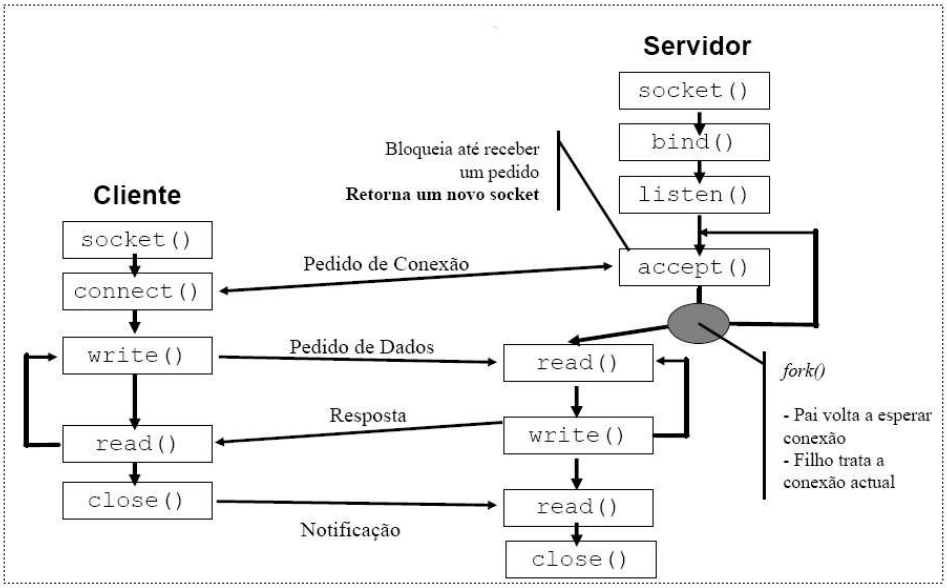


Figura 8: Funcionamento do protocolo TCP.

3.6.2 MPI

(Message Passing Interface - MPI) é uma biblioteca que faz uso do paradigma de troca de mensagens, que é largamente utilizado em certos tipos de máquinas paralelas, em especial nas arquiteturas que possuem memória distribuída. As maiores vantagens de se estabelecer troca de mensagens padronizadas são a portabilidade e a facilidade de uso. Em ambientes de memória distribuída que fazem uso de rotinas de alto nível e/ou rotinas de abstração de baixo nível, padronizadas para troca de mensagens, apresentam benefícios bastante aparentes (BENITEZ; CAVALHEIRO, 2004; COSTA; STRINGHINI; CAVALHEIRO,

2002).

O funcionamento desta biblioteca tem como base o uso de uma máquina virtual sobre uma arquitetura real. Cada nó virtual consiste de um processo executando sobre um nó real da arquitetura, sendo que mais de um nó virtual poderá residir no mesmo nó real e nós virtuais poderão estabelecer comunicação para envio e recebimento de dados quando necessário.

3.6.3 ANAHY

É uma biblioteca de programação e ambiente de execução concebido para explorar computação de alto desempenho em arquiteturas paralelas e aglomerados de computadores, onde cada nó pode ser multiprocessado com memória compartilhada (CAVALHEIRO; DALL'AGNOL; VILLA REAL, 2003; PERANCONI, 2005). Esta biblioteca oferece ao desenvolvedor uma interface de programação da aplicação (API), que tem como finalidade permitir descrever a concorrência existente na aplicação, além de fornecer também suporte para a execução do programa. Um dos pontos forte do ANAHY se deve ao fato da API desta biblioteca ser um subconjunto de comandos que são similares aos das *threads* padrão POSIX que possuem capacidade de minimizar as dificuldades de gerenciamento de um grande número de fluxos de execução concorrente e das comunicação entre eles (PERANCONI, 2005). Além disso sua arquitetura interna implementa um escalonador de estratégia baseado num escalonador de listas. O algoritmo do escalonador do ANAHY tem como característica garantir que as tarefas definidas pelo programador serão executadas respeitando as relações de dependência dos dados na aplicação (DALL'AGNOL et al., 2003; CAVALHEIRO; DALL'AGNOL; VILLA REAL, 2003). Outra característica importante desta biblioteca se deve ao fato de que em tempo de execução existe a possibilidade de ajustar o algoritmo escalonador de forma a buscar uma melhor estratégia de execução para a aplicação numa determinada arquitetura, considerados as estruturas de dependência de dados e os recursos de processador disponíveis (CORDEIRO et al., 2005).

3.7 Considerações

A execução concorrente possibilita explorar alto desempenho de forma eficiente, permitindo explorar ao máximo possível os recursos de processamento, além de utilizar melhor os recursos de comunicação. Os mecanismos de sincronização permitem explorar a concorrência em programas onde existe a necessidade de coordenar o acesso a recursos e controlar as comunicações, limitando a concorrência. As técnicas de comunicação em aglomerados de computadores possibilitam aumentar a área de memória, uma vez que um grande programa pode ser dividido em várias tarefas menores que podem ser atribuídas aos nós que compõem o aglomerado de computadores. Como conseqüência, há um aumento de custos para sincronização das tarefas. Algumas bibliotecas de comunicação facilitam a implementação de programas que devem ser executados em aglomerados de computadores e apresentam estratégias que buscam minimizar o impacto do overhead das comunicações, tais como ANAHY, que utiliza de técnicas de *multithreading* para este fim.

De uma forma mais genérica verifica-se que o uso conjunto de mecanismos assíncronos de comunicação e de multi-programação leve são comuns para o desenvolvimento de aplicações para processamento de alto desempenho.

4 TÉCNICAS PARA OTIMIZAR O MODELO DE POTTS CELULAR

Como foi visto no Capítulo 2, o modelo de Potts celular apresenta-se bastante ineficiente na sua forma tradicional. Neste capítulo serão apresentados conceitos necessários para aperfeiçoamento deste modelo, buscando otimização e ganho de velocidade em simulações, propiciando a execução de simulações de grande porte em um curto período de tempo.

4.1 Paralelização do algoritmo convencional

Este algoritmo busca desenvolver, a partir do algoritmo convencional, uma forma de concorrência para execução em arquiteturas paralelas, visando atingir altos índices de desempenho, o que geralmente é uma tarefa árdua para o programador, uma vez que fica a cargo deste escrever a concorrência da aplicação e detectar as dependências de dados (WRIGHT et al., 1997). Além disto existe a necessidade de explorar corretamente os recursos da arquitetura utilizada, a fim de obter uma implementação eficiente para uma configuração específica de hardware.

A técnica de paralelização deste algoritmo consiste em dividir a rede que armazena os rótulos em blocos (Figura 9) e cada um destes pode ser entregue a um processador juntamente com dados referentes aos blocos vizinhos (WRIGHT et al., 1997; KORNISS; NOVOTNY; RIKVOLD, 1999). Conforme a dimensão utilizada na simulação, os blocos podem ser um quadrado ou cubo retirado da rede. Dentro destes blocos cada processador disponível poderá efetuar os mesmos cálculos de troca de rótulo que seriam feitos normalmente

na arquitetura seqüencial. Uma exceção deste caso ocorre nas bordas dos blocos, local onde existe a necessidade de utilizar informação correspondente a um bloco diferente do local. Como solução para este problema, os dados das bordas são previamente armazenados possibilitando sua consulta durante a execução, sem a necessidade de comunicação, corrigindo este problema. Ao final de cada passo da simulação, os dados das bordas de cada bloco devem ser atualizados gerando um custo para a execução deste algoritmo. Dependendo do tamanho da rede de rótulos da simulação, este custo de transmissão de dados a cada passo termina por se tornar insignificante, não prejudicando o desempenho final do algoritmo.

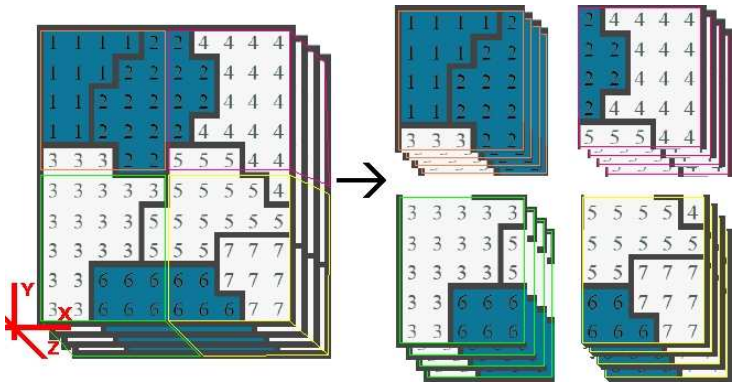


Figura 9: Separando a rede de dados em sub-regiões para processamento paralelo.

4.2 Algoritmo N-Fold Way

O algoritmo N-Fold Way é uma técnica que acelera o algoritmo de Monte Carlo, utilizado no modelo de Potts padrão. Nesta técnica é criada uma tabela que armazena todos os estados possíveis de cada rótulo da simulação de forma ordenada, visando classificar os rótulos de acordo com a probabilidade de ocorrer uma troca do seu valor (BORTZ; KALOS; LEBOWITZ, 1975; NOVOTNY, 2001). A otimização é feita escolhendo com maior probabilidade os rótulos mais prováveis de serem alterados. A cada troca do valor do rótulo deve ser feita uma atualização na tabela que armazena os estados do rótulo escolhido e dos seus vizinhos mais próximos, de acordo com a dimensão da simulação. Existe uma

relação entre os passos de Monte Carlo do algoritmo convencional e N-Fold Way. Portanto, este algoritmo apresenta resultados equivalentes (KORNISS; NOVOTNY; RIKVOLD, 1999; KORNISS et al., 1999).

Este algoritmo apresenta como desvantagem baixo desempenho na sua inicialização, uma vez que há um grande custo associado à classificação dos rótulos e seu armazenamento em memória (BORTZ; KALOS; LEBOWITZ, 1975). Este custo está diretamente relacionado à quantidade de rótulos diferentes existentes na simulação. E, dependendo da quantidade de rótulos vizinhos da simulação, a atualização da tabela de classificação pode se tornar mais custosa que um passo do algoritmo convencional que envolve apenas operações locais com um pequeno número de rótulos vizinhos. Como conseqüência, o algoritmo convencional é mais rápido que o algoritmo N-Fold Way no começo da simulação. Como solução para este problema pode-se utilizar o algoritmo convencional para iniciar a simulação e mais tarde mudar para o N-Fold Way. No entanto, isto pode acarretar em um custo computacional para simulações onde existem muitos rótulos diferentes, uma vez que há um custo para classificar os rótulos para execução do N-Fold Way (WRIGHT et al., 1997).

4.3 Algoritmo Masking

Uma das ineficiências do algoritmo convencional é a quantidade de tempo gasta na tentativa de realizar trocas de rótulos em uma região onde todos os rótulos possuem o mesmo número identificador, sabe-se que neste local estas tentativas não obterão resultado algum (WRIGHT et al., 1997). A proposta deste algoritmo é solucionar esta ineficiência, designando uma variável MASK para cada rótulo da rede, esta variável é verificada toda vez que se desejar realizar uma troca. O MASK=0 indica que os vizinhos do rótulo possuem uma identificação semelhante e MASK=1 o contrário. Os valores desta variável são alterados somente quando o rótulo sofre uma troca, ou seja, quando um rótulo é visitado e alterado. Este algoritmo gera um custo extra de memória e atualização da

matriz responsável por identificar os rótulos aptos para trocas.

4.4 Considerações

Este capítulo apresentou três técnicas que visam contornar os principais problemas do algoritmo padrão do modelo de Potts celular, possibilitando um aumento de desempenho.

A paralelização do algoritmo convencional possibilita a divisão de uma simulação qualquer em vários blocos e cada um destes blocos é entregue para um processador. Esta técnica permite obter um ganho de desempenho e, quando utilizado em aglomerados de computadores, torna possível a execução de simulações de grandes proporções que normalmente não podem ser executadas em um único computador por exigir muita memória para armazenar os dados dos rótulos. Apesar de apresentar muitas vantagens, esta técnica possui as mesmas características do algoritmo convencional (Seção 2.4), ou seja, uma grande quantidade de cálculos é desperdiçada nesta técnica.

O algoritmo N-Fold Way apresenta uma técnica que leva em consideração a possibilidade de ocorrerem trocas de valores nos rótulos da simulação, classificando estes rótulos em ordem, de acordo com sua possibilidade de troca. Este algoritmo busca explorar as características do algoritmo convencional procurando tirar vantagens destas para obter ganho de desempenho. Como desvantagem, este algoritmo apresenta um desempenho inferior ao do algoritmo convencional em simulações onde existem grandes quantidades de rótulos diferentes, uma vez que este algoritmo exige um certo tempo de processamento para poder classificar todos os rótulos da simulação. Além disto, este algoritmo exige uma quantidade extra de memória para armazenar a tabela que guarda todos os estados de cada rótulo da simulação.

Já o algoritmo Masking busca catalogar todos os rótulos que compõem a rede de uma simulação qualquer, identificando e executando os cálculos para efetuar trocas de valores somente nos rótulos que têm condições para tal. Esta técnica visa identificar

rótulos pertencentes a um mesmo agrupamento, local onde não existe a necessidade de se efetuar troca de valores, buscando desta forma eliminar um dos problemas de ineficiência do algoritmo convencional (Seção 2.4).

Este capítulo mostrou abordagens para melhorar o desempenho obtido na execução de simulações, através de transformações no algoritmo, procurando manter a abordagem e características originais. No próximo capítulo são apresentadas as técnicas utilizadas para desenvolver um programa que permite visualizar agregados celulares em três dimensões. A seguir no Capítulo 6 é apresentado um novo algoritmo, objetivo deste trabalho, que busca reduzir o tempo de execução do Modelo de Potts celular e torna possível a execução de simulações de sistemas grandes.

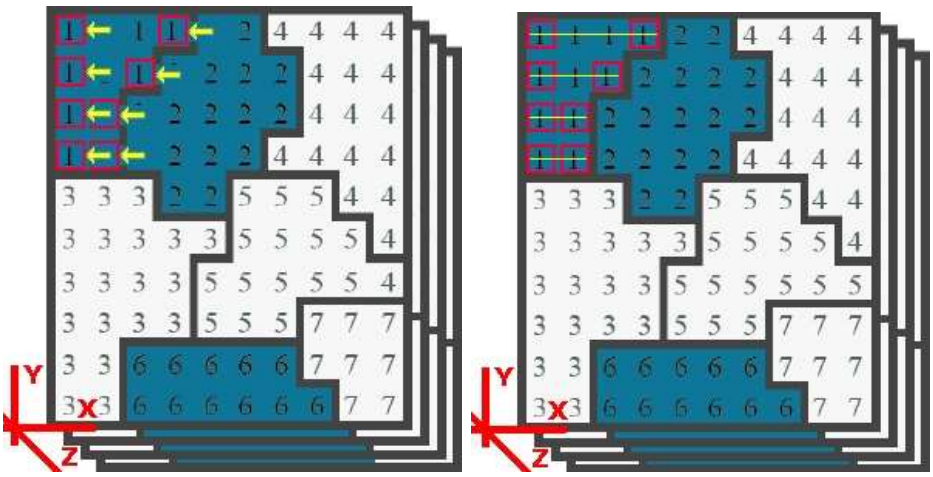
5 VISUALIZADOR DE AGREGADOS 3D

Dada a necessidade de visualizar resultados obtidos a partir do simulador do modelo de Potts, foi desenvolvido um sistema capaz de gerar imagens e filmes que permitam observar o decorrer de uma simulação. A visualização, cada vez mais, cumpre um papel crucial na decisão de estratégias de desenvolvimento dos modelos, pois permite identificar imediatamente situações que devem ser corrigidas e permitem aprimorar o modelo.

5.1 Algoritmo para visualizar agregados

Para visualizar sistemas com um grande número de células (dezenas de milhares), se desenvolveu um algoritmo que busca, a partir de primitivas básicas de desenho (GL_LINES) da biblioteca OpenGL (WOO; NEIDER; DAVIS, 1996). A técnica utilizada é a de realizar uma busca em toda a rede cúbica que forma o agregado celular a partir de secções perpendiculares ao eixo Z (Figura 3 - página 20). Para cada corte perpendicular ao eixo Z, o algoritmo procura as coordenadas X, Y, iniciais e finais, que compõem cada célula. Como se pode observar na Figura 10 (a), o grupo de números '1' representa uma célula qualquer, e tem suas coordenadas X inicias e X finais encontradas (CERCATO; MOMBACH, 2005). Estes rótulos foram destacados apenas para facilitar a compreensão do funcionamento deste algoritmo. O último passo para gerar a imagem do agregado celular consiste em unir as coordenadas X iniciais e finais de cada célula com linhas (GL_LINES) (WOO; NEIDER; DAVIS, 1996), Figura 10 (b). Para possibilitar diferenciar uma célula das demais, a intensidade da cor das linhas é distribuída de acordo com o gradiente da posição celular em relação a um eixo perpendicular ao eixo Z. Para observar a evolução de uma

simulação qualquer, foi desenvolvido um sistema que permite, através de um pequeno arquivo de configuração, capturar seqüências de imagens que são geradas pela biblioteca OpenGL. Posteriormente existe a possibilidade de armazenar estes dados em um formato de arquivo de imagem padrão BITMAP, com a finalidade de aplicar efeitos, tornando possível desenvolver uma animação.



(a) Detecção (b) Conexão

Figura 10: Detecção e conexão, com linhas, das coordenadas iniciais e finais de uma célula.

5.2 Resultados

Com a técnica de visualização apresentada foi possível implementar uma ferramenta que permite observar os resultados de simulações. Pode-se observar na Figura 11 uma visão geral da interface deste programa. Na primeira janela de controle se encontra a caixa de texto “Passos”, responsável por definir qual arquivo de simulação será exibido. Logo abaixo se encontram controles específicos para realizar cortes e efeito de transparência (“Exibir”) no agregado celular. Estes controles permitem escolher a melhor forma de exibir uma imagem detalhada de uma experimentação qualquer. Os controles que não foram agrupados (localizados à direita) são responsáveis por rotacionar e transladar o agregado celular. Por fim, na caixa de texto localizada no subitem “Imagem”, estão os comandos para captura de tela e execução de um arquivo de script para executar estes comandos. Este sistema permite a captura de seqüências de imagem usando um arquivo

de configuração (Figura 12), que busca automatizar esta tarefa que pode se tornar onerosa para o operador, conforme o número de efeitos que se deseja aplicar à animação do agregado celular e o número de passos resultantes da simulação. Nas figuras 13 e 14 são demonstradas algumas simulações executadas neste trabalho. As demais imagens de agregados celulares 3D apresentadas nesta dissertação foram obtidas com auxílio desta ferramenta.

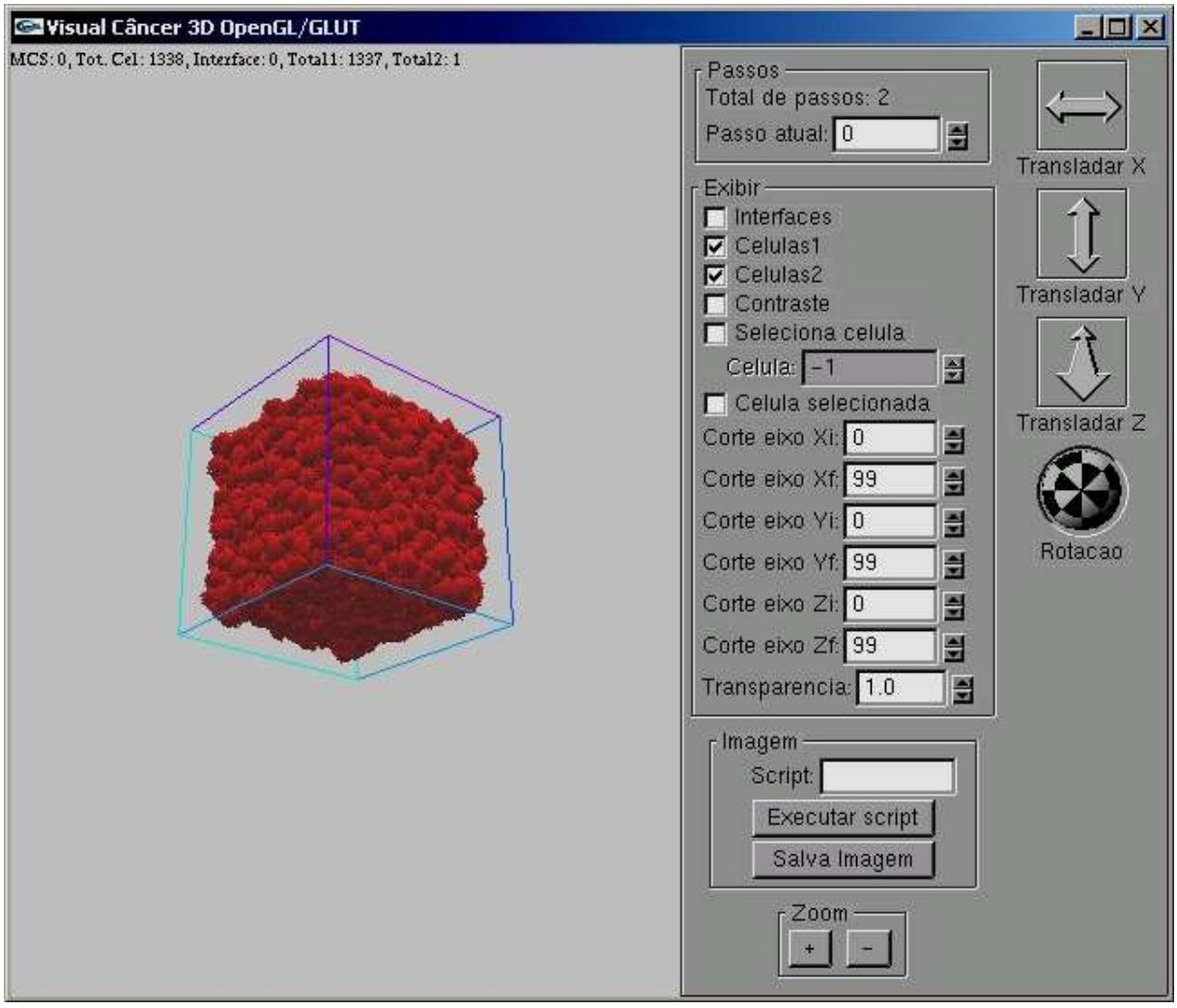


Figura 11: Interface do programa visualizador.

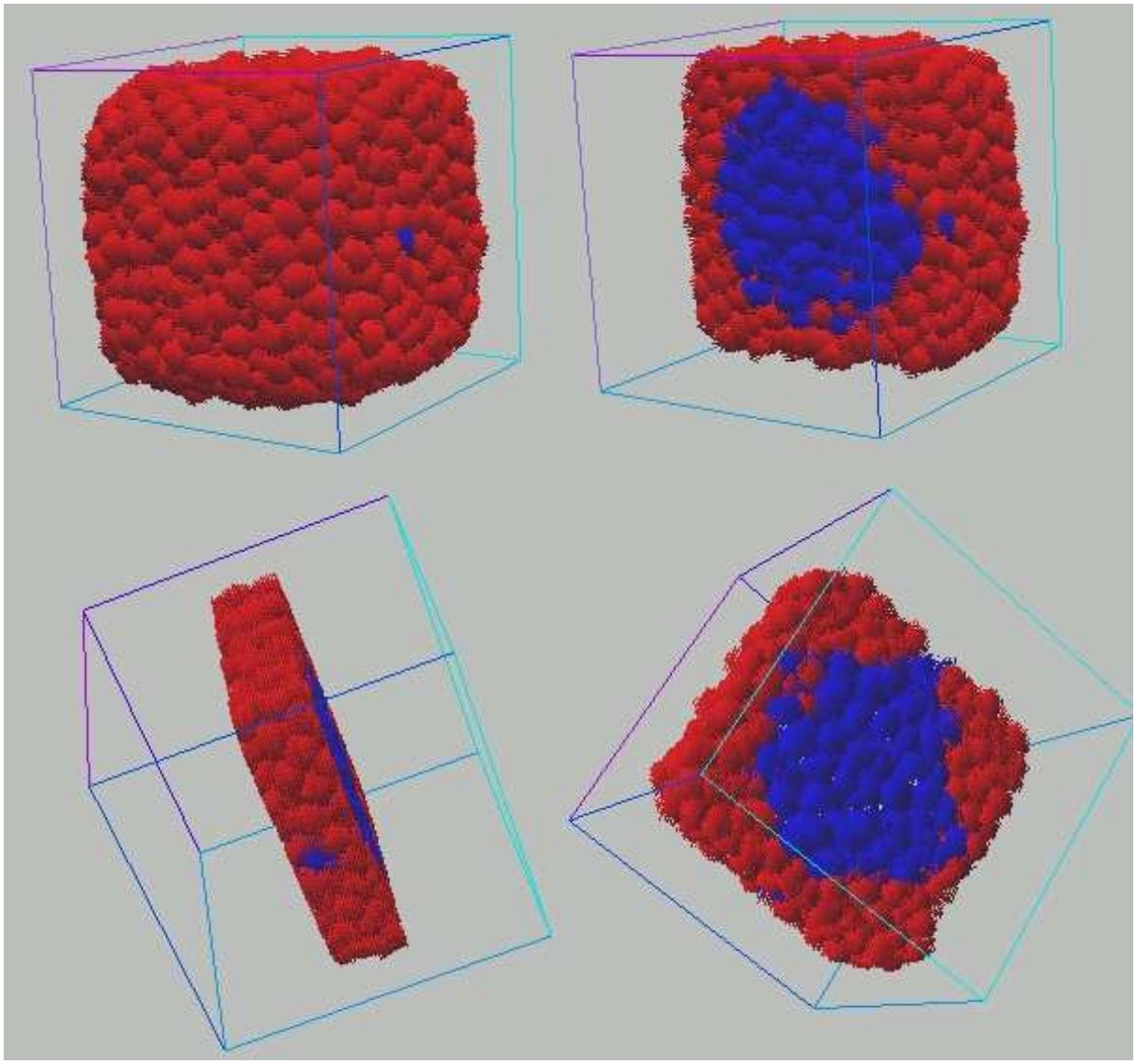


Figura 12: Imagens capturadas pelo programa visualizador.

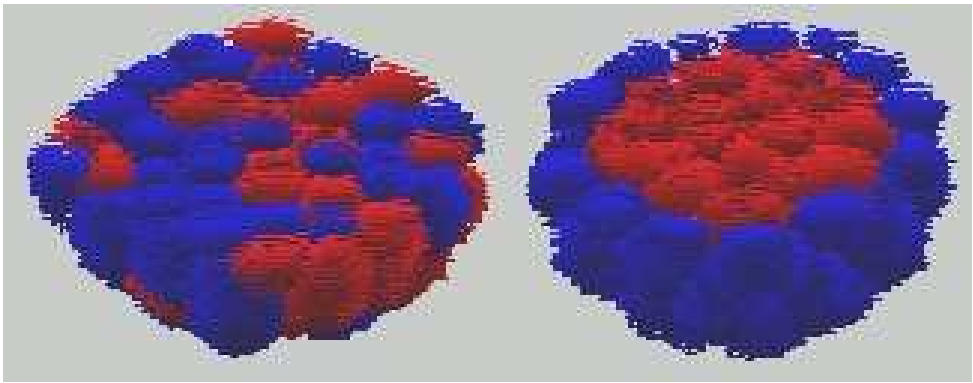


Figura 13: Simulação de segregação celular.

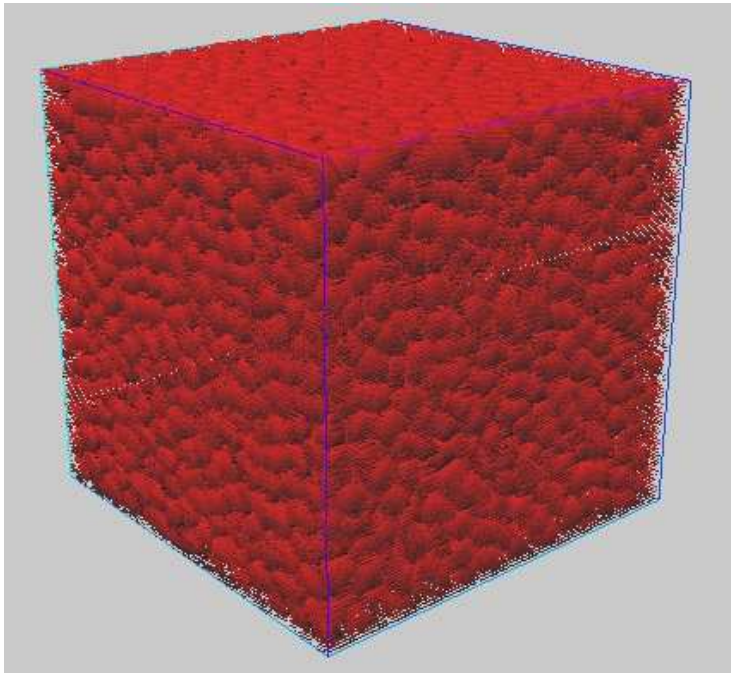


Figura 14: Simulação de espumas de sabão.

5.3 Considerações

A implementação do sistema visualizador tornou possível observar, comparar e até mesmo validar resultados de uma simulação qualquer a partir de imagens geradas dos resultados de simulações do modelo de Potts celular. Além disto este sistema permite gerar filmes, tornando possível acompanhar a evolução de uma simulação qualquer. Durante o processo para criar filmes é possível adicionar efeitos nas imagens que compõem um filme, possibilitando eliminar, colorir e segmentar grupos de células, possibilitando visualizar um grupo específico de células. Este sistema visualizador foi desenvolvido com funções que podem ser modificadas de forma a permitir gerar imagens de vários tipos de simulações em 3D, tornando este sistema útil para outros trabalhos.

6 CAMINHANTES ALEATÓRIOS

O modelo de Potts celular é utilizado para representar as propriedades das células, sendo capaz de simular diversos tipos de fenômenos celulares. Na sua forma convencional, o modelo de Potts celular apresenta um grande custo computacional, uma vez que este algoritmo possui uma série de operações e cálculos que podem ser executados de maneira pouco eficiente e, dependendo do tipo e do tamanho da simulação a ser realizada, a quantidade de memória exigida pelo algoritmo impossibilita a execução da simulação nas arquiteturas de computadores atuais.

Este capítulo apresenta inicialmente uma técnica de implementação do modelo de Potts baseada no algoritmo de Caminhantes Aleatórios (CA's) proposto por Éder Gusatto (GUSATTO, 2004) para arredondamento de agregados celulares biológicos em 2D. Neste trabalho o algoritmo de CA's foi estendido tornando possível a execução de simulações em 3D (GUSATTO; MOMBACH; CERCATO; CAVALHEIRO, 2005), além de receber a capacidade de execução concorrente com objetivo de aumentar o desempenho do modelo de Potts, gerando um novo algoritmo capaz de ser executado em paralelo, buscando reduzir o seu tempo de execução.

Em seguida, é apresentada uma forma de distribuir o algoritmo de CA's, desenvolvida neste trabalho, que torna possível sua execução em aglomerados de computadores, de forma a possibilitar a execução de simulações com um tamanho e realismo maiores como dito anteriormente, conforme o tamanho da simulação o algoritmo exige uma grande quantidade de memória para a sua execução, além de demandar um tempo de processamento alto, o que torna a execução em aglomerados de computares uma alternativa atraente

para a solução deste tipo de problema que demanda grande quantidade de recursos computacionais de processamento e armazenamento, além de serem economicamente viáveis e de fácil manutenção (VRENIOS, 2002; ANDREWS, 2000; HWANG; XU, 1998).

6.1 Algoritmo de Caminhantes Aleatórios

O algoritmo de Caminhantes Aleatórios busca apresentar uma solução para os problemas de ineficiência computacional do modelo de Potts celular (Seção 2.4). Neste trabalho uma nova abordagem de implementação utilizando a dinâmica de Metropolis procura reduzir a quantidade de sorteios de rótulos que não pertencem à borda de forma a aumentar o desempenho de uma simulação (GUSATTO, 2004).

Para isto o algoritmo de CA´s tem como premissa a realização do menor número possível de sorteios com a finalidade de encontrar o primeiro rótulo válido (a primeira borda) e, a partir desse ponto, a evolução do algoritmo é obtida pela técnica de percorrer apenas as bordas das células (Figura 5 - página 24) (GUSATTO et al., 2005). Existe a possibilidade do percurso alcançar um rótulo que não possui vizinhos com rótulos diferentes, neste caso surge a necessidade de realizar um novo sorteio para encontrar uma nova borda válida. Esta implementação elimina em grande parte o desperdício computacional com buscas aleatórias desnecessárias.

O fato de não eliminar completamente o sorteio de bordas garante que o processo de transição de estados do algoritmo de Metropolis possa atingir qualquer estado do sistema a partir de um outro estado qualquer, se este processo for executado por tempo suficiente. Garantindo assim o controle de possíveis correlações e comportamentos tendenciosos que um CA, preso em um determinado setor do agregado, poderia vir a gerar.

Este algoritmo permite o uso de um número arbitrário de caminhantes aleatórios que atualizam os rótulos de borda reduzindo a complexidade do algoritmo para $O(n^{d-1})$.

A Figura 15 apresenta o funcionamento do algoritmo de CA´s. O primeiro passo

deste algoritmo consiste em escolher aleatoriamente um rótulo válido para troca, ou seja, rótulo pertencente a borda de uma célula qualquer. A seguir é realizado um teste para verificar se existem rótulos vizinhos ao rótulo escolhido que possa realizar uma transição de estado, de acordo com o algoritmo de Metropolis (Seção 2.1). Serão considerados válidos todos os rótulos vizinhos que possuírem valor diferente do rótulo escolhido, um destes rótulos vizinhos é escolhido aleatoriamente e, de acordo com a probabilidade de transição de estado, o valor do rótulo escolhido pode vir a ser substituído pelo valor do rótulo vizinho. Independente do resultado desta transição, o rótulo vizinho passa a ser o próximo rótulo escolhido, dando continuidade ao algoritmo. O tempo é definido como Passo de Caminhante Aleatório (PCA), ou seja, executar os CA's uma quantidade de vezes igual a quantidade de rótulos pertencentes à borda de todas as células.

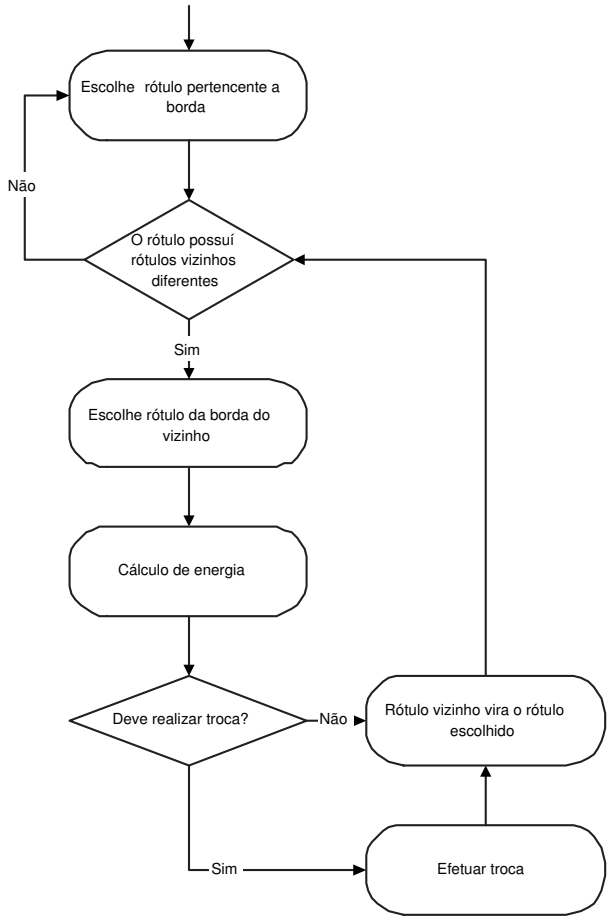
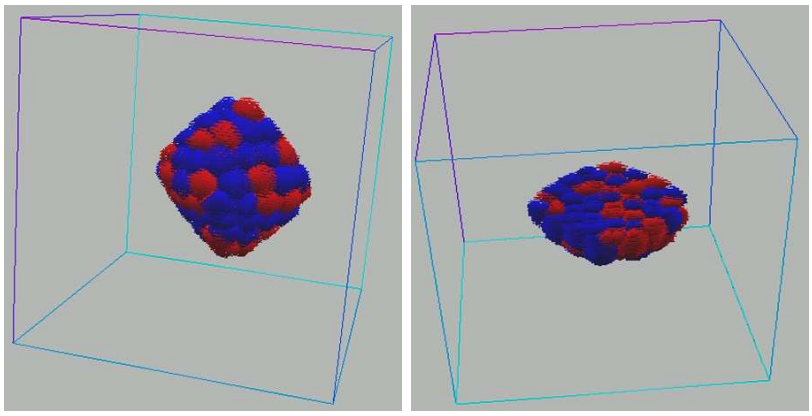


Figura 15: Fluxograma do algoritmo de CA's.

6.1.1 Resultados

Para testes de desempenho foi realizado um conjunto de simulações que efetuam a segregação celular (IZAGUIRRE; CHATURVEDI; HUANG, 2004). Os resultados do algoritmo de CA's foram comparados com resultados do algoritmo de Monte Carlo. Foi utilizado nesta simulação uma rede de dados cúbica com 100^3 rótulos. O mesmo estado inicial e conjunto de parâmetros foram utilizados para comparar o desempenho do algoritmo. O estado inicial da simulação é um agregado esférico com dois tipos de células misturados aleatóriamente envolto pelo meio externo (Figura 16). Os demais parâmetros são apresentados na Tabela 1, para este caso os valores da energia de adesão foram obtidos a partir das equações de tensão superficial (equações 2.2, 2.3 e 2.4 - página 22).



(a) Visão geral (b) Visão em corte

Figura 16: Estado inicial do agregado celular composto por 43 células vermelho claro e 69 células azul escuro no seu estado inicial.

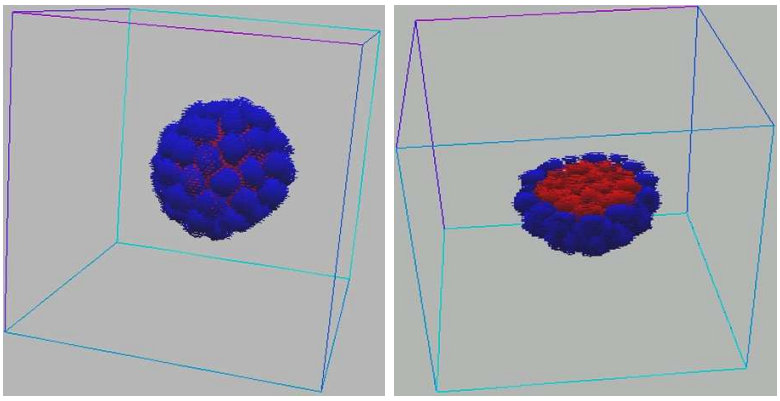
Tabela 1: Parâmetros utilizados na simulação de segregação celular.

Valores	Parâmetros
T = 15	temperatura
$V_T = 512$	volume alvo das células
$\lambda = 1$	constante de compressibilidade das células
$e_{dd} = 2$	energia de adesão entre células escuras
$e_{ll} = 6$	energia de adesão entre células claras
$e_{dl} = 5$	energia de adesão entre células escuras e claras
$e_{dM} = 5$	energia de adesão entre células escuras e o meio
$e_{lM} = 4$	energia de adesão entre células claras e o meio

Nesta simulação utilizou-se como parâmetro de controle de evolução das simulações a área total da interface de contato entre células claras e escuras, que é minimizada pelo processo de segregação. Os resultados são obtidos através da média entre 5 execuções da simulação. Foram gastos aproximadamente o mesmo intervalo de tempo de simulação, 8.000 passos, para alcançar o mínimo de energia. Depois foi utilizado o tempo real gasto para chegar nesta quantidade de passos. Para fins comparativos estabelecemos um intervalo de tempo de referência um pouco maior, 10.000 passos (Figura 17). Pode-se observar na Tabela 2, o resultado do tempo real gasto em um computador com processador Intel Xeon 2.66 GHz, 1.5GB de memória, rodando Linux Gentoo e compilador GCC 2.92. Com este algoritmo foi possível obter um ganho de aproximadamente 7 vezes (G1 da Tabela 2) com relação ao algoritmo MC padrão (GUSATTO et al., 2005). Isto mostra que a técnica de evitar sorteios para encontrar rótulos em posições de borda válidas, assim como percorrer somente as bordas das células é bastante eficiente na execução do algoritmo.

Tabela 2: Média de tempo em segundos gastos para cada algoritmo na simulação de segregação celular, em computador mono-processado Intel Xeon 2.66 GHz.

Algoritmo	Tempo(s)	G1
MC	19.560±557	-
CA	2.864±26	6,82

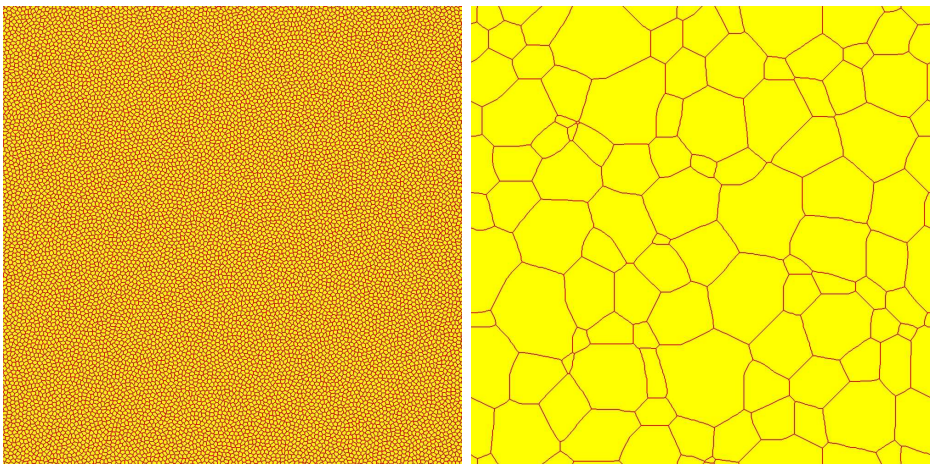


(a) Visão geral (b) Visão em corte

Figura 17: Estado final do agregado celular composto por 43 células vermelho claro e 69 células azul escuro no seu estado final.

Para verificar o comportamento do algoritmo de CA e do algoritmo de Monte Carlo

foram realizadas uma média de 10 simulações de espumas de sabão (GLAZIER; WEAIRE, 1992) em duas e três dimensões (GUSATTO et al., 2005). As simulações em duas dimensões utilizam uma rede de 1000^2 rótulos, fazendo uso de um estado inicial com bolhas com área média de 78,65 rótulos (Figura 18 (a)). Para as simulações em três dimensões foi utilizada uma rede de 100^3 rótulos, neste caso o estado inicial apresenta bolhas com volume médio de 33,67 rótulos.



(a) Estado inicial (b) Estado final

Figura 18: Estados da simulação de espumas de sabão 2d.

Nestas simulações o parâmetro de compressibilidade das células (λ) é zero. As células podem crescer ou encolher para minimizar a energia da superfície no sistema como ocorre nas bolhas de sabão reais (GLAZIER; WEAIRE, 1992). Os parâmetros destas simulações são apresentados na Tabela 3. Foi utilizado como critério de parada das simulações a chegada ao estado de escala, estado em que as distribuições estatísticas do sistema não mudam mais de forma.

Tabela 3: Parâmetros utilizados na simulação de bolhas de sabão.

Valores	Parâmetros
$T = 0$	temperatura
$\lambda = 0$	constante de compressibilidade das células
$e = 1$	energia de interação entre bolhas

Na Figura 19 se apresenta a medida da lei de von Neumann (NEUMANN, 1952)

para a simulação de bolhas de sabão 2D, apresentando uma boa linearidade de acordo com a Equação Teórica 6.1:

$$\dot{a}_n = k(n-6) \tag{6.1}$$

Onde $\dot{a}$ representa a variação no tempo da área de uma bolha de n lados, k é uma constante de difusão e n indica o número de lados da bolha. A área das bolhas com menos de seis lados tende a diminuir de tamanho, já as bolhas de seis lados têm a tendência de permanecer estáveis, por fim as bolhas com mais de seis lados têm a tendência de crescer.

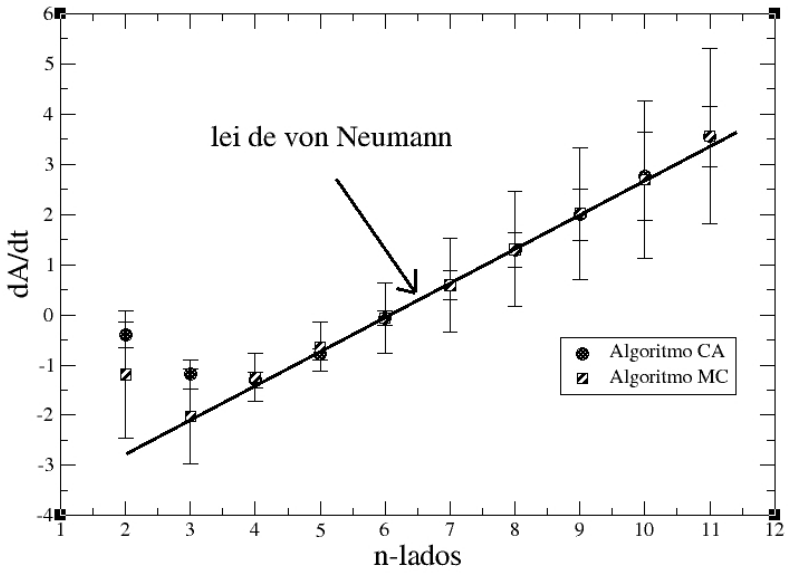


Figura 19: Resultados das simulações de espumas 2d.

A Figura 20 apresenta os resultados para as simulações em três dimensões. Os resultados em duas e três dimensões foram re-escalados de forma a tornar possível a sua comparação com os dados obtidos com o algoritmo padrão.

Os resultados das simulações de espumas utilizando o Modelo de Potts quando comparadas com resultados de experimentos reais em 3D apresentam discordância para bolhas pequenas. Isto é um artefato da simulação oriundo da utilização de um espaço discreto (GLAZIER, 1993; JIANG, 1998; MOMBACH, 1997).

O algoritmo de CA's mostra uma pequena discordância com os resultados do algo-

ritmo padrão para bolhas pequenas, isto é, bolhas com 3 lados em 2D e bolhas com faces entre 4 e 6 em 3D (Figuras 19 e 20). Estes resultados sugerem que o algoritmo de CA's não é exato, mas fornece uma ótima aproximação dos resultados do algoritmo padrão, sendo uma ótima estratégia para obtenção de resultados com rapidez, sem consumo adicional de memória.

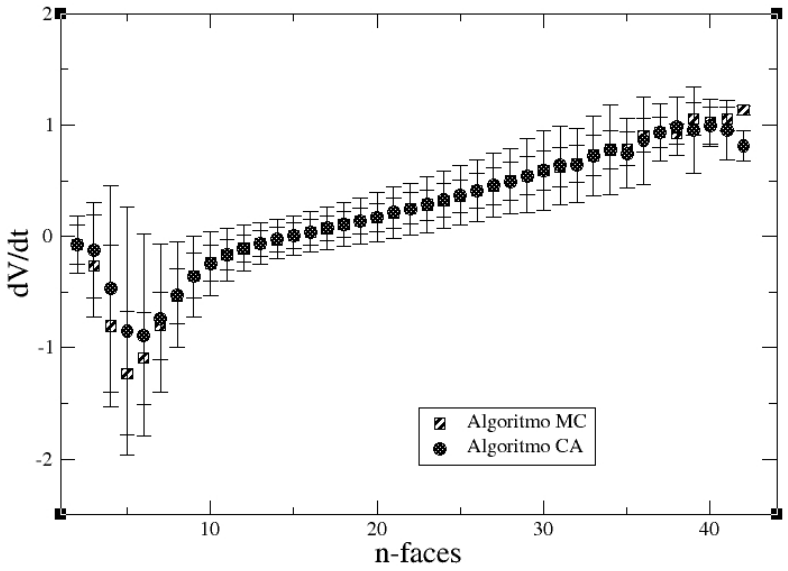


Figura 20: Resultados das simulações de espumas 3d.

6.2 Algoritmo de Caminhantes Aleatórios concorrente

Com base no algoritmo de Caminhantes Aleatórios apresentado na Seção 6.1, buscou-se detectar pontos onde existe a possibilidade de execução concorrente que permitisse desenvolver uma nova extensão deste algoritmo, buscando aumento de desempenho. A seguir será apresentado o algoritmo de CA's paralelo, e serão mostrados os resultados obtidos em algumas simulações com este novo algoritmo. Por fim é apresentado o algoritmo de CA's distribuído e os resultados das simulações.

6.2.1 Paralelizando o algoritmo de CA's

Nesta implementação foram exploradas técnicas de *thread-aware* (Seção 3.4.2) para permitir ao algoritmo executar em uma arquitetura SMP (ANDREWS, 1991).

O paralelismo do algoritmo de Caminhantes Aleatórios pode ser facilmente obtido explorando a concorrência natural do problema através da execução simultânea de vários CA´s conforme a necessidade. Nesta implementação, cada execução é suportada por uma *thread*. No entanto estas *threads* compartilham informações do mesmo conjunto de dados, e quando ocorre um encontro de CA´s pode ocorrer uma sobre-escrita dos valores dos rótulos.

Implementações típicas de programas *multithread* (Seção 3.4) podem empregar mecanismos de sincronismo para evitar acesso simultâneo a dados compartilhados (Seção 3.4.1). Nesta implementação um controle do tipo mutex é utilizado nas *threads* para evitar acesso simultâneo aos dados compartilhados.

Considerando que o número de *threads* (CA´s) depende diretamente do número de processadores disponíveis (Tabelas 4 e 5), a probabilidade de duas ou mais *threads* calcularem o mesmo conjunto de dados é pequena se a fração entre o número de rótulos das bordas e o total de rótulos no sistema é pequena (Tabela 6). Assim, uma implementação mais eficiente deve tirar vantagens deste aspecto.

Tabela 4: Média de tempo em segundos gastos para cada algoritmo na simulação de segregação celular, em um computador com 2 processadores Intel Xeon 2.66 GHz.

Algoritmo	Tempo(s)	G1	G2
MC seqüencial	19.560±557	-	-
CA seqüencial	2.864±26	6,82	-
CA paralelo 1 *thread*	2.360±22	8,28	21
CA paralelo 2 *threads*	2.145±6	9,11	33
CA paralelo 3 *threads*	2.157±11	9,06	32

Esta implementação assume que as operações de leitura de dados podem ser executadas de forma concorrente com qualquer outra operação, mas as operações de escrita devem ser executadas seqüencialmente, mesmo se chamadas simultaneamente. Esta estratégia evita uso desnecessário de operações de mutex, da mesma forma que garante que os dados compartilhados serão atualizados corretamente. Situações indesejadas podem ocorrer neste caso, quando os dados são lidos por uma *thread* e simultaneamente alterados

por outra *thread*, ou quando duas ou mais *threads* alteram simultaneamente os mesmos dados. Em ambos os casos ocorre erros nos dados.

Tabela 5: Média de tempo em segundos gastos para cada algoritmo na simulação de segregação celular, em um computador 4 processadores Intel Xeon 1.6 GHz.

Algoritmo	Tempo(s)	G1	G2
MC seqüencial	31.258±653	-	-
CA seqüencial	3.812±17	8,19	-
CA paralelo 2 *threads*	3.256±10	9,60	17
CA paralelo 3 *threads*	3.074±5,5	10,16	24
CA paralelo 4 *threads*	2.989±7,8	10,45	27

Tabela 6: Quantidade de dados descartados na execução de uma caminhada de aproximadamente 100.000.000 rótulos

Total de CA's	Quantidade descartada
2	354
4	3.396

Nesta implementação estes dois casos são tratados separadamente. Para o problema de escrita simultânea foi adotado o uso tradicional do mutex, para a gravação dos dados tornar-se seqüencial. Para o problema leitura e escrita simultânea foi usada uma estratégia diferente, de forma a explorar as características do algoritmo de CA´s. A operação de leitura dos dados (acesso a rótulos) é uma operação concorrente, ou seja, mais de um CA pode consultar simultaneamente um conjunto de rótulos de forma independente as demais operações. E para evitar erros nos dados é executado um procedimento que verifica se os dados lidos de forma concorrente permanecem inalterados, e em caso de surgir uma mudança nos dados estes são descartados e o CA é reiniciado. Este procedimento é executado antes das operações de escrita e tem como objetivo evitar erros nos dados da simulação. Isto leva a implementação a descartar um pouco de tempo de processamento, mas aumenta o desempenho de execução. Para ter uma idéia, a Tabela 6 apresenta a quantidade de dados descartados na execução de uma simulação de segregação (Seção 6.1.1).

Como se pode observar na Figura 21, o algoritmo de CA's paralelo apresenta um funcionamento similar ao algoritmo de CA´s seqüencial. O principal diferencial deste algoritmo é a seção crítica adicionada na etapa de escrita dos dados de forma a tornar esta operação seqüencial. Além disto a operação de leitura, por ser concorrente, é seguida de uma operação de armazenamento em memória não compartilhada. Desta forma torna-se possível comparar o valor destes dados da memória compartilhada e não compartilhada em toda operação de escrita, possibilitando detectar erros nos dados devido a operações de escrita simultânea por dois ou mais CA's.

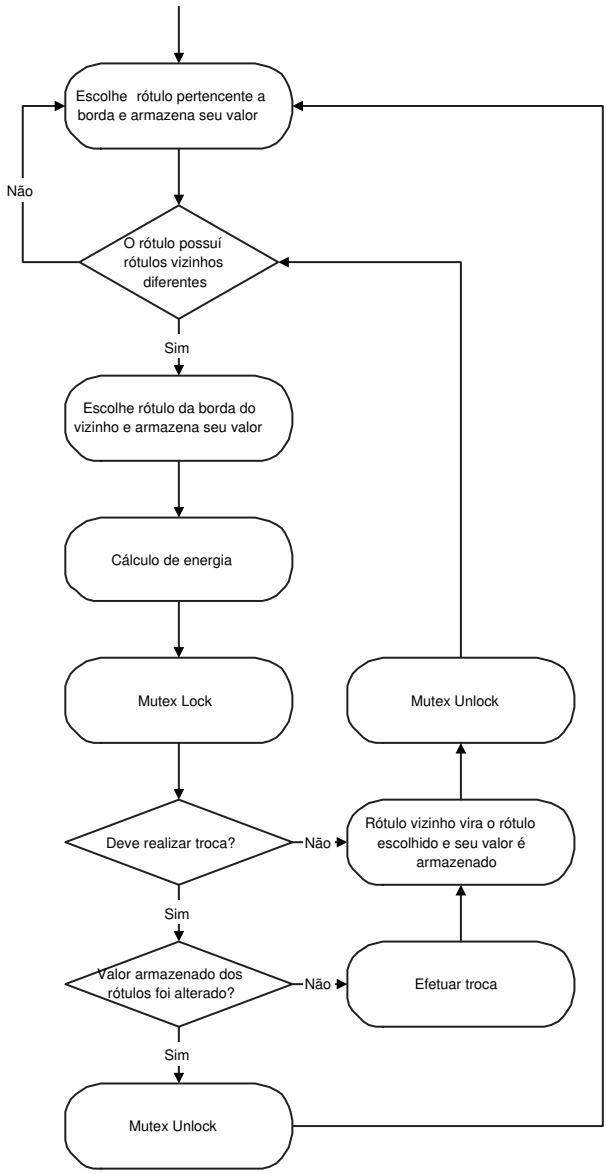


Figura 21: Fluxograma do algoritmo de CA's paralelo.

6.2.1.1 Resultados

Os mesmos testes descritos na Seção 6.1.1 foram refeitos com o novo algoritmo paralelo desta vez utilizando um computador bi-processado Intel Xeon 2.66 GHz, 1.5GB de memória e um quadri-processado Intel Xeon 1.6 GHz, 2GB de memória, ambos os casos usando sistema operacional LINUX Gentoo e compilador GCC 2.96.

As tabelas 4 e 5 (página 62) apresentam os resultados destes testes, mostrando o tempo de execução da simulação de segregação celular, (G1) mostra o ganho de velocidade do algoritmo de CA's em relação ao algoritmo de MC seqüencial e (G2) a porcentagem de ganho de velocidade obtido na execução do algoritmo de CA's paralelo em relação ao CA's seqüencial. O algoritmo de CA's seqüencial é no mínimo 6 vezes mais rápido que o algoritmo MC seqüencial. A execução em paralelo de dois CA's permite um ganho de aproximadamente 17% em relação ao algoritmo CA's seqüencial, ou seja, um ganho de 9 vezes em relação ao algoritmo de MC seqüencial (Tabela 4 (página 62)). Para os casos em que o número de CA's supera a quantidade de processadores disponíveis o fator de ganho tende a cair, isto ocorre devido aos custos associados ao controle de execução de cada CA. Outro fator diretamente associado ao desempenho do algoritmo paralelo é o tamanho da rede de dados. Redes pequenas suportam poucas células e, por conseqüência, possuem poucos caminhos para os CA's percorrerem, aumentando a probabilidade dos CA's passarem por um mesmo caminho simultaneamente, gerando descarte de dados e busca por novas posições de borda para a execução dos CA's reduzindo o desempenho.

6.2.2 Distribuindo o algoritmo de CA's

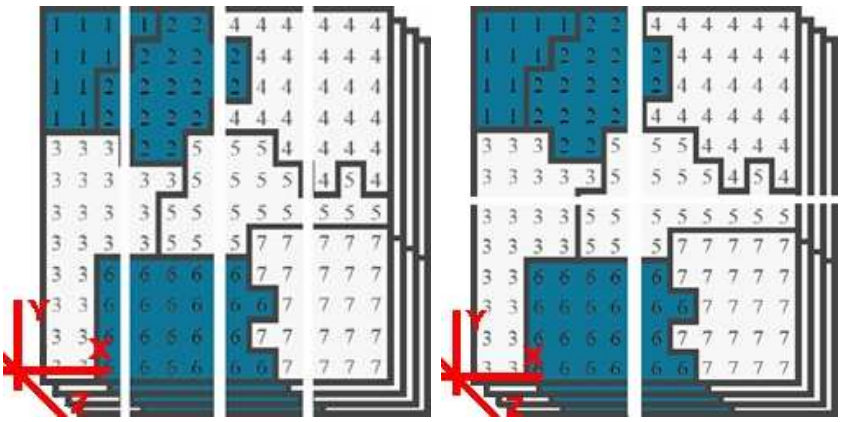
Para esta implementação foi utilizada a biblioteca de comunicação padrão Sockets para a comunicação em aglomerados de computadores. Esta biblioteca possui primitivas para troca de mensagens entre nós que compõem um aglomerado de computadores.

O objetivo desta técnica é segmentar a rede cúbica que armazena os dados referentes ao aglomerado de células em blocos, visando viabilizar a execução de simulações

grandes. Ao segmentar a rede devido a necessidade de interação de um rótulo com seus vizinhos mais próximos, surge uma dependência de dados tornando necessária a comunicação entre nós para permitir a execução normal dos CA's. Visando reduzir a dependência de dados e minimizar a comunicação, foram analisadas duas formas de segmentar a rede de dados, conforme será apresentado a seguir:

- **A segmentação a partir de cortes paralelos em dois ou mais eixos**: São realizados cortes paralelos em dois ou mais eixos visando gerar regiões similares para a execução dos CA's (Figura 22 (b)). Neste caso, como pode se observar na Figura 24, a quantidade de rótulos em regiões de fronteiras é grande, além de surgirem situações onde existe a necessidade de atualização de rótulos em mais de um nó (diagonais inferiores e superiores dos blocos).

- **A segmentação a partir de cortes paralelos a um único eixo**: Esta técnica (Figura 22 (a)), reduz a quantidade de rótulos em regiões de fronteiras (Figura 23), locais onde existe a necessidade de consulta de um ou mais rótulos pertencentes a outro nó do aglomerado de computadores. Neste técnica não existem situações onde se faz necessário atualizar dados em mais de um nó, reduzindo a comunicação. Por apresentar esta característica esta técnica foi a escolhida para o desenvolvimento do algoritmo de CA's distribuído.



(a) Um corte paralelo a um eixo qualquer

(b) Dois ou mais cortes paralelos a eixos

Figura 22: Formas de segmentação.

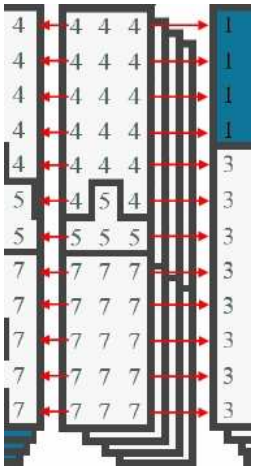


Figura 23: Comunicação existente em um bloco gerado através de um corte paralelo a um eixo qualquer.

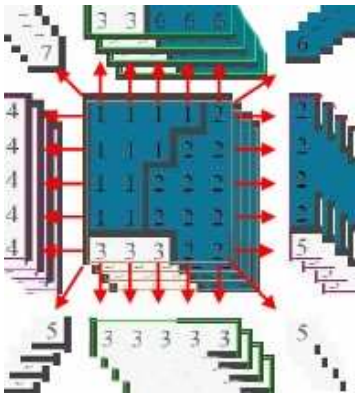


Figura 24: Comunicação existente em um bloco gerado através de dois ou mais cortes paralelos a um eixo qualquer.

Uma vez segmentada a rede, cada bloco é entregue a um nó (Figura 22 (a)) e este fica responsável por executar um ou mais CA's neste bloco. Conforme será apresentado a seguir, foram desenvolvidas duas formas de execução dos CA's nos nós do aglomerado de computadores:

- **CA's padrão:** Onde podem existir vários caminhantes em cada nó, tal como é possível no algoritmo de CA's paralelo (Seção 6.2.1). Quando um CA estiver na borda de um bloco, pode existir a necessidade deste CA passar a executar em outro nó contendo o bloco vizinho para dar continuidade à execução. Neste caso é executada uma troca de mensagens, fazendo com que o CA passe a ser executado no

nó que possui os dados necessários para completar uma caminhada. No decorrer dos testes de implementação, esta técnica se mostrou ineficiente no que diz respeito ao uso dos processadores disponíveis no aglomerado de computadores e por isto foi abandonada. O principal motivo desta ineficiência se dá pelo *overhead* gerado pela comunicação. Isto faz com que a cada mudança que um CA faz de um bloco para o outro surja um período de tempo para comunicação, onde um processador poderia estar ocupado executando uma caminhada de um CA em vez de ficar ocioso.

- **CA's prisioneiro:** Esta técnica visa aproveitar melhor os processadores disponíveis no aglomerado de computadores. Para isto eliminou-se a capacidade dos CA's de serem transferidos de um bloco para o outro, ou seja, o CA fica preso no bloco onde foi criado. Neste caso, quando um CA chega na borda do bloco, é efetuado um salto para uma nova posição válida neste mesmo bloco para continuar sua execução. É feita uma troca de mensagens e a posição pertencente ao outro bloco, que seria o caminho natural deste CA, é selecionada por um CA pertencente a este bloco. Esta posição será selecionada por um CA quando este necessitar de uma nova posição para iniciar uma caminhada. Desta forma elimina-se o problema encontrado na técnica dos CA's padrão de ociosidade de processadores e mantêm-se as mesmas características do algoritmo original.

Em ambas as técnicas de execução dos CA's observou-se uma condição especial que pode ocorrer quando os rótulos escolhidos para efetuar em uma troca (cálculo de energia envolvendo os vinte e seis vizinhos mais próximos, consulte a Seção 2.3) são pertencentes à borda de um bloco, local onde existe a necessidade de buscar estes dados pertencentes às bordas de outros blocos vizinhos. Neste caso, surge a necessidade de comunicação para buscar estes dados pertencentes aos blocos vizinhos. Isto faz com que o processador permaneça ocioso durante esta etapa, fazendo com que a execução perca eficiência. A solução para este caso (Figura 25) foi a utilização da técnica das regiões fantasmas, que faz com que cada nó do aglomerado, além de receber um bloco para execução dos CA's, receba também um conjunto de dados referente às bordas dos blocos vizinhos. Desta

forma, quando o CA estiver na borda do bloco, o nó já possui na memória local uma cópia contendo os dados referentes às bordas dos blocos vizinhos, dados necessários para o cálculo de energia (Seção 2.3). A técnica das regiões fantasmas prevê que, à medida que os rótulos pertencentes à borda de um bloco vão sendo alterados, suas respectivas cópias são atualizadas garantindo a coerência dos dados.

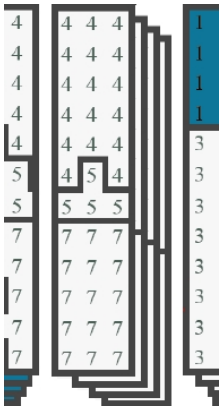


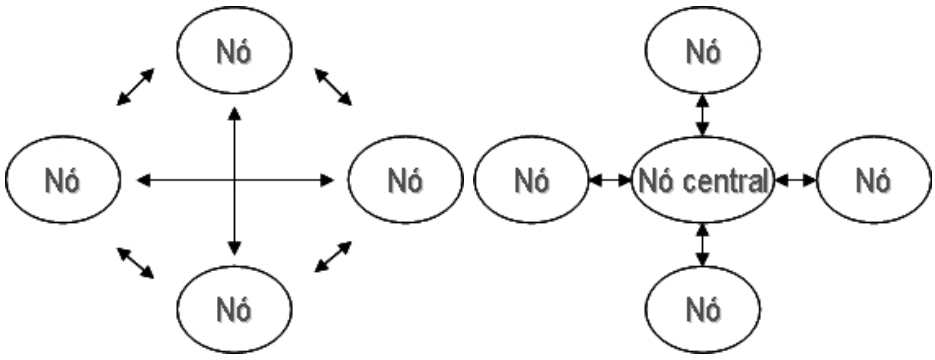
Figura 25: Um bloco de dados e suas regiões fantasmas.

No que diz respeito à comunicação, foram estudadas três formas de implementação do algoritmo utilizando Sockets, buscando a forma que melhor atenda às necessidades dos CA's e facilitando a sua implementação. A seguir são apresentados os modelos de comunicação:

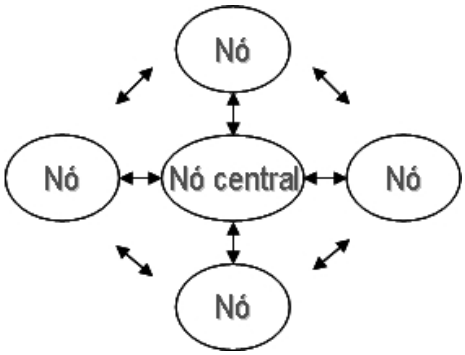
- **Descentralizado:** Esta implementação permite comunicação direta entre todos os nós do aglomerado de computadores (Figura 26 (a)), facilitando a atualização de dados de um bloco para o outro, de acordo com a técnica das regiões fantasmas. Neste modelo, a tarefa de definir como segmentar a rede de dados em blocos e posteriormente reuni-los é dificultada pelo fato de não existir um nó responsável por centralizar os dados das simulações.

- **Centralizado:** Neste modelo existe a figura de um nó centralizador (Figura 26 (b)), que se encarrega de segmentar a rede de dados de uma simulação qualquer e coordenar a execução dos CA's em cada bloco, ou seja, quando existe a necessidade de enviar mensagens de um bloco para outro é feita por intermédio do nó central.

- **Misto:** Neste caso existe um nó central da mesma forma que o modelo anterior, porém neste caso o nó central, além de coordenar a execução dos CA's, também determina que nós contendo blocos vizinhos devem criar um canal de comunicação direta e exclusivo entre si para evitar que toda comunicação, por exemplo, a atualização de um região fantasma, seja feita por intermédio do nó central (Figura 26 (c)).



(a) Descentralizado (b) Centralizado



(c) Misto

Figura 26: Modelos de comunicação.

Dos modelos apresentados acima, o modelo Misto foi escolhido para a implementação do algoritmo de CA's distribuído por se constituir num modelo de controle simples onde toda comunicação é feita de forma direta entre os nós sem a necessidade de passar pelo nó central, evitando a penalização na comunicação entre estes nós, tal como a existente no modelo de comunicação centralizado. A idéia deste algoritmo é buscar formas de reduzir a comunicação entre nós ao mínimo possível, visando alto desempenho.

Na Figura 27 é possível observar que o algoritmo de CA's distribuído é uma extensão do algoritmo de CA's paralelo, apresentando um funcionamento similar a este

algoritmo. As principais diferenças deste algoritmo são:

- Os CA's tem sua caminhada restrita ao bloco de dados onde foram criados.
- O controle existente em toda operação de leitura permite identificar quando um conjunto de dados pertence ou não a um bloco. Este controle coordena o acesso aos dados locais e às cópias dos dados pertencentes a blocos vizinhos que se encontram em outros nós do aglomerado de computadores via regiões fantasmas.
- As regiões fantasmas exigem um controle extra para sincronismo e atualização de seus valores.

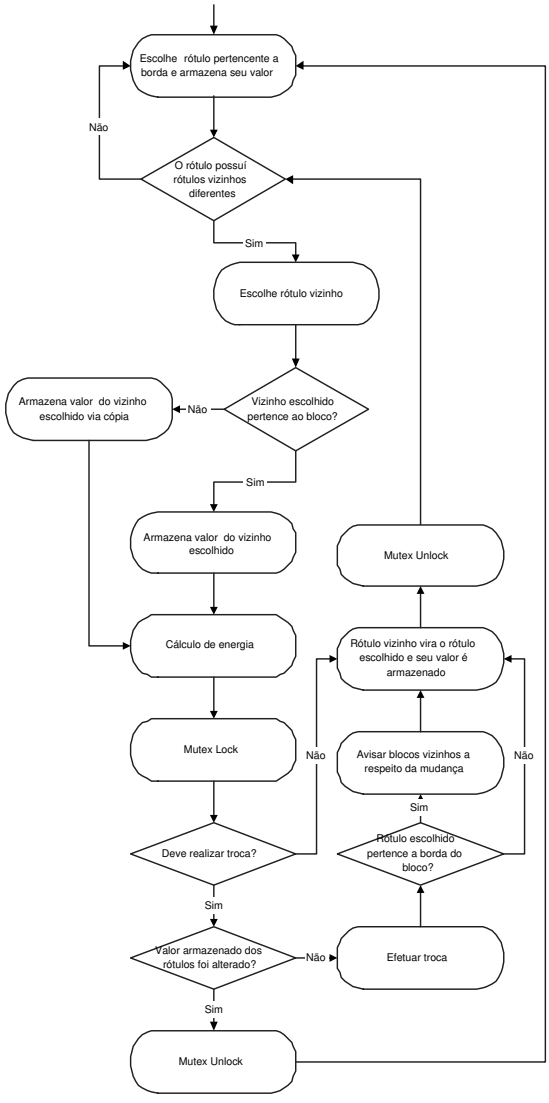


Figura 27: Fluxograma do algoritmo de CA's distribuído.

6.2.2.1 Resultados

Para testes de desempenho desta implementação do algoritmo de CA's foi realizada uma média do resultado de dez simulações envolvendo espumas em 3D. Estes resultados foram obtidos utilizando uma rede de 700^3 rótulos, um mesmo estado inicial contendo 9.620.664 bolhas com um volume médio de 35,65 rótulos. As simulações param quando o sistema obtém 8.000.000 bolhas. Os experimentos foram realizados em um aglomerado de computadores dedicado composto por quatorze nós (dual Xeon 2.8GHz, 2 GB RAM, Gigabit Ethernet).

A Tabela 8 apresenta a comparação do tempo de execução, a relação entre o tempo de execução dos algoritmos seqüenciais (Tabela 7) e o tempo de execução dos algoritmos distribuídos, *speedup* (Sp_1), a relação entre o tempo de execução do algoritmo de Monte Carlo seqüencial e o algoritmo de CA distribuído (Sp_2), e (G1) mostra o ganho de velocidade do algoritmo de CA's em relação ao algoritmo de MC distribuído executando um mesmo número de nós. Este resultado foi obtido sem utilizar o recurso de *multithread*, ou seja, existe apenas uma única *thread* em cada bloco que compõe a rede de dados. O melhor resultado neste caso foi obtido com o uso de 7 nós, que permitiu menor tempo de execução e melhor *speedup*, os demais testes obtiveram resultados inferiores. Da mesma forma que na Tabela 8, a Tabela 9 apresenta os resultados do algoritmo de CA's distribuído porém, neste caso, fazendo uso de recursos de *multithread*. Como se pode observar, os tempos de execução e *speedups* demonstram que o recurso de execução multithread permite obter melhores tempos de execução sobrepondo os tempos gastos com comunicação entre os nós por computação efetiva de dados.

Tabela 7: Tempo de execução seqüencial.

Algoritmo	Tempo (s)	G1
MC	26.295	-
CA	6.431	4,08

Tabela 8: Tempo de execução dos algoritmos Caminhantes Aleatórios e Monte Carlo em um aglomerado de computadores (1 *thread* por nó).

Algoritmo	# nós	Tempo (s)	Sp_1	Sp_2	G1
MC	2	15.611±120	1,68	-	-
	4	8.969±57	2,93	-	-
	7	7.537±120	3,48	-	-
	10	8.255±66	3,18	-	-
	14	14.011±60	1,87	-	-
CA	2	2.717±28	2,36	9,67	5,74
	4	1.586±11	4,05	15,57	5,65
	7	1.185±35	5,42	22,18	6,36
	10	1.255±38	5,12	20,95	6,57
	14	1.997±27	3,22	13,16	7,01

Tabela 9: Tempo de execução do algoritmo CA distribuído em um aglomerado de computadores (2 *threads* por nó).

Algoritmo	# nós	Tempo (s)	Sp_1	Sp_2	G1
CA	2	1.803±13	3,56	14,58	8,65
	4	1.118±17	5,74	23,51	8,02
	7	863±4	7,44	30,46	8,73
	10	764±13	8,41	34,40	10,80
	14	899±10	7,15	29,24	15,58

A Figura 28 apresenta um comparativo entre os algoritmos de CA's paralelo e distribuído, estes resultados foram obtidos a partir da execução da simulação de espumas 3D (Seção 6.1.1). Para este comparativo, os resultados das simulações não estão no estado de escala.

Por fim foi realizada execução de uma simulação com uma rede de $900^2 * 800$ rótulos para testar a capacidade de execução de simulações grandes. A Tabela 10 apresenta a quantidade de memória necessária para isto. O total de memória requerida neste caso é maior que a quantidade de memória disponível em um único nó do aglomerado de computadores utilizado neste experimento, portanto esta simulação só pode ser executada fazendo uso de dois ou mais nós do aglomerado.

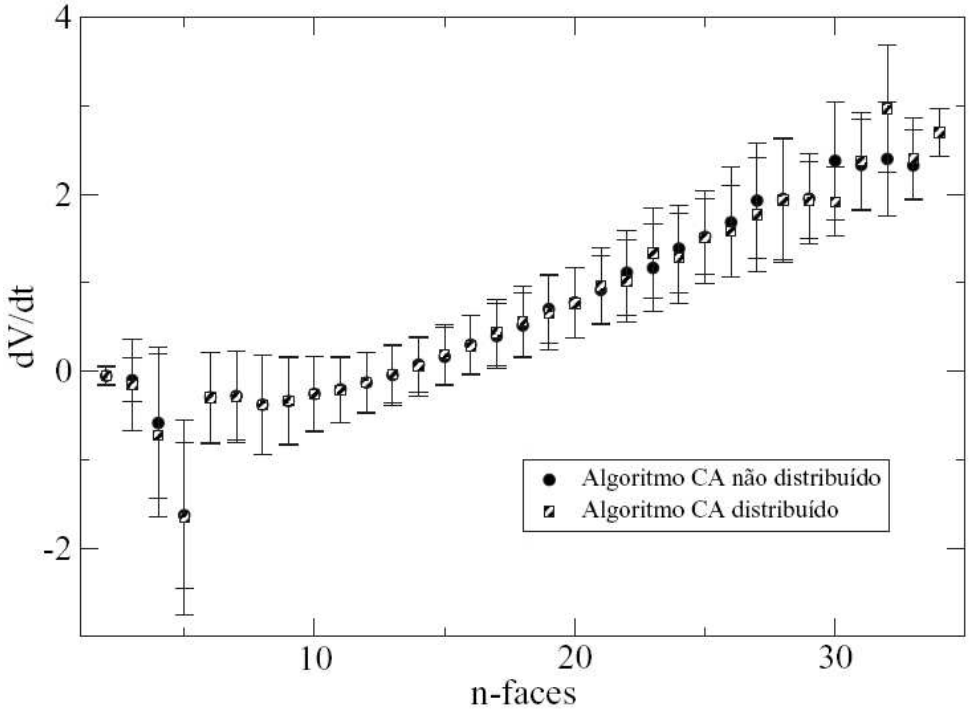


Figura 28: Resultados das simulações de espumas 3d.

Tabela 10: Tempo de execução do algoritmo de CA em um aglomerado de computadores (2 *threads* por nó) com uma rede de dados grande contendo 900^2x800 rótulos.

Algoritmo	# nós	Tempo (s)	Memória Total (MB)	Memória/nó (MB)
CA	4	2.834±53	2.736	792
	10	2.550±33		403

6.3 Discussão dos algoritmos de CA's

Buscou-se acrescentar ao algoritmo de CA's as principais características dos algoritmos apresentados no Capítulo 4, visando melhor desempenho e possibilitando execução de grandes simulações. O algoritmo foi acrescido de uma técnica que busca explorar as características do algoritmo convencional, modificando e criando uma nova técnica para substituir o método de Monte Carlo, de forma similar ao algoritmo N-Fold Way (Seção 4.2), porém sem necessitar armazenar tabelas contendo as características de cada rótulo. Os resultados preliminares utilizados para validar o algoritmo de CA's, não apresentaram nenhuma desvantagem com relação à quantidade de rótulos diferentes na simulação,

como o algoritmo N-Fold Way apresenta (WRIGHT et al., 1997). Além disto, o algoritmo de CA's foi desenvolvido de forma a escolher apenas rótulos com condições para realizar trocas, tal como o algoritmo Masking (Seção 4.3), porém sem fazer uso de variáveis para armazenar a situação de cada rótulo, apresentando melhor desempenho e menor custo de memória para sua execução. O algoritmo de CA's emprega uma técnica de paralelismo similar ao apresentado na Seção 4.1, e obtém-se um maior ganho de desempenho nesta implementação devido às vantagens do algoritmo de CA's (Seções 6.1.1 e 6.2.1.1).

O algoritmo de CA's seqüencial é no mínimo 6 vezes mais rápido que o algoritmo de MC, isto pelo fato de percorrer somente as bordas das células e reduzir ao mínimo o número de sorteios necessários para encontrar posições de borda válidos para execução de caminhadas. Já o algoritmo de CA's paralelo é no mínimo 17% mais rápido que o algoritmo de CA's seqüencial, devido às técnicas de controle de acesso aos dados que permite execução de mais de um CA simultaneamente. A técnica de execução distribuída utilizada pelo algoritmo de CA's distribuído permite a execução de simulações de grandes proporções, devido à segmentação da rede de dados em vários blocos pequenos. Este algoritmo sobrepõe custos de comunicação, necessária para execução em aglomerados, por processamento efetivo através da execução de mais de um CA por bloco de dados. Este algoritmo é mais de 20 vezes mais rápido que o algoritmo de MC seqüencial.

7 CONSIDERAÇÕES FINAIS

O modelo de Potts celular é o modelo computacional estocástico que melhor representa as propriedades físicas das células, através do qual é possível a simulação de diversos tipos de fenômenos, sendo largamente utilizado em simulações envolvendo o crescimento de tumores e espumas de sabão entre outros. Este modelo apresenta um custo computacional que depende diretamente do tamanho da simulação e dimensão utilizada. Foi demonstrado que o método de Monte Carlo, neste caso, consome muito tempo para realizar operações para a evolução de uma simulação.

O algoritmo proposto neste trabalho consiste em uma extensão do algoritmo de Caminhantes Aleatórios e tem por finalidade reduzir o tempo de execução de simulações melhorando o desempenho. Este algoritmo procura manter as características originais das simulações do modelo de Potts. O funcionamento se dá selecionando rótulos pertencentes apenas às bordas das células, desta forma se evolui um sistema de maneira similar ao algoritmo padrão em uma parcela de tempo de execução reduzida. Aliando a este algoritmo a capacidade de execução concorrente *multithread* tornou-se possível explorar os recursos de arquiteturas multiprocessadas, onde cada processador se encarrega de executar um CA. Isto foi possível com uso de *threads* e mecanismos de controle de acesso a dados *mutex*. Por fim, foi apresentada uma forma de distribuir o algoritmo de CA's, tornando possível sua execução em aglomerados de computadores, permitindo a execução de simulações com um tamanho e realismo maiores, além de reduzir o tempo total de execução.

O algoritmo de Caminhantes Aleatórios distribuído foi implementado de forma a explorar concorrência com o objetivo de reduzir o custo gerado pela comunicação, que

afeta o desempenho e aumenta o tempo total de execução de uma simulação qualquer. Os resultados desta implementação demonstraram uma melhora no desempenho do algoritmo, reduzindo o tempo total de execução das simulações realizadas. A execução concorrente possibilitou utilizar os recursos dos nós do aglomerado de forma eficiente, permitindo explorar os recursos de processamento, além de utilizar o mínimo possível os recursos de comunicação. Nesta implementação são executados vários CA's por nó, garantindo que sempre existirá um CA apto a ocupar um processador de um nó mesmo que um ou outro venha a realizar comunicação. O uso de aglomerado de computadores permitiu a execução de simulações de sistemas grandes. Para isto a rede de dados que armazena os rótulos da simulação foi segmentado em vários blocos paralelos a um dos eixos, que são entregues para os nós do aglomerado.

Além da implementação do algoritmo de CA, neste trabalho foi desenvolvido um sistema que tem como finalidade permitir visualizar resultados obtidos nas simulações do modelo de Potts celular, possibilitando validar e observar o correto funcionamento destas simulações. Esta ferramenta foi concebida de forma a tornar possível modificá-la, o que permite visualizar outros tipos de formas em três dimensões além dos resultados obtidos com o modelo de Potts celular.

Até o momento atual, as implementações e descobertas feitas resultaram em dois artigos publicados, e dois pedidos de registro, um de software e um de patente, ambos em parceria com a HP:

- Um sistema para investigação e visualização do crescimento de tumores. V Workshop de Informática Médica (2005);
- An efficient parallel algorithm to evolve simulations of the cellular Potts model. Parallel Processing Letters (2005);
- Registro de patente de software no Estados Unidos (US Patent Office application No: 200406461-2) para o algoritmo de CA's paralelo;

- Registro de software na Inglaterra (GB0517891-8) para o algoritmo de CA's distribuído.

Uma possibilidade de continuação deste trabalho envolve aplicar o algoritmo de CA's em problemas reais. Existe também a necessidade de implementação de um algoritmo para compressão de dados, uma vez que os resultados obtidos por simulações grandes, dependendo da configuração utilizada, podem vir a ocupar um espaço em disco bastante expressivo. Além disso existe a possibilidade de desenvolvimento de uma interface gráfica que englobe o algoritmo de CA's juntamente ao sistema visualizador, pois assim estes programas podem ser utilizados de maneira mais intuitiva, permitindo que mais usuários possam utilizar este simulador.

REFERÊNCIAS

ANDREWS, G. R. *Concurrent Programming: principles and practice*. Menlo Park, CA: L. Olsen, 1991.

ANDREWS, G. R. *Foundations of Multithreaded, Parallel, and Distributed Programming*. Menlo Park, CA: Addison-Wesley, 2000.

BAKER R. BUYYA, D. H. M. Cluster computing: A high-performance contender. *IEEE Computer*, v. 1, n. 1, p. 79–83, 1999.

BARCELLOS, A. M. P.; GASPARY, L. P. Tecnologias de rede para processamento de alto desempenho. In: 3ª *Escola Regional de Alto Desempenho*. Santa Maria - Rio Grande do Sul - Brasil: SBC, 2003. p. 67–102.

BENITEZ, E. D.; CAVALHEIRO, G. G. H. Análise comparativa do uso de MPI e sockets aplicados na convolução de imagens. In: 4ª *Escola Regional de Alto Desempenho*. Pelotas - Rio Grande do Sul - Brasil: SBC, 2004. p. 321–324.

BIRRELL, A.; NELSON, B. Implementing remote procedure calls. *ACM Transactions on Computers Systems*, v. 2, n. 1, p. 39–59, 1984.

BODEN, N. J. et al. Myrinet - a gigabit-per-second local-area-network. *IEEE Micro*, v. 15, n. 1, p. 29–36, 1995.

BORTZ, A. B.; KALOS, M. H.; LEBOWITZ, J. L. A new algorithm for monte carlo simulation of ising spin systems. *Journal of Computational Physics*, v. 17, n. 1, p. 10–18, 1975.

CAVALHEIRO, G. G. H. Introdução à programação paralela e distribuída. In: 1ª *Escola Regional de Alto Desempenho*. Gramado - Rio Grande do Sul - Brasil: SBC, 2001. p. 35–74.

CAVALHEIRO, G. G. H. Princípios da programação concorrente. In: 4ª *Escola Regional de Alto Desempenho*. Pelotas - Rio Grande do Sul - Brasil: SBC, 2004. p. 3–39.

CAVALHEIRO, G. G. H.; DALL'AGNOL, E. C.; VILLA REAL, L. C. Uma biblioteca de processos leves para a implementação de aplicações altamente paralelas. In: *IV Workshop em Sistemas Computacionais de Alto Desempenho*. São Paulo - Brasil: SBC, 2003. p. 117–124.

CERCATO, F. P.; MOMBACH, J. C. M. Um sistema para investigação e visualização do crescimento de tumores. *IN: V Workshop de Informática Médica*, 2005.

COMER, D. E.; STEVENS, D. L. *INTERNETWORKING With TCP/IP Client-Server Programming and Applications*. 4. ed. New Jersey: Prentice Hall, 2001.

CORDEIRO, O. C. et al. Exploiting multithreaded programming on cluster architectures. p. 90–96, 2005.

COSTA, C. M. da; STRINGHINI, D.; CAVALHEIRO, G. G. H. Programação concorrente: Threads, mpi e pvm. In: 2^a *Escola Regional de Alto Desempenho*. São Leopoldo - Rio Grande do Sul - Brasil: SBC, 2002. p. 31–65.

DALL'AGNOL, E. C. et al. Portabilidade na programação para o processamento de alto desempenho. In: *IV Workshop em Sistemas Computacionais de Alto Desempenho*. São Paulo - Brasil: SBC, 2003. p. 141–148.

EICKEN, T. von et al. Active messages: a mechanism for integrated communication and computation. Gold Coast, Australia, v. 20, n. 5, p. 256–266, 1992.

GLAZIER, J. A. Grain growth in three dimensions depends on grain topology. *Physical Rewie Letters*, v. 70, n. 14, p. 2170–2173, april 1993.

GLAZIER, J. A.; WEAIRE, D. The kinetics of cellular patterns. *Journal of Physics*, v. 4, n. 1, p. 1867–1894, 1992.

GOLDMAN, A. Modelos para a computação paralela. In: 3^a *Escola Regional de Alto Desempenho*. Santa Maria - Rio Grande do Sul - Brasil: SBC, 2003. p. 35–66.

GRANER, F. Can surface adhesion drive cell rearrangement? part i: Biological cell-sorting. *Journal of Theoretical Biology*, v. 164, p. 455–476, 1993.

GRANER, F.; GLAZIER, J. A. Simulation of biological cell sorting using a twodimensional extended potts model. *Physical Review Letters*, v. 1, n. 69, p. 2013–2016, 1992.

GUSATTO, E. Uma nova abordagem na implementação do modelo de Potts celular buscando eficiência e explorando paralelismo. Monografia (Graduação) - Universidade do Vale do Rio dos Sinos. 2004.

GUSATTO, E. et al. An efficient parallel algorithm to evolve simulations of the cellular Potts model. *Parallel Processing Letters*, v. 15, n. 1–2, p. 199–208, 2005.

HWANG, K.; XU, Z. *Scalable Parallel Computing*. 1. ed. United States of America: Thomas Casson, 1998.

IZAGUIRRE, J. A.; CHATURVEDI, R.; HUANG, C. Compucell, a multimodel framework for simulations of morphogenesis. *Bioinformatics*, v. 20, p. 1129–1137, 2004.

JIANG, Y. *Cellular Pattern Formation*. Tese (Doutorado) — University of Notre Dame, 1998.

KNEWITZ, M. A. *Um Modelo para Investigação do Crescimento e da Morfologia de Tumores*. Dissertação (Mestrado) — Universidade do Vale do Rio dos Sinos, 2002.

KNEWITZ, M. A.; MOMBACH, J. C. M. Computer simulation of the influence of cellular adhesion on the morphology of the interface between tissues of proliferating and quiescent cells. *Computers in Biology and Medicine*, 2005.

KORNISS, G. et al. Hard simulation problems in the modeling of magnetic materials: Parallelization and langevin micromagnetics. *Computer Simulation Studies in Condensed Matter Physics*, v. 84, p. 98, 1999.

KORNISS, G.; NOVOTNY, M. A.; RIKVOLD, P. A. Parallelization of a dynamic Monte Carlo algorithm: a partially rejection-free conservative approach. *Journal of Computational Physics*, v. 153, p. 488, 1999.

LAW, A. M.; KELTON, W. D. *Simulation Modeling & Analysis*. 2. ed. New York, London: Thomas Casson, 1991.

LUMETTA, S. S.; MAINWARING, A.; CULLER, D. E. Multi-protocol active messages on a cluster of SMPs. In: *Proceedings of Supercomputing'97*. San Jose, CA: ACM SIGARCH and IEEE, 1997. University of California, Berkeley.

MARÉE, F. M.; HOGEWEG, P. How amoeboids self-organize into a fruting body: Multicellular coordination in *Dictyostelium discoideum*. *Proceedings of the National Academy of Sciences*, v. 98, p. 3879–3883, 2001.

MOMBACH, J. C. M. *Um Estudo da Formação de Padrões Celulares Biológicos*. Tese (Doutorado) — Universidade Federal do Rio Grande do Sul, 1997.

NAVAUX, P. O. A. Execução de aplicações em ambientes concorrentes. In: 1^a *Escola Regional de Alto Desempenho*. Gramado - Rio Grande do Sul - Brasil: SBC, 2001. p. 179–193.

NEUMANN, J. von. Metal interfaces. *American Society for Metals*, p. 108, 1952.

NEWMAN, M. E. J.; BARKEMA, G. T. *Monte Carlo Methods in Statistical Physics*. 1. ed. [S.l.]: Oxford University Press, 1999.

NICHOLS, B.; BUTTLAR, D.; FARREL, J. *Pthreads Programming*. 1. ed. Sebastopol, CA: O'Reilly & Associates, 1996.

NOVOTNY, M. A. A tutorial on advanced dynamic monte carlo methods for systems with discrete state spaces. *Annual reviews of computational physics IX*, v. 1, n. 1, p. 153–206, 2001.

PERANCONI, D. S. *Alinhamento de Seqüências Biológicas em Arquiteturas com Memória Distribuída*. Dissertação (Mestrado) — Universidade do Vale do Rio dos Sinos, 2005.

PERANCONI, D. S.; CAVALHEIRO, G. G. H. Using active messages to explore high performance in clusters of computers. *XXV Conferencia Internacional de la Sociedad Chilena de Ciencia de la Computación*, v. 1, n. 1, p. 1–8, 2005.

ROSE, C. A. F. D. et al. The scalable coherent interface (sci) as an alternative for cluster interconnection. *13th Symposium on computer architecture and high performance computing*, v. 1, n. 1, p. 156–163, 2001.

SEBESTA, R. W. *Conceitos de linguagens de programação*. Porto Alegre: Bookman, 2000.

SILBERSCHATZ, A.; GALVIN, P. B. *Operating System Concepts*. New York: John Wiley & Sons, Inc., 1997.

STOTT, E. L. et al. Stochastic simulation of benign avascular tumour growth using the potts model. *Mathematical and Computer Modelling*, v. 30, p. 183–198, 1999.

TANENBAUM, A. S. *Modern Operating Systems*. Englewood Cliff, NJ: Prentice Hall, 1992.

TANENBAUM, A. S. *Scalable Parallel Computing*. 4. ed. Englewood Cliff, NJ: Prentice Hall, 1999.

VAHALIA, U. *UNIX Internals*. Englewood Cliffs: Prentice Hall, 1976.

VALIANT, L. G. A bridging model for parallel computation. *Communications of the ACM*, v. 33, n. 8, p. 103–111, 1990.

VRENIOS, A. *Linux Cluster Architecture*. 1. ed. Indianapolis, Indiana: G. E. Nedeff, 2002.

WALLACH, D. A. et al. Optimistic active messages: a mechanism for scheduling communication with computation. *ACM SIGPLAN Notices*, v. 30, n. 8, p. 217–226, 1995.

WILKINSON, B.; ALLEN, M. *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*. Upper Saddle River, New Jersey: Prentice-Hall, 1999.

WOO, M.; NEIDER, J.; DAVIS, T. *OpenGL Programming Guide*. Reading, MA: Addison Wesley Professional, 1996.

WRIGHT, S. A. et al. Potts-model grain growth simulations: Parallel algorithms and applications. *Sandia Report*, p. 1–47, 1997.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Administração
Baixar livros de Agronomia
Baixar livros de Arquitetura
Baixar livros de Artes
Baixar livros de Astronomia
Baixar livros de Biologia Geral
Baixar livros de Ciência da Computação
Baixar livros de Ciência da Informação
Baixar livros de Ciência Política
Baixar livros de Ciências da Saúde
Baixar livros de Comunicação
Baixar livros do Conselho Nacional de Educação - CNE
Baixar livros de Defesa civil
Baixar livros de Direito
Baixar livros de Direitos humanos
Baixar livros de Economia
Baixar livros de Economia Doméstica
Baixar livros de Educação
Baixar livros de Educação - Trânsito
Baixar livros de Educação Física
Baixar livros de Engenharia Aeroespacial
Baixar livros de Farmácia
Baixar livros de Filosofia
Baixar livros de Física
Baixar livros de Geociências
Baixar livros de Geografia
Baixar livros de História
Baixar livros de Línguas

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo