UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

CARINE P. BEATRICI

Proposta de Implementação em GPU do Modelo de Partículas Auto-propelentes para Segregação Celular

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação

Orientador: Prof. Dr. Marcio Dorn

Porto Alegre
2016

RESUMO

O movimento celular é a base de diversos processos biológicos, por exemplo: segregação celular, cicatrização de tecidos, metástase do câncer, morfogênese, entre outros. Em alguns destes, como no caso da segregação celular, os mecanismos responsáveis pela dinâmica podem ser físicos ou químicos, tornando seu estudo intrinsecamente multidisciplinar. Para estudar as células, pode-se fazer experimentos in vivo ou in vitro, entretanto, nem sempre é possível isolar os fatores em análise, dificultando a interpretação dos resultados. Alternativamente, existem as simulações computacionais para células. Podemos dividir os modelos de simulação computacional de células em dois grandes grupos: i) os modelos em rede: onde as células são representadas por um conjunto de sítios que se movem apenas em posições válidas da rede; ii) modelos fora da rede: as células são representadas por pontos que podem se mover por todo o espaço contínuo. Neste trabalho, vamos analisar um modelos clássico de simulação do movimento celular para o caso da segregação: o modelo de Vicsek. Propomos duas formas de implementação serial do modelo, baseadas no problema da Dinâmica Molecular (DM). Adicionalmente, tratamos da possibilidade de paralelização dos algoritmos referentes a cada uma das duas formas seriais, em específico, para execução em GPUs, para descobrir qual versão, paralela ou serial, possui os menores tempos de processamento.

Palavras-chave: Segregação Celular. Custo computacional. Modelo de Vicsek.

Computational Cost Analysis in Self Propelled Particles for Cell Sorting

ABSTRACT

Cellular movement is the basis of several biologic processes, for example: cell sorting, wound healing, cancer metastases, morphogenesis, and others. In some of them, like cell sorting, the dynamics mechanism are physical or chemical, so this study is essentially multidisciplinary. Cell studies are *in vivo* or *in vitro*, however, it is very hard to isolate the analysis methods and the results interpretation. Computational simulations are another way to study cells. They can be classified in two groups: i) the lattice based models (cells are a pixels set and they only move to valid net positions); ii) the off-lattice models(the cells are dots and they are free to move to all the continuous space). In this work, we analyse a classic cell sorting simulation model: the Vicsek's Model. We propose two serial implementation ways based on the molecular dynamics problem. Aditionally we treat the massive parallel possible algorithms, relative to each serial one, speacially for the GPUs executions. The aim is to find out which version has the smaller execution time.

Keywords: Cell Sorting, Vicsek Model, Computational complexity.

LISTA DE ABREVIATURAS E SIGLAS

HAD	Hipótese da Adesão Diferenciada
HVD	Hipótese da Velocidade Diferenciada
PAP	Partículas Auto-Propelentes
EPS	Exopolissacarídeo
GGH	Glazier Graner e Hogeweg
DM	Dinâmica Molecular
CPU	Unidade Central de Processamento, do inglês *Central Processing Unit*
GPU	Unidade de Processamento Gráfico, do inglês *Graphic Processing Unit*
MP	*Multiprocessors*
SMX	*Next Generation Streaming Multiprocessor*

LISTA DE SÍMBOLOS

$\vec{x}_i$	Vetor posição da i-ésima partícula;
$\vec{v}_i$	Vetor velocidade da i-ésima partícula;
v_i	Módulo da velocidade da i-ésima partícula;
$\hat{\mathbf{x}}$	Versor na direção do eixo x;
$\hat{\mathbf{y}}$	Versor na direção do eixo y;
t	Tempo de processamento em segundos, passo de tempo da simulação;
Δt	Passo de tempo da simulação;
t_f	Tempo final da simulação;
θ_i	Direção de movimento da da i-ésima partícula;
$j \sim$<i>	Apenas j pertencente á vizinhança de i;
r_0	Raio ou distância de vizinhança, alcance da força entre pares de partículas;
r_e	Raio ou distância de equilíbrio;
r_c	Raio ou distância impenetrável, co inglês *core*;
r_{HAR}	Raio ou distância máxima de interação harmônica;
$r_{i,j}$	Distância entre os centros das partículas i e j;
f_0	Valor da representação numérica do $+\infty$ na força;
$f_{i,j}$	Força entre as partículas i e j;
$\alpha_{i,j}$	Tendencia da partícula i seguir a partícula j, pode ser relacionado á viscosidade;
$\beta_{i,j}$	Adesão entre as partículas i e j;
$\vec{u}_i$	Vetor aleatório, relacionado à temperatura, ou flutuação de membrana celular;
L	Tamanho do sistema;
N	Número de partículas do sistema;
N_i	Número de vizinhos da i-ésima partícula;
λ	Expoente da lei de potencia que relaciona o tempo de processamento ao número de partículas do sistema;

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Alguns tipos de células, ao serem dispostas aleatoriamente em uma mistura binária, podem se organizar dentro do agregado em camadas de células diferentes. Este processo de organização é chamado de segregação celular, que é um fenômeno base para diversos processos biológicos importantes (GILBERT, 1997). Por exemplo, durante a morfogênese as células se diferenciam e migram para posições adequadas, onde se formarão as estruturas no indivíduo adulto. Outro exemplo é a cicatrização de tecidos, que envolve migração celular e pode ser estudado com o mesmo ferramental matemático da segregação celular. Os mesmos modelos e técnicas podem ainda ser aplicadas ao estudo da metástase do câncer, onde as células cancerígenas passam a se espalhar para as demais regiões do corpo do individuo.

É possível estudar a segregação celular por diversas formas: desde experimentos biológicos *in vitro*, com modelos matemáticos (descrevendo a evolução de alguma medida da segregação a partir de hipóteses estatísticas), ou mesmo por modelos de simulação computacional (onde elementos físicos ou químicos mínimos, considerados essenciais, são mapeados em representações computacionais de células). Os modelos de simulação envolvem: um modelo matemático, uma hipótese para a segregação celular, o algoritmo que implemente o modelo e o programa de computador da simulação *in silico* propriamente dita. Entraremos em detalhes em cada um destes componentes ao longo dos capítulos seguintes. No momento, destacamos a interdisciplinaridade deste problema, uma vez que o estudo de células está ligado à biologia e demais áreas da saúde, enquanto a hipótese de segregação celular é tipicamente um mecanismo físico ou químico. Ainda temos a adaptação do modelo matemático aos algoritmos e destes aos programas para executar as simulações que são da área da computação.

1.1 Histórico do Problema Biológico

Os estudos com a segregação celular começaram com Wilson (WILSON, 1907), que estudou misturas de células de esponja e observou sua regeneração completa. Entretanto, não era possível diferenciar os tecidos, pois eles são muito parecidos visualmente e a tecnologia de microscopia na época não permitia grande resolução. Dessa forma, não se sabia se a segregação ocorria devido à re-diferenciação celular, ou se as células migravam dentro do agregado até atingir suas posições adequadas. Isto foi esclarecido pelo experi-

mento de Holtfreter (HOLTFRETER, 1944), que pode observar a migração de grupos de células dentro do agregado ao utilizar células de vertebrados, que possuem pigmentação diferenciada por tecido. Ou seja, a segregação ocorre pela migração de células dentro do agregado, indicando a presença de fatores físico-químicos para explicá-la.

Resta então entender qual ou quais são os mecanismos físico-químicos responsáveis pela segregação celular. Vários foram propostos, entre eles a Hipótese da Adesão Diferenciada (HAD) (STEINBERG, 1963) que é a mais difundida para explicar como as células se reorganizam. Nela, as células mais aderentes tendem a ocupar o centro do agregado. Essa é uma hipótese bastante estudada através de experimentos e simulações (STEINBERG, 1963; GRANER, 1992; BELMONTE L. G. BRUNNET, 2008). Há também a Hipótese da Velocidade Diferenciada (HVD), proposta em um trabalho experimental (JONES P. M. EVANS, 1989). Nesta, as células se reorganizam devido à diferença nas motilidades dos diferentes tecidos, as células mais rápidas se encontram mais facilmente formando o tecido central do agregado, enquanto as mais lentas ficam ao redor. Porém, este é um resultado contraintutivo: se pensarmos, por exemplo, em dois líquidos, o resultado esperado seria com o mais energético tendendo a migrar para fora. Os autores propuseram um segundo mecanismo responsável pela origem da segregação, o mecanismo de reconhecimento.

Para decidir qual ou quais as hipóteses são responsáveis pela segregação celular é necessário isolar cada mecanismo e medir os efeitos deles na segregação. A melhor forma de se obter tal isolamento de parâmetros é com uma simulação numérica, pois nela podemos facilmente fixar parâmetros garantindo que as células sejam idênticas exceto por uma única característica, por exemplo, pelas suas motilidades.

1.1.1 Hipóteses da Segregação celular

O experimento de Holtfreter comprovou que a difusão é o agente separador de tecidos, mas não explicou qual ou quais mecanismos são responsáveis pela organização dos tecidos. Surgiram então hipóteses para explicar este problema, sendo as principais:

- Quimiotaxia (TURING, 1952): sugerida por Turing em 1952, essa hipótese defende que as células dos diferentes tecidos se reorganizam de acordo com sinais químicos emitidos do ambiente em torno das células, ou por elas próprias;
- Adesão Diferenciada (STEINBERG, 1963) (HAD): proposta por Steinberg em 1962,

segundo ela, os diferentes tecidos possuem graus diferentes de adesão entre si devido a alguma substância em suas membranas e as células mais adesivas formam um agregado central, com as células do tecido menos adesivo em torno (BELMONTE L. G. BRUNNET, 2008; GRANER, 1992);

- Adesão Temporal (CURTIS, 1961): proposta por Curtis em 1961, esta teoria defende que os padrões de reorganização se devem aos diferentes tempos que as células de tecidos diferentes levam para se aderirem. As células que se aderem mais rapidamente formam um agregado central enquanto as que levam mais tempo ficam na superfície desta.
- Contração Superficial (HARRIS, 1976): sugerida por Harris, essa teoria afirma que quanto menos contrativa é uma célula em relação à sua superfície exposta, mais internamente ela deve se posicionar dentro do agregado;
- Velocidade Diferenciada (JONES P. M. EVANS, 1989) (HVD): esta hipótese foi proposta para explicar o experimento de Jones, Evans e Lee, publicada em 1989, onde atribuiu à reorganização celular a diferença das velocidades de difusão das células dos diferentes tecidos.
- Custo de Coexistência (GESTEL F. J. WEISSING; KOVÁCS, 2014): nesta hipótese as células que secretam um exopolissacarídeo (EPS), células cooperativas, se separam das células que não o produzem, não cooperativas. Este EPS traz benefícios ao grupo, porém tem custo associado a produção. Esta hipótese foi utilizada para descrever a segregação celular em biofilmes bacterianos. Nota-se que esta hipótese se difere das demais, pois ela considera um sistema com células se reproduzindo.

É importante ressaltar que todas essas hipóteses não se excluem mutuamente então, mesmo comprovando que algum destes mecanismos tem participação no processo de separação de tecidos, não podemos descartar os demais. Temos de considerar que todos os mecanismos propostos também podem estar presentes em diferentes seres vivos, pois em espécies de animais diferentes as células podem ter características muito distintas, levando à segregação celular por outros processos físicos. O mais provável é que as hipóteses acima descritas, tenham cada uma sua participação na segregação celular, com proporções diferentes, dependendo também dos tecidos envolvidos na segregação.

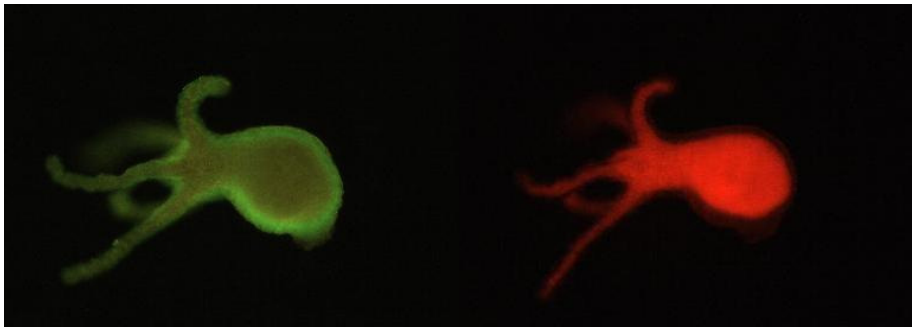
1.2 Experimentos de Segregação Celular

Para estudar a segregação celular experimentalmente é muito comum o uso de hidras (*Hydra vulgaris* ver Figura 1.1), pois as hidras possuem enorme potencial de regeneração. Suas células, quando separadas umas das outras e misturadas aleatoriamente, se reorganizam em um ser completo. A primeira parte desse processo é justamente a segregação celular, onde as células migram dentro do agregado até as posições corretas e, como são apenas dois tecidos, a configuração final é um tecido localizado internamente e outro externamente no agregado. As células das hidras podem ser marcadas com fluoróforos que permitem a identificação das células de cada tecido. Na Figura 1.2 vemos a hidra da Figura 1.1 com as duas cores de fluorescência marcadas.

Figura 1.1: Imagem representando uma hidra. Os tecidos podem ser diferenciados: o mais claro é a ectoderme; enquanto que o mais escuro, por dentro da hidra é a endoderme. A aquisição das imagens foi realizada em um microscópio Olympus IX70 no laboratório de estruturas celulares do IF-UFRGS. A objetiva utilizada é de 4x.

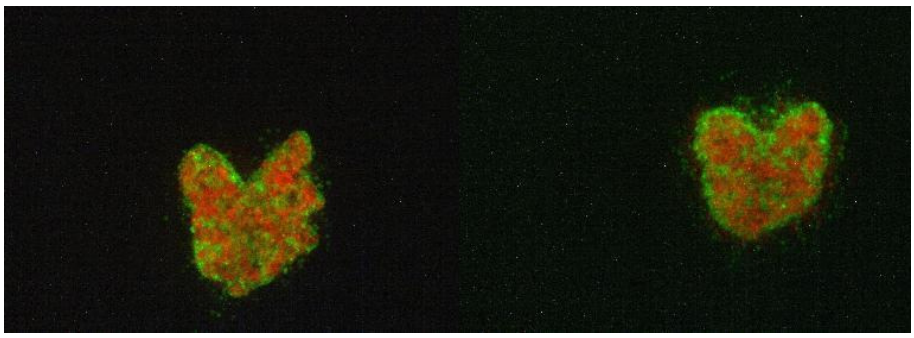
Fonte: a autora.

Figura 1.2: Imagem representando uma hidra. Os tecidos podem ser diferenciados: a ectoderme está marcada em verde; enquanto a endoderme está marcada em vermelho. A aquisição das imagens foi realizada em um microscópio Olympus IX70 no laboratório de estruturas celulares do IF-UFRGS. A objetiva utilizada é da 4x.



Fonte: a autora

Figura 1.3: À esquerda, estado inicial do experimento. À direita, depois de quatro horas de experimento.



Fonte: a autora

Um exemplo de experimento de segregação pode ser visto na Figura 1.3. As células são misturadas e colocadas entre placas formando uma mono-camada celular, que é aproximadamente bidimensional. No experimento mostrado na figura, a hidra ainda não atingiu o estado final de segregação após as quatro horas, pois vemos algumas células de ectoderme (verdes) misturadas às células de endoderme (vermelhas).

As medidas de segregação obtidas neste tipo de experimento tipicamente são: o padrão final de segregação, a interface total entre os tecidos e a área média (ou a distribuição) dos agregados de endoderme. As medidas podem ser comparadas às respectivas obtidas nas simulações computacionais, para determinar qual a participação de cada mecanismo possível na segregação.

1.3 Representação de Células Biológicas no Computador

Existem diversas maneiras de se representar uma célula biológica no computador. Pode-se simular a célula inteira, mas é factível apenas para células simples, como a *M. Genitalium* (KARR J. C. SANGHVI, 2012). Estes modelos computacionais levam em conta a função integrada de cada gene e cada molécula em uma célula. Dessa forma, dependem da disponibilidade de informação, para descrever qualquer tipo de célula. Outra possibilidade é considerar apenas alguns conjuntos de genes que interagem diretamente e estudar um processo de cada vez, entretanto, essa abordagem não pode ser usada para descrever a dinâmica externa da célula.

Para estudar como a célula se move no ambiente, o modelo precisa considerar a célula como um todo. Dessa forma, os detalhes internos da célula podem ser desprezados. Na literatura, temos dois modelos clássicos para simular dinâmica celular: o modelo de

Vicsek (VICSEK A. CZIRÓK, 1995) e o modelo de Glazier Graner e Hogeweg (GGH) (GRANER, 1992), que são a base dos modelos atuais.

O modelo GGH é um modelo baseado em rede, onde uma célula é um conjunto de sítios da rede. Cada sítio possui um conjunto de atributos, como número identificador da célula, qual o tecido que a célula faz parte, etc. As interações ocorrem por trocas locais dos atributos de cada pixel, dependendo de energias globais. Ou seja, sorteia-se dois sítios vizinhos na rede. Calcula-se a energia antes e depois da troca, se a energia total diminuir a troca é aceita, caso contrário a troca ainda pode ser aceita com uma probabilidade proporcional à temperatura. As células se movem sítio por sítio. Os mecanismos geradores de segregação celular podem ser modelados nessa função de energia. A energia pode conter termos variados, como volume celular, adesão e outros.

A alternativa ao modelo GGH é o modelo de Vicsek, que é um modelo fora de rede. Cada partícula é um ponto, que se move com velocidade fixa em módulo, alterando apenas a direção do movimento. Essa direção depende da força aplicada sobre a célula, das velocidades da vizinhança, um termo de ruído, etc. E os mecanismos geradores de segregação podem ser simulados em algum dos termos da direção de velocidade ou na própria velocidade das partículas. Ao contrário do modelo GGH, o modelo de Vicsek é muito versátil, podendo ser aplicado à modelagem de diversos problemas, por exemplo, bandos de animais, grupos de pessoas e até robôs. Por esta versatilidade, este é o modelo adotado neste trabalho e entraremos em detalhes sobre ele no próximo capítulo.

1.4 Objetivos

O modelo de Vicsek pode ser adaptado para diversos tipos de sistema, por exemplo, simulação com bandos de animais (VICSEK A. CZIRÓK, 1995), movimento de células (BELMONTE L. G. BRUNNET, 2008) e qualquer sistema que os corpos se movam ao usar energia interna. Em geral, os trabalhos com o modelo de Vicsek descrevem as equações matemáticas adaptadas para o problema abordado e observam parâmetros de medida de interesse, como expoentes críticos, sem grandes preocupações com o algoritmo e as respectivas implementações. Entretanto, os detalhes de implementação são importantes, uma vez que uma simulação completa de um experimento pode demorar da ordem de meses para executar do estado inicial até o estado final. Isto ocorre em sistemas relativamente pequenos, da ordem de dez mil partículas. Os tempos podem ser ordens de grandeza maiores para cálculos de espaços de fase, que contém até dez milhões de

partículas, tornando o tempo de execução impraticável.

Neste trabalho, realizamos o estudo comparativo de diferentes algoritmos para implementação do modelo de Vicsek. Escolhemos uma adaptação específica do modelo de Vicsek com termo de força entre pares de células, formulada para descrever sistemas celulares, conhecido por modelo de Vicsek-Grégoire. O modelo matemático está descrito na literatura (GRÉGOIRE H. CHATÉ, 2003; BELMONTE L. G. BRUNNET, 2008), contudo não existe uma descrição deste algoritmo para implementar o modelo na literatura.

O principal objetivo deste trabalho é encontrar qual o melhor algoritmo para implementar o modelo de Vicsek com forças, no contexto deste trabalho, melhor significa qual algoritmo apresenta menor tempo de execução para a mesma entrada, com o mesmo nível de acurácia. Para isto, vamos nos basear em algoritmos da literatura para um problema similar: a dinâmica molecular(SCHERER, 2005). Descreveremos dois algoritmos: o todos *versus* todos e o método das caixas, ambos seriais, projetados para execução em CPU. Também descreveremos suas versões massivamente paralelas, para executar em GPU. Determinaremos qual algoritmo tem menor tempo de execução, ou seja, é o mais indicado, para diversos sistemas com número de partículas diferentes.

O caminho usual de trabalho seria procurar os algoritmos e implementações do modelo, para então compara-lás, entretanto, esta não é uma tarefa trivial. Pois, na literatura, as informações disponíveis são relacionadas às equações matemáticas e às medidas físico-químicas do sistemas (VICSEK A. CZIRÓK, 1995; GRÉGOIRE H. CHATÉ, 2003; BELMONTE L. G. BRUNNET, 2008). Pouco, ou nada, se fala sobre tempos de execução, custo computacional, uso de paralelismo, e detalhes de implementação. A solução encontrada para contornar este problema é redefinir os algoritmos com base em um problema similar, a dinâmica molecular. Seguiremos a abordagem do livro "Métodos Computacionais da Física"(SCHERER, 2005), no Capítulo 5 deste livro são descritas formas de se implementar a dinâmica molecular com potencial de Lenard-Jones (LENNARD-JONES, 1931). As implementações massivamente paralelas para GPU também não são descritas para o modelo de Vicsek especificamente. Novamente, nos baseamos nos trabalhos com dinâmica molecular (ANDERSON; LORENZ; TRAVESSET, 2008).

As grandes dificuldades em se trabalhar com o modelo de Vicsek com forças são: o número total de pares de partículas é muito grande para sistemas com muitas partículas; pode-se precisar de sistemas com muitas partículas para alguns tipos de análises físicas; em simulações de segregação celular, o tempo para se atingir o estado final depende do número total de partículas. As forças entre pares de células são de curto alcance, vamos

explorar esse fato nas versões do modelo com caixas. Isto, aliado ao fato do estado do sistema num determinado tempo t depender apenas do estado do sistema tempo anterior $t - \Delta t$, permite o paralelismo no cálculo das forças, velocidades e posições de cada partícula.

1.5 Organização do Trabalho

Neste trabalho, exploramos o modelo matemático de Vicsek do ponto de vista dos diversos algoritmos que podem implementá-lo. Analisamos dois algoritmos: o algoritmo todos *versus* todos e o método das caixas. Cada um deles é implementado em duas formas: serial e massivamente paralela. Dessa forma, analisamos 4 versões diferentes de implementações para o mesmo modelo matemático e as comparamos. O objetivo deste trabalho é descobrir qual versão possui menor tempo de execução para cada tamanho de entrada. E em quais casos o paralelismo em placa aceleradora é vantajoso do ponto de vista de tempo de processamento.

No Capítulo 2, descrevemos o modelo de Vicsek, desde sua formulação original até a adaptação utilizada neste trabalho incluindo os termos de força adaptados para células. Em seguida, no Capítulo 3, introduzimos os algoritmos seriais para implementar o modelo de Gréoire-Vicsek. A primeira versão é a simples conversão das equações para algoritmo serial em processador, sem nenhum tipo de otimização. A segunda versão é ainda serial para processador, agora, com um método com menor quantidade de cálculos de distâncias e forças entre pares de partículas.

Uma vez definidos os algoritmos sequenciais, passamos ao paralelismo, no Capítulo 4 discutimos rapidamente a arquitetura paralela da GPU utilizada, para então, definir as versões massivamente paralela dos algoritmos seriais. As terceira e quarta versões são os algoritmos massivamente paralelos para placas aceleradoras, ou *Graphic Processing Units* (GPUs), das respectivas versões sequenciais.

Em seguida, no Capítulo , apresentamos os experimentos e resultados. Os experimentos consistem no conjunto de simulações executados, a linguagem que os algoritmos foram implementados, juntamente com as configurações do servidor. Listamos os parâmetros escolhidos para as simulações e qual a hipótese de segregação será usada para os testes. Os resultados são os tempos de segregação medidos para diversas quantidades de partículas no sistema, para cada algoritmo. Em duas situações usuais: tamanho do sistema fixo (aumentar o número de partículas significa aumento de densidade), tamanho do

sistema variável (aumentar o número de partículas implica no aumento do tamanho do sistema, mantendo a densidade constante).

Para encerrar, temos o Capítulo 6 com as conclusões e perspectivas futuras do trabalho. Com os resultados, concluímos que o algoritmo mais simples é o mais eficiente para sistemas com poucas partículas. Entretanto, para sistemas com mais partículas as outras versões se tornam melhores. Até os tamanhos analisados, a versão serial com método das caixas é mais rápida do que as versões paralelas. Sugerindo que testes para sistemas com mais partículas são necessários, porém, eles não foram concluídos, ficando como perspectivas. Outra perspectiva, é a implementação de versões paralelas dos algoritmos seriais para processador. Além, de melhorias nos algoritmos massivamente paralelos, como por exemplo, o uso explícito da memória compartilhada da GPU.

2 O MODELO DE PARTÍCULAS AUTO PROPELENTES

As células biológicas são muito complexas, tornando virtualmente impossível criar um modelo matemático que descreva todos os seus componentes internos e a dinâmica externa. Um modelo de célula inteira foi descrito para o caso de uma célula muito simples (KARR J. C. SANGHVI, 2012). Entretanto, para células mais complexas, é necessário utilizar modelos matemáticos simplificados que contenham as principais características das células. Na literatura, existem diversas abordagens para simular a dinâmica das células. Pode-se, por exemplo, estudar a dinâmica interna com equações diferenciais que descrevem as reações químicas intracelulares (DEICHMANN et al., 2014). Outra alternativa é estudar a dinâmica externa da célula, onde o objeto de estudo é a célula como entidade que responde ao ambiente e às vizinhas (VICSEK A. CZIRÓK, 1995; GRANER, 1992). Neste trabalho, estamos interessados em modelos para representar a dinâmica externa da célula.

Neste capítulo, vamos apresentar um breve histórico do modelo de Vicsek, que é um modelo clássico da literatura para representar células e animais, até sua versão final analisada.

2.1 Referencial Teórico do Modelo de PAP

O modelo de animoides, do inglês *boids*, é um modelo clássico de PAPs (Partículas Auto-propelentes). Foi concebido em 1987 com intuito de representar no cinema o movimento de bandos de animais, por exemplo morcegos, pássaros e peixes. No trabalho original de Reynolds (REYNOLDS, 1987), cada elemento (animoide) se movimenta seguindo três regras (ver Figura 2.1):

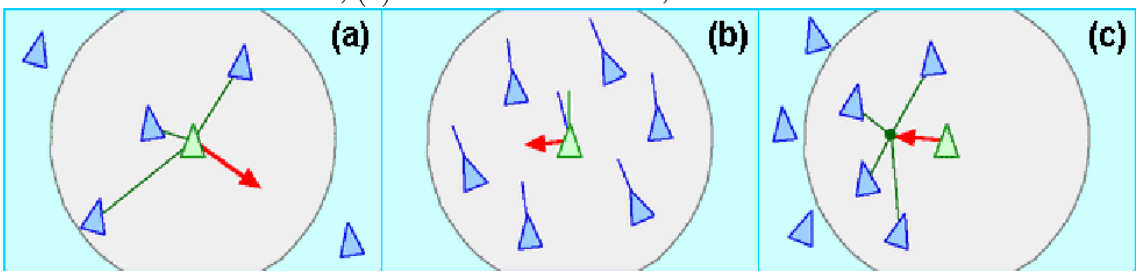
(a) manter uma certa separação dos vizinhos para evitar colisões;

(b) alinhar-se de acordo com o movimento dos vizinhos;

(c) aproximar-se dos demais caso esteja distante do bando.

Formalizando matematicamente essas regras de movimento, incluindo ruído e adaptando-as para dinâmica de células obtém-se um modelo que permite introduzir facilmente diversas hipóteses da segregação celular.

Figura 2.1: Regras de movimentação de animoides. (a) Separação dos animoides; (b) alinhamento com vizinhos; (c) coesão com o bando;



Fonte: (REYNOLDS, 1987)

2.1.1 O Modelo de Vicsek

Somente em 1995, 8 anos após o trabalho de Reynolds, o grupo liderado por T. Vicsek (VICSEK A. CZIRÓK, 1995) fez um estudo sistemático desse modelo. O objetivo era compreender com mais detalhes as condições necessárias para o surgimento do movimento coletivo sem líderes e na ausência de forças externas. Como ocorre, por exemplo, em bandos de pássaros e cardumes.

A essência da formulação de Vicsek é a velocidade dos animoides (nome da partícula) ser constante em módulo, cuja a direção depende das interações com a vizinhança. Assim, em duas dimensões, a posição do i-ésimo animoide no tempo $t + \Delta t$ é dada por:

$$\vec{x}_i(t + \Delta t) = \vec{x}_i(t) + \vec{v}_i(t)\Delta t \tag{2.1}$$

onde a velocidade pode ser escrita como:

$$\vec{v}_i(t) = v_i(t)\cos(\theta_i(t))\hat{\mathbf{x}} + v_i(t)\cos(\theta_i(t))\hat{\mathbf{y}} \tag{2.2}$$

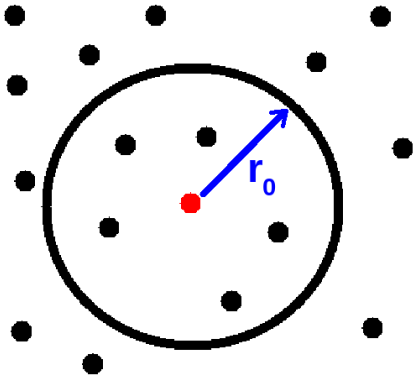
com a direção, θ_i, da velocidade $\vec{v}_i(t + \Delta t)$ do i-ésimo animoide, definida por:

$$\theta_i(t + \Delta t) = \arg\left[\alpha \sum_{j\sim<i>} \vec{v}_j(t) + \eta\vec{u}_i(t)\right] , \tag{2.3}$$

onde $\arg(\vec{v})$ indica que está sendo tomado o ângulo associado ao vetor $\vec{v}$; v_i é o módulo da velocidade da i-ésima partícula; α determina a contribuição à direção de movimento devida aos animoides vizinhos; η determina o peso do ruído, representado por $\vec{u}_i(t)$, um vetor aleatório de módulo unitário. Definimos $\Delta t = 1$ sem perda de generalidade, para manter a estabilidade numérica, a velocidade de cada partícula deve ser menor do que um.

São considerados vizinhos os animoides que estiverem dentro de um raio de alcance r_0 finito, ver Figura2.2. Nota, não estamos usando aqui a mesma notação utilizada por Vicsek em seu artigo original para manter a coerência com a notação adotada neste trabalho.

Figura 2.2: Representação da vizinhança de um animoide, onde o círculo de raio r_0 limita a área de vizinhança do animoide marcado em vermelho.



Fonte: a autora

Assim, a cada passo de tempo, o módulo da posição sempre varia de uma quantidade fixa e o animoide move-se em uma direção definida pela configuração dos vizinhos. Usualmente o sistema é simulado com dimensões fixas, $L \times L$ e com condições de contorno periódicas. Este modelo apresenta resultados interessantes para a dinâmica de aves ou peixes, pois eles raramente colidem. Entretanto este modelo não descreve a força entre os animoides, logo objetos que colidem frequentemente não são bem descritos.

2.1.2 A adaptação de Grégoire

Com o intuito de modelar grupos de animais, Grégoire (GRÉGOIRE H. CHATÉ, 2003) incluiu um termo de força entre as partículas, atribuindo-lhes características não contempladas pelo modelo original de Vicsek. Deve-se mencionar, no entanto, que há uma publicação posterior de Vicsek abordando sistemas celulares
(SZABÓ G. L. SZÖLLÖSKI, 2006). A força proposta por Grégoire possui quatro partes distintas, dependendo da distância entre os centros dos animoides interagentes (logo a seguir a detalharemos). Por ser diretamente derivado do modelo de Vicsek, a velocidade de cada animoide continua possuindo módulo fixo, variando somente sua direção de movimento que será dada pela soma vetorial de três termos: os dois termos originais de Vicsek (tendência a seguir vizinhos e ruído) e um novo termo associado à força de interação entre

animóides. Assim, a expressão para a direção da velocidade do i-ésimo animoide é:

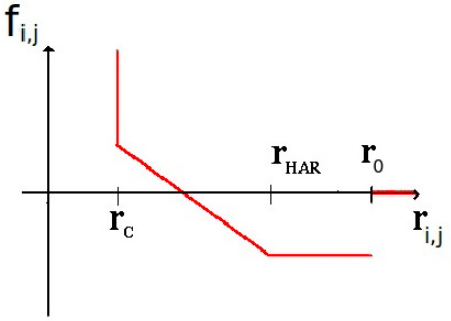
$$\theta_i(t+\Delta t) = \arg\left[\sum_{j\sim<i>} \alpha_{i,j}\vec{v}_j(t) + \sum_j \beta_{i,j}\vec{f}_{ij}(t) + N_i\eta\vec{u}_i(t)\right] \tag{2.4}$$

onde,

- $\alpha_{i,j}$ é o parâmetro de movimento coordenado;
- $\vec{v}_j$ é a velocidade dos vizinhos da i-ésima partícula;
- $\beta_{i,j}$ regula a intensidade da força de contato entre as partículas i e j;
- $\vec{f}_{ij}$ é a força de contato entre as partículas i e j;
- $\vec{u}_i$ é o vetor unitário aleatório, responsável pelo ruído;
- η determina a contribuição relativa do ruído;
- N_i é o número de vizinhos da i-ésima partícula;
- $j \sim< i >$ indica que j deve estar na vizinhança da partícula i;

A definição da vizinhança de um dado boid é a mesma do modelo de Vicsek, as vizinhas são todas as partículas dentro do raio de alcance da força (ver Figura 2.3). A força $\vec{f}_{ij}$ é agora composta por quatro termos: para distâncias menores que um raio r_c é infinita (de repulsão), caracterizando o volume do animoide; para distâncias em torno de r_e é do tipo linear restauradora (o animóide tende a permanecer numa posição de equilíbrio); para distâncias entre o r_{har} e o r_0 é atrativa e constante (o animóide procura não se distanciar do bando); para distâncias maiores que r_0 é nula, o animóide se perdeu do bando.

Figura 2.3: Dependência da força com a distância entre as partículas na adaptação de Grégoire.



Fonte: a autora

$$\vec{f}_{ij} = \vec{e}_{ij} \begin{cases} +\infty & ; r_{ij} \leq r_c \\ 1 - \frac{r_{ij}}{r_e} & ; r_c < r_{ij} < r_{har} \\ 1 - \frac{r_{har}}{r_e} & ; r_{har} \leq r_{ij} < r_0 \\ 0 & ; r_{ij} \geq r_0 \end{cases}$$

onde

- $\vec{e}_{ij}$ é o vetor unitário na direção da partícula i para j;
- r_{ij} é a distância entre as partículas i e j;
- r_e é a distância de equilíbrio entre duas partículas;
- r_0 é a distância de alcance da força;
- r_c é o raio do volume mínimo da partícula, o cerne;
- r_{har} é a distância máxima onde a força ainda está no regime harmônico;

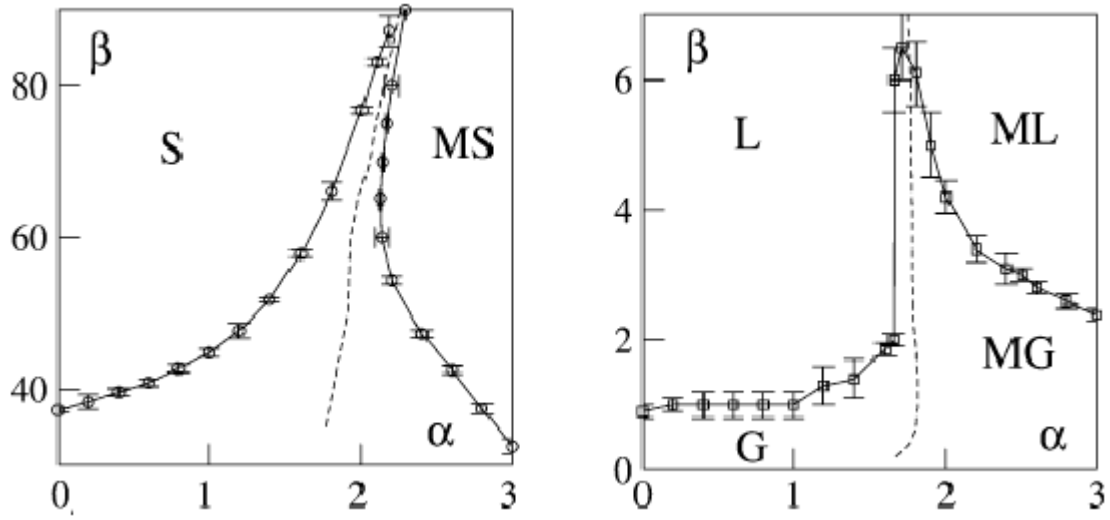
Esta força foi modelada para animais em bandos. As partículas deixam de ser puntuais e, como ocorre no gás de Van der Waals (WAALS, 1910), esperam-se transições para fases condensadas líquida e sólida a baixas temperaturas, ou seja, a baixos níveis de ruído, ou a altas densidades. Além disso, como mostrado no diagrama 2.4, a fase móvel presente no modelo de Vicsek continua existindo, ou seja, este modelo apresenta uma versão móvel correspondendo à cada uma das fases: gás, líquido e sólido.

2.2 Modelo de Animoides para a Segregação Celular

Até este momento mostramos como foi o desenvolvimento e aprimoramento do modelo de animoides, que foi projetado para simular movimento de bandos de animais de forma organizada. A partir de agora descrevemos o modelo final adaptado especificamente ao problema da segregação celular, então mostramos as adaptações feitas no modelo de Grégoire para simular o comportamento dinâmicos de células. Este modelo, juntamente com estas adaptações descritas, foi utilizado para modelar a hipótese da adesão diferencial (HAD) (STEINBERG, 1963; BELMONTE L. G. BRUNNET, 2008).

Para representar as interações de contato de forma realista, a dependência da força com a distância é alterada, passando a ter três termos (ver Figura 2.5): o primeiro representa um cerne impenetrável que tem alcance r_C, responsável pelo volume celular ; o

Figura 2.4: Diagrama de fases para densidade, ρ, não nula e tamanho finito, dimensão igual a 180, $\eta = 1.0$, $\rho = 1/16$, $N = 2025$. Nas figuras vemos as três fases: sólida, líquida e gasosa, representadas respectivamente por: S, L e G, em qualquer das fases o sistema pode possuir movimento coletivo, nesse caso as fases moventes são representadas acrescentando M ao símbolo da fase original. Imagem retirada da ref. (GRÉGOIRE H. CHATÉ, 2003).



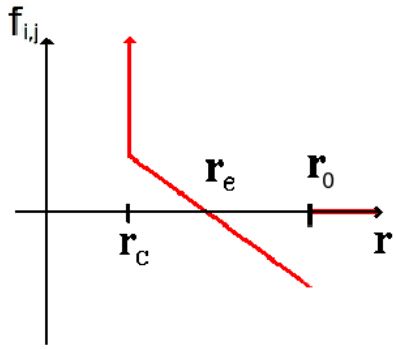
Fonte: (GRÉGOIRE H. CHATÉ, 2003)

segundo é um termo harmônico, representando uma força linear em torno de um raio de equilíbrio r_e e alcance r_0; o terceiro termo é de força nula, definida a partir desse raio de alcance até o infinito. Com esta definição temos uma interação de curto alcance. O regime de força constante não nula não mais existe, os animais podem ver o bando a distâncias maiores porém as células não. A força entre pares de células (BELMONTE L. G. BRUNNET, 2008; BEATRICI; BRUNNET, 2011) é dada por:

$$f_{i,j} = \begin{cases} 0 & ; r_{i,j} \geq r_0 \\ 1 - \frac{r_{i,j}}{r_e} & ; r_c < r_{i,j} < r_0 \\ +\infty & ; r_{i,j} \leq r_c \end{cases}$$

A última adaptação foi inserida na formulação da direção de movimento das partículas, θ, no seu termo de ruído, retirando a dependência deste com o número de vizinhos. Em bandos de animais, o erro que um indivíduo vai cometer ao ver os vizinhos se moverem é proporcional ao número de vizinhos. Mas para células, que estão em contato físico, não há sentido atribuir tal erro, pois este termo estocástico está ligado às flutuações de membrana da própria célula e não a erros de leitura da vizinhança. Dadas estas

Figura 2.5: Dependência da força com a distância entre as partículas no modelo de Belmonte e colaboradores.



Fonte: a autora

alterações, a direção de movimento assume a seguinte forma:

$$\theta_i(t+\Delta t) = \arg\left[\sum_{j<viz>} \alpha_{ij}\frac{\vec{v}_j(t)}{v_0} + \sum_j \beta_{ij}\vec{f}_{ij}(t) + \eta\vec{u}_i(t)\right] \tag{2.5}$$

Para simplificar a notação, ao longo deste trabalho, chamaremos esta versão do modelo de PAP.

Neste capítulo apresentamos um breve histórico do modelo de Vicsek, as adaptações para simular dinâmica de células, o modelo de Grégoire-Vicsek, e finalmente, as equações finais que serão analisadas nos capítulos seguintes. Em seguida, apresentaremos as versões dos algoritmos para CPU que serão implementados.

3 ALGORITMOS PARA CPU

No capítulo anterior, apresentamos o modelo matemático adotado para descrever o movimento de células biológicas. Neste capítulo, mostraremos como implementar o modelo PAP, para realizar os experimentos *in silico*. Trataremos de duas formas distintas de implementação: a) todos com todos e b) método das caixas.

Tipicamente, uma simulação de segregação celular começa com o estado inicial definido aleatoriamente. Isto significa, que em espaços bidimensionais, serão sorteadas $2N$ coordenadas e N ângulos de velocidade. Os valores iniciais para o ângulo da velocidade, $\theta_i(0) \, \forall \, i \leq N \in [0, 2\pi)$, ou seja todas as direção de movimento estão no intervalo entre zero e 2π. As posições estão limitadas pelo tamanho total do sistema, $L \times L$. Cada partícula é totalmente descrita pelo seu índice único, posição no espaço e velocidade. A implementação da estrutura partícula será formada por estes componentes, assim como o módulo da velocidade da partícula.

A dinâmica obedece às equações 2.1 e 2.5, apresentadas no capítulo anterior. O tempo é a variável discreta do problema, como mencionamos anteriormente e ele sempre é incrementado em uma unidade, $\Delta t = 1$. Já as demais variáveis são contínuas e precisam ser recalculadas a cada passo de tempo. Essas características são semelhantes a outra forma de cálculos numéricos, como por exemplo, a dinâmica molecular (DM) (SCHERER, 2005). Para realizar os cálculos da dinâmica diretamente (novas posições e velocidades), precisamos calcular as distâncias e forças entre cada par de células, assim o tempo de processamento cresce com N^2. Mas podemos utilizar técnicas conhecidas da DM para reduzir este tempo otimizando o cálculo das distâncias entre as células.

A implementação do modelo vai ser baseada nos métodos clássicos que Dinâmica Molecular (SCHERER, 2005), pois ambos apresentam similaridades: são sistemas compostos de partículas interagentes, confinados num espaço finito, onde as forças são calculadas a cada passo de tempo, assim como as posições e velocidades, entre outras características comuns. Seguindo o exemplo da DM, o primeiro algoritmo calcula as distâncias entre todas as partículas no sistema, independente da posição relativa entre elas. Esta é a versão mais simples de implementar o modelo, entretanto, é possível evitar muitos cálculos de distância, apenas dividindo o sistema em caixas virtuais. Com esta divisão, calcula-se as distâncias apenas entre as partículas de caixas vizinhas. Este algoritmo é conhecido como o método das caixas e também será abordado (ALLEN; TILDESLEY, 1989).

3.1 Algoritmo todos versus todos

A implementação direta do modelo PAP considera todas as combinações possíveis de pares de partículas para o cálculo das distâncias e forças. A versão mais simples é denominada 1. Esta versão não é muito utilizada na literatura, pois apresenta tempo de processamento muito grande em relação ao método das caixas (SCHERER, 2005) para as quantidades de partículas usuais. Vamos apresentá-la pois a versão correspondente em GPU pode apresentar resultados satisfatórios.

Algorithm 1 Pseudo-código com todas as distâncias calculadas

```
Require: Vetor de estrutura de partículas com N posições
 1: Inicialização das posições e velocidades das partículas;
 2: while t < t_f do
 3:     for i < N do
 4:         for j < N do
 5:             Cálculo das distâncias entre i e j
 6:             Cálculo das forças entre i e j
 7:             Cálculo da média das velocidades dos vizinhos j
 8:         end for
 9:         Atualização da posição e velocidade da i-ésima partícula
10:     end for
11:     if t é múltiplo de 1000 then
12:         Armazena partículas em arquivo
13:     end if
14:     t = t + Δt
15: end while
```

No Algoritmo 1 apresentamos as principais etapas do cálculo da dinâmica de Vicsek-Grégoire. Na Linha 1, as estruturas das partículas são inicializadas, com velocidades aleatórias e posições contidas no espaço do sistema, dessa forma, a inicialização percorre um vetor de N partículas, e executa $3N$ sorteios aleatórios (um para as coordenadas x, o segundo para as coordenadas y e o terceiro para a direção de movimento inicial). Na Linha 2, temos o laço temporal, que percorre os valores inteiros de passo de tempo, desde o tempo inicial, fixado em $t = 0$ até o tempo final, escolhido como $t_f = 10^5$. Na linha 3, percorre-se todas as partículas, pois cada partícula vai sofrer a ação das demais, tanto em termos da força quanto da tendencia a seguir a vizinhança.

Para cada partícula, i, percorremos todas as partículas (Linha 4) e efetuamos os cálculos das distâncias, forças e velocidade média das vizinhas (Linhas 5, 6 e 7). Estes cálculos são implementações diretas das equações do modelo PAP Equações 2.1 e 2.5, que correspondem a parte com maior custo computacional. Depois de calcular todas as forças e velocidades, na Linha 9, faz-se o cálculo do ângulo de movimento de cada partícula e atualiza-se a posição e a velocidade da mesma. Ao longo da dinâmica, a cada mil passos

de tempo, o estado do sistema é armazenado em disco (Linhas 11, 12 e 13) isto é feito para permitir cálculos de parâmetros de interesse da simulação. Quando o tempo final é atingido a memória é liberada e o algoritmo é encerrado.

A complexidade deste algorítimo é $\Theta(N^2)$. Podemos mostrar isso trivialmente, pois a inicialização das partículas é linear com o número de partículas, assim como a atualização das posições e velocidades e o armazenamento em arquivo do estado do sistema. O termo dominante da complexidade é o cálculo das forças e distâncias que escala com o número de pares de partículas do sistema, ou seja, $N(N-1)$, que é de ordem quadrática.

A grande desvantagem deste algoritmo é calcular todas as interações entre as partículas. Isto é necessário para sistemas onde as forças possuem alcance infinito. Entretanto em sistemas com interação de curto alcance, ou seja, a força é nula à partir de uma distância, pode-se evitar isso. Podemos excluir pares que estejam muito afastados, isto é, não sejam vizinhas, sem efetivamente calcular a distância.

3.2 Método das Caixas

O segundo algoritmo explorado foi baseado em métodos de dinâmica molecular (SCHERER, 2005; ALLEN; TILDESLEY, 1989), vamos identificá-lo pelo número 2. A ideia é otimizar o cálculo das distâncias e forças, efetuando apenas os cálculos para partículas que estejam relativamente próximas. Este algoritmo para CPU é o mais utilizado para implementar o PAP, pois apresenta melhor desempenho do que calcular todas as forças (BELMONTE L. G. BRUNNET, 2008; BEATRICI; BRUNNET, 2011; SOLON; CHATÉ; TAILLEUR, 2015; BERTIN; DROZ; GRÉGOIRE, 2009).

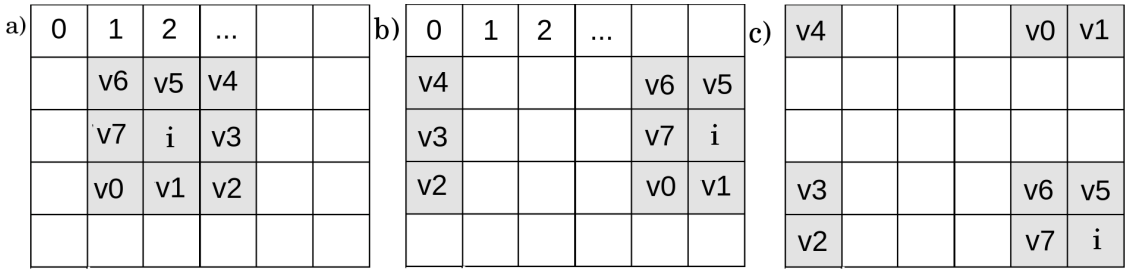
O método das caixas consiste em dividir o espaço em regiões de igual tamanho, denominadas caixas (HOCKNEY; GOEL; EASTWOOD, 1974). Dado que as forças presentes no problema tem curto alcance, não haverá qualquer erro de aproximação se calcularmos as distâncias (e as forças) apenas dentro da mesma caixa ou entre partículas de caixas vizinhas, desde que o tamanho da caixa seja suficientemente grande para conter o alcance da força. No modelo adotado neste trabalho, as forças possuem um raio de alcance, r_0, a partir do qual a força é nula. Atribuindo ao lado das caixas o valor da distância de alcance da força, podemos garantir que todas as vizinhas estarão dentro da mesma caixa ou em caixas vizinhas. Podemos, assim, atribuir diretamente valor zero às forças entre células com distâncias superiores ao lado da caixa.

Para se reduzir os efeitos de borda do sistema e obter propriedades estatísticas com

número finito de partículas, uma tática muito comum advinda da Dinâmica Molecular (DM) é o uso de condições de contorno periódicas. Nessas condições, cada extremidade do espaço é ligada a extremidade oposta. Dessa forma, uma partícula pode se mover em linha reta indefinidamente, sem sair do espaço finito do sistema.

Em duas dimensões, cada caixa possui oito caixas vizinhas. Dado que utilizamos condições de contorno periódicas, mesmo as caixas nos limites espaciais do sistemas possuem oito vizinhas. Pois topologicamente, o sistema é equivalente a um toroide. Denotamos as caixas com números inteiros de 0 ao número de caixas do sistema. A definição da vizinhança é simples, mas devemos ter cuidado nas extremidades do espaço. Nas laterais algumas das caixas vizinhas estarão do lado oposto do sistema, assim como, nas caixas inferiores as vizinhas são as caixas superiores e vice-versa. Nos cantos, são usadas as duas formas anteriores, tendo estes casos separados, as demais vizinhas são facilmente determinadas, podemos ver isso na Figura 3.1.

Figura 3.1: Divisão em caixas do espaço e vizinhas da i-ésima caixa, por exemplo, em três casos: a) caixa i central; b) caixa i lateral, devemos definir a vizinhança das caixas laterais e caixas inferiores; c) caixa i em um dos quatro cantos do sistema, novamente devemos defini-los em separado;



Fonte: a autora

Este algoritmo apresenta menor tempo de execução para os problemas de DM (HOCKNEY; GOEL; EASTWOOD, 1974), o que indica que para o modelo PAP também será mais eficiente. O pseudocódigo da versão 2 do algoritmo é dada por:

O Algoritmo 2 para o PAP com o método das caixas se inicia de forma semelhante ao método anterior. A estrutura da partícula contém as coordenadas e as velocidades, que possuía na versão 1, além disso possui um campo para a caixa a qual a partícula está e o seu identificador. Na Linha 1, temos a inicialização do vetor de estruturas do tipo partícula. Em seguida (Linha 2) inicia-se o laço temporal, onde percorre-se os tempos desde o tempo inicial, ficado em $t_0 = 0$ até o tempo final, $t_f = 10^5$.

Na Linha 6, reordenamos o vetor de partículas, de forma que as partículas sejam

Algorithm 2 Pseudo-código com método das caixas

Require: Dois vetores de estrutura de partículas com N posições, um deles terá as partículas ordenadas.

```
1:  Inicialização das posições e velocidades das partículas;
2:  while t < t_f do
3:      for i < N do
4:          Atribuição das caixa à i-ésima partícula
5:      end for
6:      Ordena o vetor das partículas na ordem das caixas
7:      for caixa < número de caixas do
8:          for início da caixa < i < fim da caixa do
9:              for início das caixas com vizinhas < j < fim das caixas com vizinhas do
10:                 Cálculo das distâncias entre i e j
11:                 Cálculo das forças entre i e j
12:                 Cálculo da média das velocidades dos vizinhos j
13:             end for
14:             Atualização da posição e da velocidade da i-ésima partícula
15:             Calculo das caixas da i-ésima partícula
16:         end for
17:     end for
18:     if t é múltiplo de 1000 then
19:         Armazena partículas em arquivo
20:     end if
21:     t = t + Δt
22: end while
```

ordenadas por suas caixas, não importando a ordem interna à caixa. Percorremos as caixas (Linha 7) este laço se inicia em 0 e se estende pela quantidade de caixas do sistema. A quantidade total de caixas depende apenas do raio de alcance da força e do tamanho do sistema. Caso sejam mantidos constantes, esse valor não depende do tamanho do vetor de partículas.

Nas Linhas 8 e 9, percorremos as partículas de forma semelhante a versão 1 apresentada anteriormente. Nessa versão, contudo, somente as partículas dentro da mesma caixa, ou das caixas vizinhas serão acessadas nos cálculos. Cada partícula tem as posições, velocidades, atualizadas na Linha 14. O cálculo das caixas é feito em seguida, Linha 15. Novamente, a cada mil passos de tempo armazenamos o estado do sistema, para possibilitar o cálculo de parâmetros de interesse da segregação, bem como, produzir imagens do sistema (Linhas 18, 19 e 20). Após o laço de tempo terminar, o programa encerra.

A complexidade deste algoritmo tem as partes lineares em comum com a versão 1, como a alocação e a inicialização das variáveis. O caso pessimista é aquele com todas as partículas em uma caixa. Neste caso, todos os pares serão percorridos e temos o caso $O(N^2)$, como a versão anterior. Nos demais casos, a complexidade será dada pelo algoritmo de ordenação utilizado. A ordenação é feita por caixas, dessa forma, a ordem interna das partículas da mesma caixa não é importante. Isso permite o uso do algoritmo *Counting Sort* (CORMEN et al., 2001), que é linear em complexidade. Este algorítmo já

é linear em complexidade, por isso não precisamos buscar outros algoritmos, como por exemplo, o *radix sorting* e o *bucket sorting*.

Se considerarmos o caso com densidade uniforme e o tamanho do sistema fixo, ou seja, não dependendo do tamanho da entrada, cada caixa terá o mesmo número de partículas. Com isto, o custo computacional associado aos cálculos de distância é $O(N^2)$.

Computacionalmente falando, as simulações tanto de DN quanto do modelo PAP são grandes em dois sentidos: o número de partículas (alguns casos são da ordem de 10^6 partículas(SOLON; CHATÉ; TAILLEUR, 2015)), e o número de passos de tempo (casos com até nove décadas ocorrem (PLIMPTON, 1995; BEATRICI; BRUNNET, 2011; BELMONTE L. G. BRUNNET, 2008)). A possibilidade de paralelismo já é amplamente discutida e implementada na literatura para DM (BOGHOSIAN, 1990; ANDERSON; LORENZ; TRAVESSET, 2008), onde já está comprovado o ganho em tempo de processamento. Ao se programar para GPU, o ganho na performance é equivalente a trinta núcleos de processamento com memória distribuída. Assim, no próximo capítulo iremos apresentar uma proposta de paralelismo massivo para o PAP, baseado em algoritmos paralelos para a DM, para obter ganhos semelhantes aos obtidos com dinâmica molecular.

4 ALGORITMOS MASSIVAMENTE PARALELOS

A parte computacionalmente mais cara de simular o modelo de célula é o cálculo das distâncias e forças, que dependem apenas das coordenadas das partículas no passo tempo anterior. Neste caso, uma boa alternativa para reduzir os tempos de processamento é paralelizar a execução do programa. Com isso, o cálculo da força que cada partícula sofre é realizado em paralelo com as demais.

Algoritmos paralelos em CPU não são muito proveitosos, uma vez que em sistemas de segregação celular várias amostrar são necessárias por experimento. É mais eficiente realizar as simulações das amostras diferentes em paralelo. Neste caso, a melhor alternativa é paralelizar o programa para executar nas placas aceleradoras, ou em GPUs (*Graphic Processing Units*).

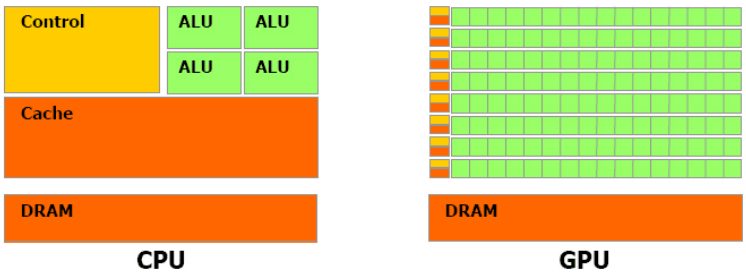
A programação em paralelo exige cuidados especiais, primeiro por que precisamos garantir que não existam duas ou mais escritas na mesma posição da memória ao mesmo tempo, depois as cópias de dados entre as memórias do processador e da placa de vídeo, ou GPU, precisam ser feitas com cautela pois estas são operações demoradas para a GPU. Precisamos então reconstruir todo o algoritmo pensando em melhor desempenho em paralelo.

4.1 Programação Massivamente Paralela

Uma placa aceleradora, ou GPU, possui a habilidade de efetuar significativamente mais cálculos de ponto flutuante por unidade de tempo (FLOPs) do que um processador, ou CPU. Deixando espaço para grandes reduções no tempo de processamento ao usar este tipo de *hardware* (ANDERSON; LORENZ; TRAVESSET, 2008).

A arquitetura de uma GPU é diferente de uma CPU, como pode ser visto na Figura 4.1, enquanto CPUs tem poucas unidades de processamento, uma GPU pode possuir de centenas a milhares. Os detalhes de organização das unidades processamento interna e da memória dependem da arquitetura utilizada. Devido à disponibilidade maior de *hardware*, neste trabalho, utilizamos a arquitetura CUDA. Na arquitetura da Nvidia, cada unidade de processamento de uma GPU é chamada de *Cuda Core*. Essas unidades são dividas em grupos, que possuem o mesmo controle, além da mesma memória compartilhada, esses grupos são chamados de *Multiprocessors* ou (MP). Essa arquitetura é diretamente traduzida para a hierarquia de linhas de processamento, do inglês *threads*.

Figura 4.1: Comparação entre uma CPU típica, com 4 núcleos de processamento, e uma GPU com múltiplos núcleos de processamento. Cada quadrado verde é uma unidade de processamento (*Cuda Core*). As linhas de Cuda Cores possuem a mesma unidade de controle e a mesma memória compartilhada são os *Multiprocessors* (MP). Todos os MP podem acessar a memória global da GPU.



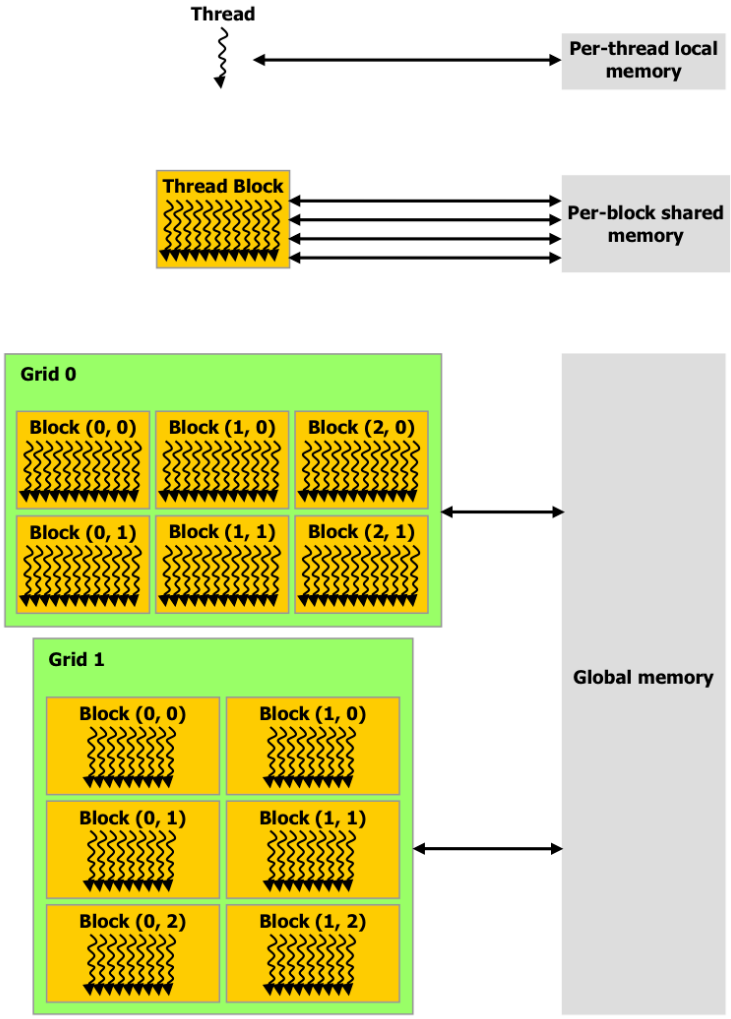
Fonte: (NVIDIA, 2016)

As funções paralelizadas em GPU são chamadas de *kernels*, quando um *kernel* é chamado, ele irá executar em diversas linhas de processamento paralelas (*threads*). Cada linha paralela será executada em um *Cuda Core*. As *threads* são agrupadas em blocos, da mesma forma que os *Cuda Cores* são agrupados nos MPs. E da mesma forma que na arquitetura os blocos possuem a mesma memória compartilhada entre suas *threads*. Ou seja, os blocos residem no mesmo MP, o que resulta em limitações no número máximo de *threads* por bloco. Os blocos podem ser agrupados ainda nas grids, caso a quantidade de dados exceda as capacidades da GPU (NVIDIA, 2016). Essa hierarquia de *threads* é mostrada na Figura 4.2 em conjunto com a memória.

Cada *thread* possui uma memória exclusiva, que apenas ela pode ler, é nessa memória que ficam as variáveis locais de cade *threads*. Os blocos de *threads* tem acesso à memória compartilhada, ou seja, qualquer *thread* do mesmo bloco tem acesso a mesma memória compartilhada. Todas as *threads* podem acessar a memória global, que pode ser acessada pelo processador. Por exemplo, para salvar dados de uma execução em GPU, deve-se: i) copiar os dados das memórias locais para a memória global; ii) copiar os dados para a memória RAM do processador iii) finalmente gravar os dados em disco.

A arquitetura das GPUs evoluiu rapidamente nos últimos anos, existem no mercado diversas opções, como a Maxwell (NVIDIA, 2014a), a Fermi (NVIDIA, 2009) e a Kepler (NVIDIA, 2014b). Neste trabalho, utilizamos um servidor com uma GPU Kepler, a K40. Por este motivo, descrevemos esta arquitetura específica, o objetivo é apenas introduzir os conceitos necessários para a programação massivamente paralela neste tipo de arquitetura.

Figura 4.2: Hierarquia de *threads* e de acesso à memória da GPU. Cada *thread* possui uma memória local que apenas a *thread* pode acessar. O bloco tem acesso à memória compartilhada, do inglês (*shared*). Os *grids* tem acesso à memória global da GPU. Esta memória global pode ser acessada também pela CPU.



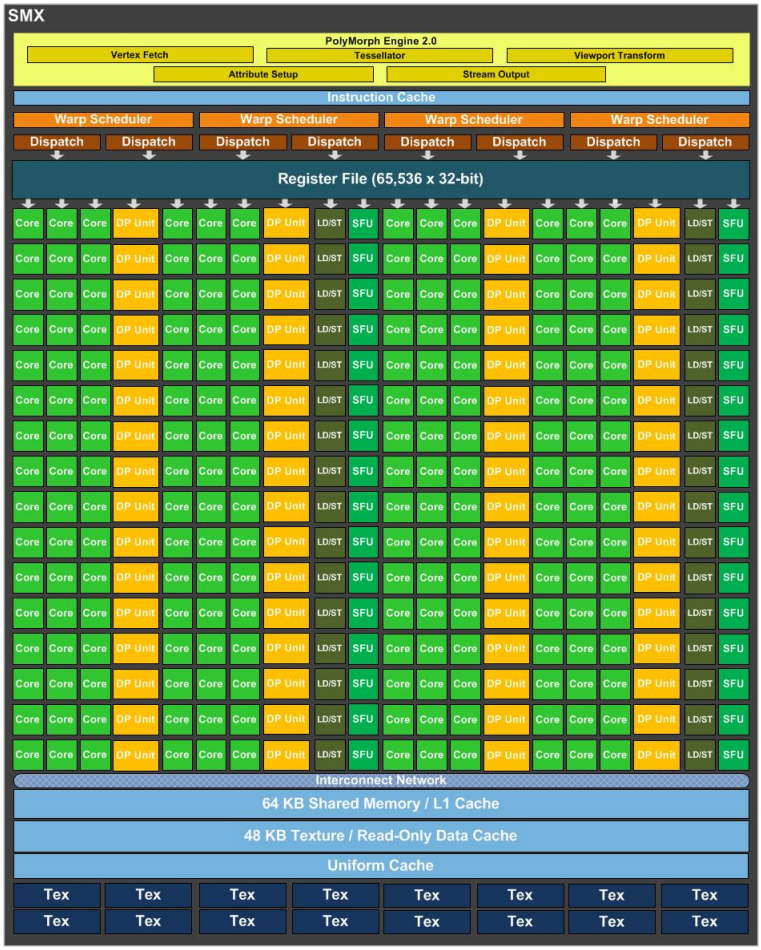
Fonte: (NVIDIA, 2016)

4.1.1 A Arquitetura Kepler

A arquitetura Kepler (NVIDIA, 2014b) trouxe alguns avanços em relação a predecessora, a Fermi (NVIDIA, 2009). Por exemplo, o maior número de *Cuda Cores* por bloco, 192 contra os 32 da Fermi. Entretanto os MP da Kepler possuem outras características diferentes da Fermi, e são chamados de SMX ou do inglês, *Streaming Multiprocessor*. A arquitetura interna do SMX está representada na Figura 4.3, os 192 cores são representados em verde, as 32 unidades de dupla precisão estão em amarelo, também são 32 as

unidades de entrada e saída, em cinza, e em verde azulado, as 32 unidades de funções especiais. Esses valores são importantes na hora de escolhermos as configurações de paralelismo, ou seja, na hora de determinar quantas *threads* e quantos blocos utilizaremos.

Figura 4.3: Na região central de figura, encontram-se as unidades de processamento: em verde, são os *Cuda cores*, em amarelo, as unidades de dupla precisão; em cinza, as unidades de *load store*; em verde azulado, as unidades de processamento especial. Todas essas unidades compartilham um banco de registradores, que se comunica com as unidades de despacho e os warps, que por sua vez se comunicam com a cache de instruções. Os núcleos do SMX também se comunicam com a memória compartilhada (ou *shared*).

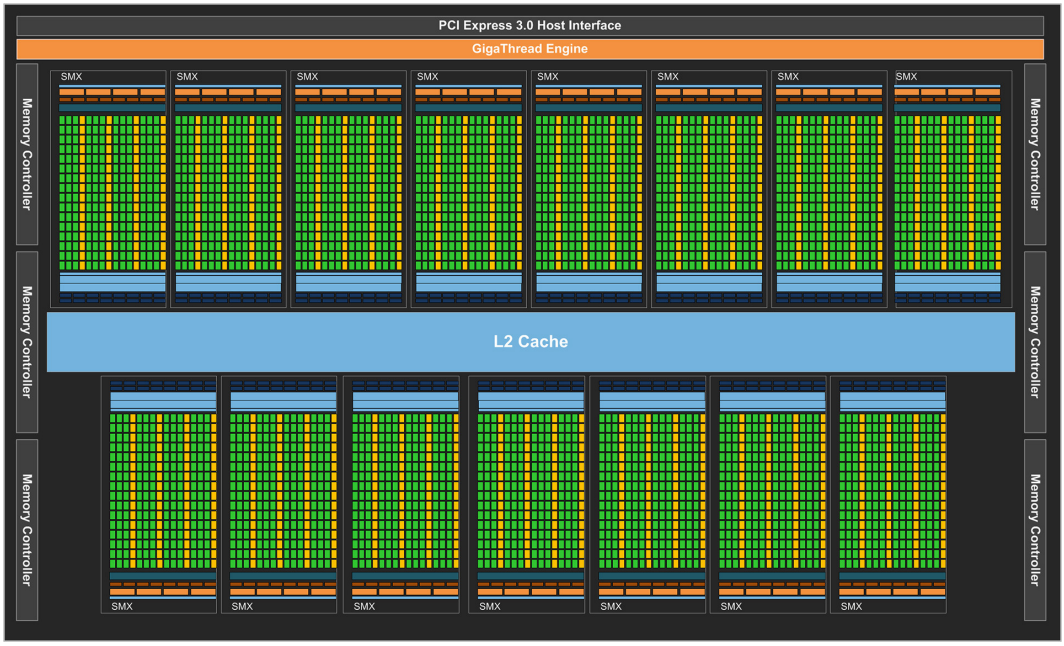


Fonte: (NVIDIA, 2014b)

A representação da arquitetura Kepler pode ser vista na Figura 4.4. Os principais avanços da arquitetura Kepler em relação à antecessora são:

- Paralelismo dinâmico: permite que a GPU gere trabalho para ela mesma, os *kernels* podem chamar outros *kernels* de dentro da GPU com diferentes quantidades de *threads* e de blocos. Na arquitetura anterior a CPU era a única que podia redefinir as quantidades de *threads* e de blocos.
- Unidade de Gerenciamento de Grade, do inglês *Grid Management Unit*: o parale-

Figura 4.4: Todos os SMX têm acesso à memória *cache* L2, assim como a memória global.



Fonte: (NVIDIA, 2014b)

lismo dinâmico requer um sistema de controle e um gerenciamento das grades mais flexíveis. Assim, unidade de gerenciamento de grade que garanta que as cargas de trabalho tanto de GPU quanto de CPU sejam despachadas corretamente.

- *NVIDIA GPUDirect*™: permite que GPUs do mesmo computador, ou de computadores diferentes conectados por rede, se comuniquem entre si, sem precisar passar pela memória principal do computador.
- *Hyper-Q*: várias CPUs podem realizar tarefas dentro de uma GPU ao mesmo tempo, isto diminui a quantidade de cores e o tempo ociosos da GPU;

Nem todas as características desta arquitetura serão relevantes, pois trabalharemos executando apenas um processo de cada vez, em uma única GPU. Entretanto as quantidades de *Cuda cores* e de SMX, ou MPs são fundamentais para determinar os valores ideais de número de *threads* e número de blocos da execução paralela.

Os modelos PAP e a DM são similares por construção, dessa forma, podemos aplicar o mesmo procedimento de paralelismo no PAP que foi aplicado na DM tanto em processadores (PLIMPTON, 1995) quanto em GPUs (ANDERSON; LORENZ; TRAVESSET, 2008). Iniciamos apresentando o algoritmo todos *versus* todos aplicado para GPU do modelo PAP, baseado no trabalho de Anderson (ANDERSON; LORENZ; TRAVESSET, 2008) com dinâmica molecular.

4.2 Método todos versus todos em GPU

Seguindo o trabalho de DM (ANDERSON; LORENZ; TRAVESSET, 2008), vamos implementar a versão massivamente paralela da versão 1 (todos *versus* todos em CPU), que chamaremos de versão 3. O pseudo-código é similar, porém o primeiro laço sobre as partículas não existem mais. Em seu lugar temos o paralelismo implementado:

Algorithm 3 Pseudo-código paralelo com todas as distâncias calculadas

Require: Vetor de estrutura de partículas com N posições
1: Chamada de N linhas paralelas. Paralelismo nas partículas.
2: Inicialização das posições e velocidades das partículas;
3: **while** $t < t_f$ **do**
4: Chamada de N linhas paralelas.
5: **for** j < N **do**
6: Cálculo das distâncias entre i e j
7: Cálculo das forças entre i e j
8: Cálculo da média das velocidades dos vizinhos j
9: **end for**
10: Atualização da posição e velocidade da i-ésima partícula
11: **if** t é múltiplo de 1000 **then**
12: Armazena partículas em arquivo
13: **end if**
14: $t = t + \Delta t$
15: **end while**

O Algoritmo 3 se inicia como os demais, pela atribuição das coordenadas e velocidades às partículas (Linha 1), mas agora, está inicialização ocorre de forma paralela, para um sistema com N partículas, teremos N *threads* realizando os sorteios de forma paralela. Em seguida, na Linha 3, temos o laço temporal exatamente como nas versões seriais, o tempo inicia em $t_0 = 0$ e terminar em $t = 10^5$ passos de tempo. Na Linha 4, temos a chamada do *kernel*, que calcula N *threads* paralelas, as distâncias da partícula com as demais partículas do sistema. Isto não é mais um laço temporal, mas sim uma função paralela. Em cada linha de processamento, Linhas 6, 7 e 8, calculam-se as distâncias, forças e velocidades das demais partículas. Estes cálculos são realizados de forma serial dentro de cada *thread*. Em seguida, na Linha 10, as posições e velocidades das partículas são atualizados. Assim como nos algoritmos para CPU, a cada mil passos de tempo o estado do sistema é armazenado, para cálculos de parâmetros de interesse, Linhas 11, 12 e 13. Por fim, quando o número de passos de tempo final é atingido o programa termina.

A complexidade dos algoritmos em paralelo é a mesma das correspondentes versões seriais, nesse caso quadrática. Dos resultados com DM, esperamos que o tempo de execução seja menor do que o correspondente em CPU, pois haverão mais unidades de processamento fazendo os cálculos ao mesmo tempo.

4.2.1 Algoritmo em GPU com método das caixas

Assim como nos casos seriais, e de acordo com (ANDERSON; LORENZ; TRAVESSET, 2008), exploramos a versão do algoritmo com a divisão em caixas para arquitetura massivamente paralela. Novamente, seguindo os resultados para o modelo exemplo da dinâmica molecular, esperamos que a economia no tempo de processamento das distâncias compense o tempo gasto para atribuição das caixas e ordenação do vetor de partículas. O algoritmo paralelo, chamado versão 4 é dado por:

Algorithm 4 Pseudo-código paralelo com método das caixas

Require: Dois vetores de estrutura de partículas com N posições, um deles terá as partículas ordenadas.

1: Chamada de N linhas paralelas. Paralelismo nas partículas.
2: Inicialização das posições e velocidades das partículas;
3: **while** $t < t_f$ **do**
4: Chamada de N linhas paralelas.
5: Atribuição das caixa à i-ésima partícula
6: Chamada de número de caixas linhas paralelas.
7: Ordena o vetor das partículas na ordem das caixas
8: Chamada de número de caixas linhas paralelas. Paralelismo em caixas
9: **for** inicio da caixa < i < fim da caixa **do**
10: **for** inicio das caixas com vizinhas < j < fim das caixas com vizinhas **do**
11: Cálculo das distâncias entre i e j
12: Cálculo das forças entre i e j
13: Cálculo da média das velocidades dos vizinhos j
14: **end for**
15: Atualização da posição e da velocidade da i-ésima partícula
16: **end for**
17: **if** t é múltiplo de 1000 **then**
18: Armazena partículas em arquivo
19: **end if**
20: $t = t + \Delta t$
21: **end while**

O Algoritmo 4 se inicia de forma similar ao anterior, na Linha 1 temos a chamada do *kernel* que inicializa as partículas de forma paralela. Com N *threads* paralelas. Como na respectiva versão serial, as estruturas das partículas possuem um indice de identificação e o valor da caixa que a pertícula se encontra, além das posições e velocidades. Em seguida, Linha 3, inicia-se o loop no tempo, de $t_0 = 0$ à $t_f = 10^5$. O cálculo das caixas é feito de forma paralela, na Linha 5, com N linhas de processamento paralelas. Na Linha 7, temos a ordenação do vetor de partículas de forma paralela, no entanto este paralelismo, é mais diretamente aplicável as caixas, então, temos o número total de caixas *threads paralelas* e não com N *threads* paralelas.

O cálculo das distâncias e das forças serão paralelizados em caixas (Linha 8), isto é uma escolha para simplificar o controle dos dados em memória, poderíamos ter feito

o paralelismo em partículas da mesma forma. Dentro de cada *thread* são calculadas as forças que atuam sobre as partículas de uma caixa, a origem dessas forças podem ser as partículas da mesma caixas (laço da Linha 9), ou das partículas de caixas vizinhas (laço da linha 10). Os cálculos de distâncias, forças e velocidade são realizados nas Linhas 11, 12 e 13. Para então, na Linha 14, atualizarmos de forma paralela as posições e velocidades das partículas, novamente o paralelismo é feito com N *threads* paralelas.

Neste capítulo, apresentamos alguns conceitos de programação massivamente paralela. Mostramos a arquitetura da GPU do servidor utilizado, assim como descrevemos os principais avanços desta arquitetura em relação a sua antecessora. Definimos os algoritmos paralelos, baseados no trabalho com dinâmica molecular (ANDERSON; LORENZ; TRAVESSET, 2008). No próximo capítulo, vamos realizar os testes de simulação com as quatro versões de algoritmos apresentadas, para o modelo PAP. Vamos medir os tempos de processamento para diversos tamanhos de sistema, para saber se o PAP realmente se comporta de forma similar a DM em tempo de processamento.

5 EXPERIMENTOS E RESULTADOS

Para testar qual algoritmo possui menor tempo de execução em cada situação implementamos as quatro versões: para os testes seriais em CPU implementamos o algoritmo na linguagem C. Dentre as diversas linguagens de programação para GPU, escolhemos o CUDA *Compute Unified Device Architecture*, pois a linguagem pertence aos fabricantes da placa que temos acesso no servidor.

Todos os testes foram executados na mesma máquina, cujas configurações estão dispostas na Tabela 5.1.

Tabela 5.1: Configurações do servidor

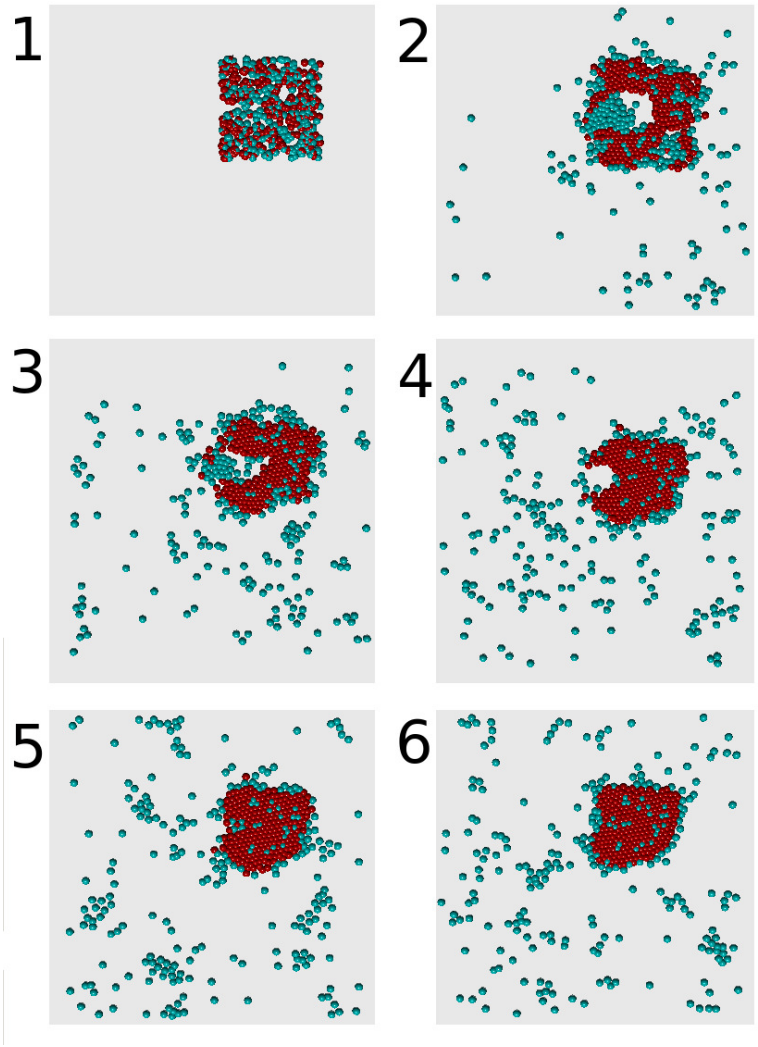
Característica	*Especificação*
Processador:	
Modelo	Intel®Xeon®E5-2407
Frequência	2.20GHz
Cache	10240kB
Memória:	32GB
GPU:	
Modelo:	Tesla K40
Memória global	12GB
Memória compartilhada	49152bit
Cuda Cores:	
Multiprocessors	15
Cuda Cores por MP	192
Total	2880
Max. threads por bloco	1024

Fonte: a autora

A previsão da complexidade assintótica é verificada na primeira parte dos testes. Essa verificação é importante para determinar qual o tamanho do sistema é suficientemente grande para a análise posterior de comparar os algoritmos entre si. Em seguida, comparamos as versões entre si para dois casos: sistema pequeno, $N = 40$, e sistema grande, $N = 1000$. Um terceiro caso de teste está sendo desenvolvido, infelizmente os testes não estão completos para sistema muito grande, $N = 10000$.

5.1 Parâmetros das Simulações

Figura 5.1: Exemplo típico de segregação celular ao longo do tempo. Para sistema com $N = 500$ partículas. Para o sistema segregar as partículas possuem dois tipos: i) vermelhas: são partículas mais lentas; ii) ciano são 4 vezes mais rápida do que as vermelhas. 1 Estado inicial; 2 Snapshot para $t = 10000$. 3 Snapshot para $t = 20000$. 4 Snapshot para $t = 30000$. 5 Snapshot para $t = 40000$. 6 Snapshot para $t = 50000$ que já é o estado de saturação da segregação celular.



Fonte: a autora

As simulações serão todas realizadas em espaços bidimensionais. O sistema é um quadrado de lado $L = 15$, este tamanho será fixo e independente do número de partículas do sistema. Esta informação é particularmente importante nos casos com caixas, pois nestes, o número total de caixas pode afetar os tempos de execução do programa.

A condição inicial é aleatória com as partículas dentro de um quadrado, com lado

igual a um terço o lado do sistema $L/3$. Este quadrado não está no centro do sistema, ou seja, os valores iniciais das coordenadas vão de $L/2$ à $5L/6$. Como mencionado na apresentação do método das caixas, o sistema apresenta condição de contorno periódica, isto significa que se uma partícula ultrapassar alguma margem do sistema, ela será transportada para o lado interno da margem oposta.

A força entre pares de partículas possui um regime infinito responsável pelo volume das partículas. O valor dessa distância é o raio de *core* que foi fixado em $r_c = 0.2$. Entretanto não é possível representar um número infinito no computador, utilizamos o valor $f_0 = 1000$, pois este valor é muito maior do que as demais forças envolvidas no sistema. Os demais regimes de força são delimitados por distâncias. A distância de alcance da força é $r_0 = 0.55$, a distância de equilíbrio é $r_e = 0.4$.

Para que haja segregação nos testes, utilizamos uma hipótese de segregação celular comprovada, a hipótese da velocidade diferenciada (BEATRICI; BRUNNET, 2011; JONES P. M. EVANS, 1989). Com isso, teremos dois tipos de partículas, uma mais lenta, com velocidade $v_0 = 0.007$ que será representada pela cor vermelha, a outra mais rápida, com velocidade $v_2 = 0.028$ representada pela cor ciano. A quantidade de partículas de cada tipo é a mesma em todas as simulações.

Em simulações de segregação celular, o número de passos de tempo necessários para atingir o estado de saturação varia com o número total de partículas. Isso é fundamental para obter as medidas de segregação celular ao longo de todo o processo, entretanto, a computação envolvida é a mesma. Ou seja, do ponto de vista computacional, o tempo de execução dos passos iniciais é o mesmo dos passos finais. Não havendo a necessidade de acompanharmos o processo todo de segregação. Então, interrompemos a execução do programa em $t_f = 10^5$ passos de tempo para evitar longos tempos de simulação.

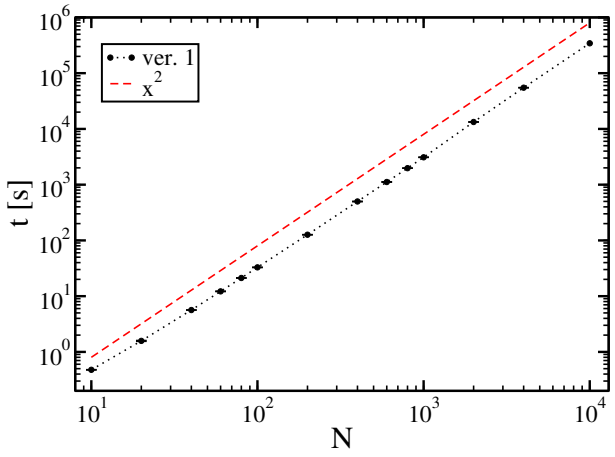
As medidas de segregação celular podem ser obtidas em pós processamento, partindo dos estados do sistema armazenados em disco. O intervalo de passos de tempo entre estados pode variar, pois ele depende do tipo de estudo e da hipóteses de segregação analisadas. Neste trabalho, vamos armazenar o estado do sistema a cada mil passos de tempo.

5.2 Simulações e testes em CPU

A primeira parte das simulações consiste em executar o programa em C, das versões 1 e 2, para diferentes números totais de partículas no sistema. Os tempos de CPU

foram tomados com auxilio da biblioteca time.h do C. Os números totais de partícula medidos foram: $N = \{10, 20, 40, 60, 80, 100, 200, 400, 600, 800, 1000, 2000, 4000, 8000, 10000, 20000\}$ para cada valor de N, executamos o programa 5 vezes para melhorar a estatística.

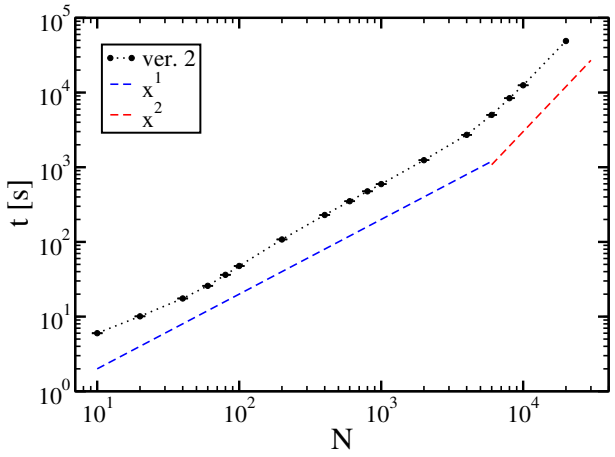
Figura 5.2: Tempo médio de execução da simulação da segregação celular para um sistema com N partículas. Os pontos pretos são as médias em 5 amostras, com os desvios representados pelos traços horizontais. A reta tracejada vermelha indica o expoente da lei de potencia que relaciona o t e N. Com $t \propto N^{\lambda}$ e $\lambda = 2$. Como calculado no capítulo anterior.



Fonte: a autora

Os tempos de execução médios estão dispostos na Figura 5.2. Os pontos formam uma reta com inclinação 2 no gráfico log-log. Indicando que a função que relaciona os tempos de execução com o tamanho da entrada é uma lei de potencia da forma: $t(N) \propto N^{\lambda}$, com $\lambda = 2$. Este resultado já era esperado dos cálculos de complexidade assintótica do capítulo anterior.

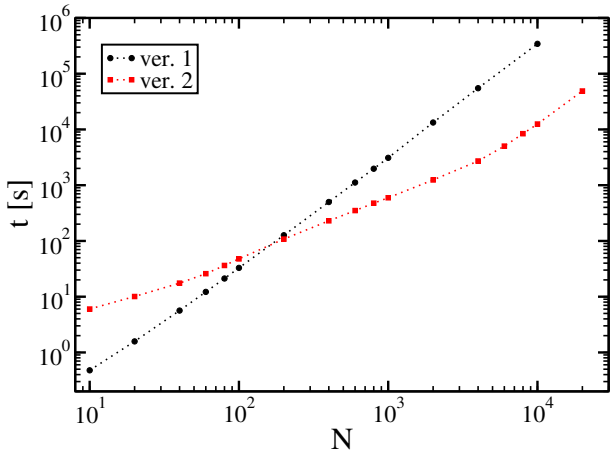
Figura 5.3: Tempo médio de execução da simulação da segregação celular para um sistema com N partículas. Os pontos pretos são as médias em 5 amostras, com os desvios representados pelos traços horizontais. As retas tracejadas são leis de potência para guiar os olhos. A curva tracejada azul representa o regime linear, enquanto a vermelha indica o regime quadrático.



Fonte: a autora

Os resultados da segunda versão são apresentados na Figura 5.3. Para sistemas com poucas partículas, $N < 6000$ o tempo de processamento cresce linearmente com o número de partículas. Conforme a densidade aumenta, o tempo de processamento passa a crescer quadraticamente com o número de partículas.

Figura 5.4: Tempo médio de execução da simulação da segregação celular para um sistema com N partículas. Os pontos pretos são as médias em 5 amostras, com os desvios representados pelos traços horizontais.



Fonte: a autora

Na Figura 5.4 vemos os dados para as duas versões do algoritmo em CPU. Para sistemas pequenos, até $N = 200$ partículas, o método das caixas não é vantajoso. A

criação das variáveis auxiliares e reordenar o vetor tem custo computacional maior do que os cálculos de distância desnecessários. Com isso, o algoritmo mais simples é o mais eficiente. A situação rapidamente se inverte, à medida que aumentamos o número de partículas, os cálculos de distância se tornam mais importantes em relação às variáveis auxiliares, tornando o método das caixas mais eficiente.

5.3 Simulações e Testes em GPU

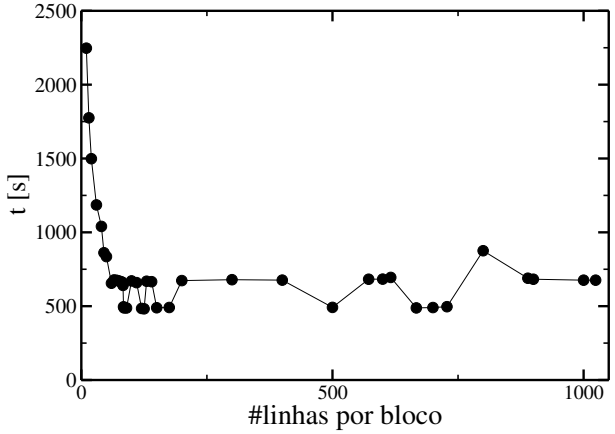
Nas execuções paralelas, temos N linhas de processamento, do inglês *threads*, que fazem cálculos ao mesmo tempo. O tempo de processamento total vai depender fortemente de quantas linhas paralelas temos e também de como elas estão dispostas na arquitetura da GPU utilizada. Então, é muito importante levar em conta qual a organização interna da GPU para decidir como serão agrupadas as linhas de processamento.

A placa aceleradora do servidor utilizado para executar os testes é uma Tesla K40 da empresa Nvidia®. Implementamos os algoritmos paralelos com uma linguagem desta mesma empresa, o CUDA. Não vamos nos aprofundar nas técnicas de programação paralela, tão pouco nos detalhes da arquitetura Kepler. Mesmo assim alguns pontos precisam ser discutidos para a compreensão dos resultados.

As funções paralelas no cuda são chamadas de *kernel*, elas são marcadas por identificadores, como __global__ e __device__. O paralelismo do *kernel* é especificado na chamada da função, através de dois parâmetros: a quantidade de blocos e a quantidade de linhas por bloco. Estes valores podem ser multidimensionais e estão intimamente relacionados com a arquitetura da placa. Pois cada linha é processada em um CudaCore e os blocos são divididos entre os *multiprocessors* da GPU (NVIDIA, 2014b).

Para se otimizar o tempo de execução procuramos quais os valores de blocos e linhas que evitem núcleos de processamento ociosos. Além disso, tomamos cuidado para que a quantidade de linhas paralelas inclua todas as partículas e não tenham linhas sem uma partícula associada. Para aproveitar o máximo dos CudaCores disponíveis, atribuímos que o número de linhas é igual ao número de partículas dividido pelo número de blocos (que será tipicamente o número de multiprocessors da placa). Em alguns casos, há mais partículas do que CudaCores disponíveis e mais blocos serão necessários, nesses casos o número de blocos é um múltiplo inteiro da quantidade de multiprocessors disponível.

Figura 5.5: Tempo de execução para $N = 40000$ partículas. Para poucas linhas por bloco temos um tempo muito grande para a execução, pois grande parte dos *Cuda cores* estão ociosos. À medida que o número de linhas por blocos aumenta o tempo de execução diminui até atingir um patamar assintótico. Para valores maiores do que 100 não há mais variação significativa no tempo total de execução.



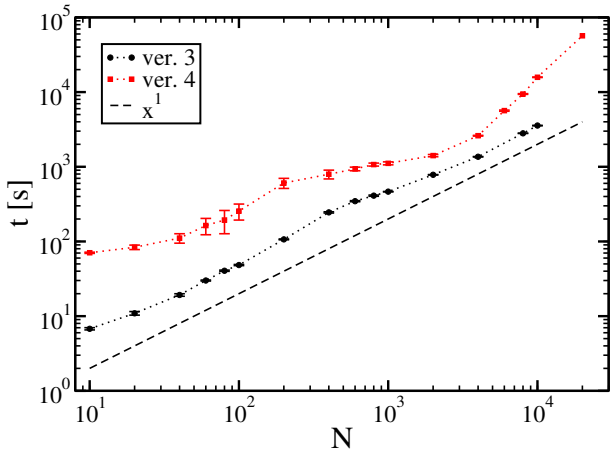
Fonte: a autora

O primeiro teste contém $N = 40000$ partículas para procurar quais valores do número de linhas por bloco gera o menor tempo de execução. Como o total de partículas é fixo, o número de blocos é determinado diretamente pelo número de linhas por bloco. Na Figura 5.5. Na figura mostramos apenas as execuções com número total de linhas paralelas aproximadamente igual ao número de partículas. Caso haja poucas linhas paralelas o programa não executaria corretamente, caso haja um excesso de linhas, o programa executa corretamente, entretanto nunca em menor tempo do que a versão com número correto de linhas. Esse teste mostra, que para mais do que 100 linhas por bloco não há diferença significativa no tempo de execução. As diferenças locais que existem são pequenas e podem desaparecer com maior estatística nos pontos, nessa curva temos apenas uma amostra por ponto.

Executamos os testes para GPU dos algoritmos 3 e 4. Para cada um deles o paralelismo foi pensado para maximizar o número de linhas paralelas por blocos. E o número de blocos é sempre o menor múltiplo do número de multiprocessors da placa que contenha todas as partículas. Essa escolha foi baseada nos resultados do teste anterior. Executamos os algoritmos para tempos $t_f = 10^5$ como nos algoritmos para CPU e comparamos os resultados. Os tempos de execução para as versões paralelas, todos *versus* todos (versão 3) e com métodos das caixas (versão 4) são mostrados na Figura 5.6. As curvas se aproximam de uma reta indicando que ainda estamos longe do regime de complexidade quadrático calculado. Ao contrário do que acontecia para a CPU, o método das caixas

tem maior custo computacional do que calcular todas as interações, para todos os valores testados. Uma inversão ainda é possível para valores maiores de N.

Figura 5.6: Tempo de execução para as versões 3 e 4. Os pontos indicam os valores médios para cinco amostras, a linha tracejada, indica o regime linear dos dados. A complexidade assintótica dos algoritmos é quadrática, mas com $N = 10^4$ isto não é claro.



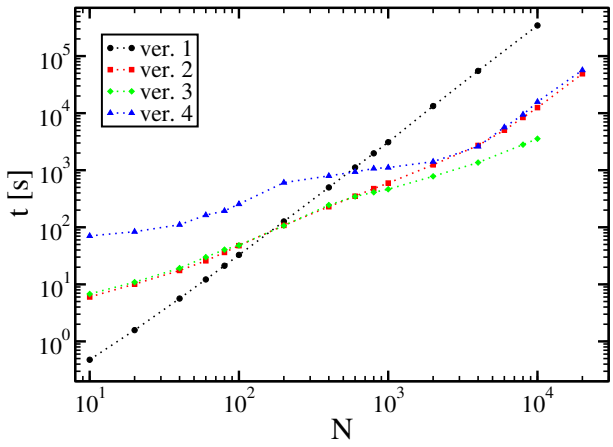
Fonte: a autora

5.4 Comparação entre Algoritmos Seriais e Paralelos

Para finalizar, mostramos um comparativo entre as quatro versões de algoritmos para simular a segregação celular. Este gráfico corresponde aos dados de tempo de processamento contra o número de partículas do sistema apresentados nas sessões anteriores. A Figura 5.7 mostra os tempos médios de execução para cinco amostras, sem as barras de erro para melhor visualização. Na figura, fica claro que o algoritmo para CPU com todas as interações calculadas é mais eficiente para um número pequeno de partículas. À medida que o número de partículas cresce a situação se inverte, e os outros métodos ficam mais eficientes. Ao paralelizarmos os cálculos, temos ganho no tempo de processamento. A versão em GPU com todas as distâncias calculadas é a mais rápida para sistemas com mais de mil partículas. O resultado mais surpreendente é que a versão paralela do método das caixas (versão 4) não é melhor do que a alternativa em GPU, também perde para a correspondente em CPU. Para sistemas grandes, a versão em GPU se iguala em tempo de processamento à respectiva versão em CPU. Isto se deve a grande quantidade de variáveis auxiliares necessárias para permitir o paralelismo em caixas, assim como o cálculo serial para definir quais trechos do vetor de partículas é necessário em cada bloco. Outro fator limitante é a paralelização do cálculo das distâncias entre as caixas, ou seja, cada linha de

processamento paralela é responsável pelo cálculo das distâncias das partículas de uma caixa. Se alterarmos este paralelismo para cada linha ser responsável por uma partícula, teríamos mais linhas paralelas, aumentando o uso relativo da placa aceleradora.

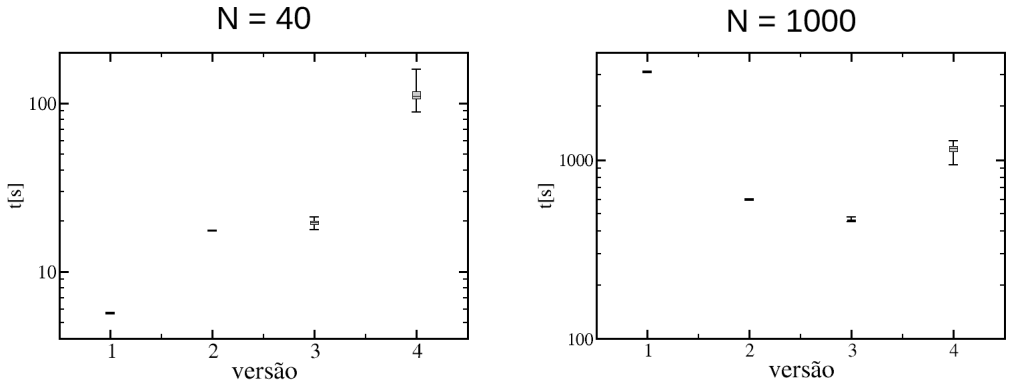
Figura 5.7: Tempo médio de execução para todas as quatro versões. Os pontos indicam os valores médios para cinco amostras. As barras de erro foram omitidas para maior clareza.



Fonte: a autora

Entretanto, para uma comparação conclusiva, cinco amostras não são suficientes para garantir as que as diferenças entre os algoritmos seja significativas. Por este motivo, escolhemos duas quantidades de partículas para fazer um comparativo com mais amostras, no caso trinta execuções idênticas. Simulamos, então, sistemas com $N = 40$ e $N = 1000$ partículas, o caso $N = 10000$ está em andamento.

Figura 5.8: As linhas centrais das caixas indicam a mediana dos dados. Os extremos da caixa são o primeiro e o quarto quartis dos tempos da amostra. As barras indicam os valores extremos dos tempos. Cada caixa foi construída com trinta amostras.



Fonte: a autora

5.5 Tamanho do Sistema Variável

Em simulações de segregação é bastante usual ajustar o tamanho total do sistema ao aumentar o número de partículas para manter a densidade de equilíbrio das células. Para levar isto em consideração, executamos os algoritmos descritos anteriormente, para sistemas com tamanho L variável. Com isto, L depende do número de partículas e da distância de equilíbrio da força, como na Equação 5.1.

$$L = \sqrt{\pi N r_e^2} \tag{5.1}$$

Como r_e é fixo, os valores de L para as quantidades de partículas podem ser visto na Tabela 5.2:

Tabela 5.2: Tamanho do sistema pelo número de partículas

N	L
10	2.2
20	2.75
40	4.4
60	4.95
80	6.05
100	6.6
200	9.9
400	13.75
600	17.05
800	19.8
1000	22.0

Fonte: a autora

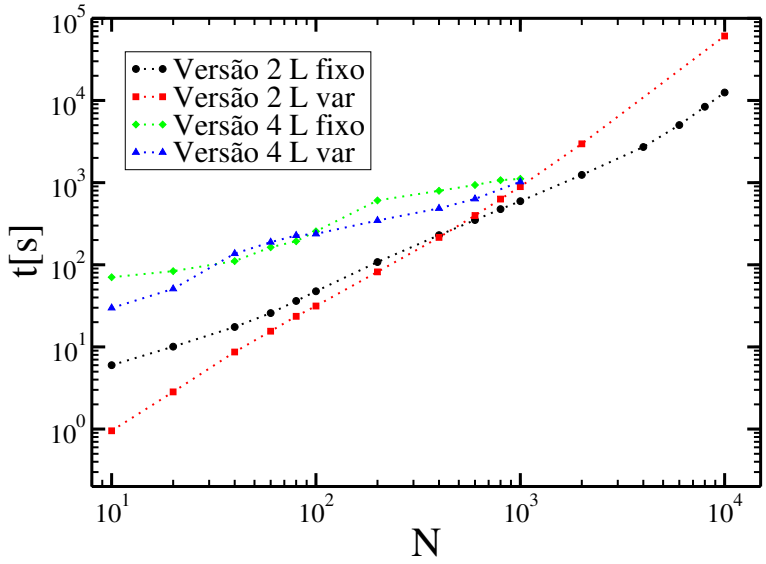
Os algoritmos que calculam todas as forças entre partículas não são afetados pelo tamanho do sistema. Como todas as interações são calculadas independente da densidade de partículas. Para os algoritmos com caixas, o fato de mantermos a densidade constante afeta o tempo de processamento. Por um lado temos mais caixas, que crescem com a raiz quadrada do número de partículas. Por outro, as caixas mantém a quantidade de partículas constante.

Os tempos de processamento para sistemas com tamanho variável, são comparados aos resultados anteriores com tamanho fixo na Figura 5.9. Estas simulações ainda estão sendo executadas, por esta razão não temos resultados conclusivos. Para os tamanhos apresentados, os tempos da versão 2 ainda estão no regime linear, porém sabemos dos cálculos da complexidade e dos resultados da Figura 5.3 que existe um regime qua-

drático para sistemas com mais partículas. Esperamos, então que as curvas se cruzem para sistemas com mais partículas.

Uma inversão no melhor algoritmo também é prevista para as versões em massivamente paralelas em GPU. Novamente, estes testes ainda estão em andamento, impossibilitando uma análise mais detalhada.

Figura 5.9: Sistemas com tamanho fixo com caixas: versão 2 para CPU (em preto) e versão 4 para GPU (em verde) comparados aos sistemas com tamanho variável: versão 2 (em vermelho) e versão 4 (em azul).



Fonte: a autora

6 CONCLUSÕES E PERSPECTIVAS

Analisamos formas de simular a segregação celular com diferentes hipóteses de forma controlada. O modelo de Vicsek é um modelo matemático mínimo que pode ser adaptado para simular o movimento de células de forma realista. Além disso ele permite que mais de uma hipótese da segregação celular seja simulada de forma direta. O modelo matemático pode ser transformado em diversos algoritmos, assim como implementado em diversas linguagens de programação. Apresentamos quatro formas distintas de algoritmo para simular o modelo de Vicsek: 1) Todas as distâncias são calculadas, algoritmo serial; 2) Método das caixas, cálculo das distâncias apenas de partículas próximas, algoritmo serial; 3) Todas as distâncias calculadas, algoritmo massivamente paralelo; 4) Método das caixas, algoritmo massivamente paralelo. Os algoritmos foram implementados em C e em CUDA.

Os primeiros resultados, foram para comprovar a complexidade assintótica típica calculada. A primeira versão do algoritmo apresenta o comportamento assintótico calculado desde sistemas com poucas partículas. Este algoritmo não é muito eficiente, do ponto de vista de cálculos desnecessários, pois ele calcula as distâncias entre todos os pares de partículas. Entretanto, para sistemas com poucas partículas ele apresenta os menores tempos de execução. A segunda versão do algoritmo tem o tempo de processamento linear com o número de partículas para sistemas pequenos, porém à medida que o sistema aumenta de densidade, o regime passa a ser quadrático. Esse regime quadrático pode ser evitado se a quantidade de caixas do sistema crescer com o número total de partículas, estes testes estão em execução e ficarão como perspectiva para trabalhos futuros.

Os algoritmos paralelos, à princípio deviam ser mais rápidos que as versões seriais, por efetuar diversos cálculos ao mesmo tempo. Entretanto, isso nem sempre é verdade, pois o paralelismo exige um controle maior na comunicação entre linhas paralelas e no sincronismo, resultando em tempo maiores de execução. O algoritmo paralelo, versão 3, que calcula todas as interações é o mais rápido para sistemas com mais de $N = 200$ partículas.

A versão paralela com método das caixas não ficou muito eficiente, pois para sistemas pequenos é versão que leva mais tempo para executar. Enquanto que para sistemas maiores, no melhor dos casos, chega a se igualar a versão serial equivalente. Mas nunca é melhor do que a versão paralela sem caixas. Estes tempos podem ser melhorados, com uma análise semelhante à apresentada na Figura 5.5 para o paralelismo nas caixas. É

possível também modificar o paralelismo dos cálculos das distâncias para um paralelismo em partículas, com isso, talvez a ocupação dos núcleos de processamento das placas seja maior e os resultados sejam melhores.

6.1 Perspectivas

Algumas das perspectivas de continuidade deste trabalho já foram brevemente apresentadas ao longo do capítulo de resultados, nesta sessão vamos entrar nos detalhes de cada uma delas.

Implementação dos Algoritmos em Paralelo na CPU: Uma comparação interessante que não foi abordada é o que acontece com algoritmos em paralelo para processadores (CPUs). A implementação pode ser feita através se bibliotecas de *threads* do próprio C ou até com auxilio de interfaces de programação de mais alto nível, como o OpenMP.

Modificar o nível de paralelismo em GPU: Para o algoritmo com método das caixas massivamente paralelo, foi utilizado o paralelismo em caixas para o cálculo das distâncias. Dessa forma cada linha paralela é responsável por todos os cálculos das distâncias e forças de todas as partículas de uma caixa. Este cálculo pode ser paralelizado á nível de partícula, assim, cada linha paralela é responsável pelos cálculos de interação de uma partícula. A função de cálculo da distância precisa ser reescrita para controlar quais partículas são potencialmente vizinhas.

Usar a memória compartilhada da GPU: Os detalhes da arquitetura interna das GPUs precisam ser levados em consideração. Podemos melhorar o desempenho dos programas ao usar de forma explicita a memória compartilhada dos blocos da GPU. Esta memória é utilizada por padrão pelo compilador, entretanto, pode-se obter resultados melhores controlando-se as cópias e os acessos a memória na programação dos *kernels*.

Simulações em outras Configurações de Servidores: A eficiência relativa das versões dos algoritmos é fortemente influenciada pelas configurações do servidor. Entretanto, neste trabalho, todos os testes foram executados em um único computador. Isto limita as conclusões que podem ser inferidas dos resultados, pois uma placa aceleradora com mais CudaCores ou com outra arquitetura talvez apresentasse resultados melhores, do que os seriais. Da mesma forma, as demais configurações do

servidor podem afetar os resultados.

Simulações para Sistemas com mais Partículas: Em alguns dos casos atingimos o regime assintótico calculado. Mesmo nestes casos, seria interessante dar continuidade aos testes. Nos demais, ainda não foi possível de atingir os regimes assintóticos, nestes casos sistemas maiores são necessários para decidir qual dos algoritmos possui o menor tempo de processamento. Uma possibilidade é considerar tempo final de simulação menor, pois todos os algoritmos de segregação são lineares com o número total de passos de tempo.

Unificar os códigos fonte em um único projeto: Não entramos em detalhe da organização dos códigos fonte das implementações apresentadas. Cada versão de algoritmo foi programada separadamente, uma perspectiva futura é a unificação dos códigos em um único software, permitindo ao usuário escolher qual o nível de paralelismo o programa utilizará.

Disponibilizar o software para a comunidade: Uma vez que os códigos sejam unificados e que o software passe em diversos testes para garantir seu funcionamento correto, vamos disponibilizar o software para a comunidade científica.

REFERÊNCIAS

ALLEN, M.; TILDESLEY, D. **Computer Simulation of Liquids**. [S.l.]: Clarendon Press, 1989. (Oxford Science Publ). ISBN 9780198556459.

ANDERSON, J. A.; LORENZ, C. D.; TRAVESSET, A. General purpose molecular dynamics simulations fully implemented on graphics processing units. **J. Comput. Physics**, v. 227, n. 10, p. 5342–5359, 2008.

BEATRICI, C.; BRUNNET, L. G. Cell sorting based on motility differences. **Physical Review E**, v. 84, n. 031927, 2011.

BELMONTE L. G. BRUNNET, G. L. T. R. M. C. d. A. H. C. J. Self-propelled particle model for cell-sorting phenomena. **Physical Review Letters**, v. 100, p. 248702, 2008.

BERTIN, E.; DROZ, M.; GRÉGOIRE, G. Hydrodynamic equations for self-propelled particles: microscopic derivation and stability analysis. **Journal of Physics A: Mathematical and Theoretical**, v. 42, n. 44, p. 445001, 2009.

BOGHOSIAN, B. M. Computational physics on the connection machine. **Computers in Physics**, v. 4, p. 14–33, 1990.

CORMEN, T. H. et al. **Introduction to Algorithms**. 2nd. ed. Cambridge, MA, USA: MIT Press, 2001. ISBN 0-262-03293-7, 9780262032933.

CURTIS, A. S. G. Timing mechanisms in the specific adhesion of cells. **Exp. Cel Res. Suplemento**, v. 8, p. 107–122, 1961.

DEICHMANN, U. et al. Commemorating the 1913 michaelis–menten paper die kinetik der invertinwirkung: three perspectives. **FEBS Journal**, v. 281, n. 2, p. 435–463, 2014. ISSN 1742-4658.

GESTEL F. J. WEISSING, O. P. K. J. v.; KOVÁCS, A. Density of founder cells affects spatial pattern formation and cooperation in bacillus subtilis biofilms. **The ISME Journal**, v. 8, p. 2069–2079, 2014.

GILBERT, S. F. **Developmental Biology**. 5th. ed. [S.l.]: Sinauer Associates Inc., 1997. ISBN 978-0878932443.

GRANER, J. G. F. Simulation of biological cell sorting using a two-dimensional extended potts model. **Physical Review Letters**, v. 69, p. 2013–2017, 1992.

GRÉGOIRE H. CHATÉ, Y. T. G. Moving and staying together without a leader. **Physica D**, v. 181, 2003.

HARRIS, A. K. Is cell sorting caused by differences in the work of intercellular adhesion? a critique of the steinberg hypothesis. **J. Theor. Biol.**, v. 61, p. 267–285, 1976.

HOCKNEY, R.; GOEL, S.; EASTWOOD, J. Quiet high-resolution computer models of a plasma. **Journal of Computational Physics**, v. 4, n. 2, p. 148 – 158, 1974. ISSN 0021-9991.

HOLTFRETER, J. A study of the mechanics of gastrulation. **Rev. Can. Bio.**, v. 3, p. 220–250, 1944.

JONES P. M. EVANS, D. A. L. B. M. Relation between the rate of cell movement under agarose and the positioning of cells in heterotypic aggregates. **Exp. Cel. Res.**, v. 180, p. 287–296, 1989.

KARR J. C. SANGHVI, D. N. M. M. V. G. J. M. J. B. B. J. N. A.-G. J. I. G. . M. W. C. J. R. A whole cell computational model predicts from genotype. **Cell**, v. 150, 2012.

LENNARD-JONES, J. E. Cohesion. **Proceedings of the Physical Society**, v. 43, n. 5, p. 461, 1931.

NVIDIA. **Whitepaper NVIDIA's Next Generation CUDA Compute Architecture: Kepler GK110/210**. [S.l.], 2009. Available from Internet: <http://www.nvidia.com.br/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf>.

NVIDIA. **Whitepaper NVIDIA GeForce GTX 980 Featuring Maxwell, The Most Advanced GPU Ever Made.** [S.l.], 2014. Available from Internet: <http://international.download.nvidia.com/geforce-com/international/pdfs/GeForce_GTX_980_Whitepaper_FINAL.PDF>.

NVIDIA. **Whitepaper NVIDIA's Next Generation CUDA Compute Architecture: Kepler GK110/210**. [S.l.], 2014. Available from Internet: <http://international.download.nvidia.com/pdf/kepler/NVIDIA-Kepler-GK110-GK210-Architecture-Whitepaper.pdf>.

NVIDIA. **CUDA C Programming Guide**. [S.l.], 2016. Available from Internet: <http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#axzz4RWVH8J50>.

PLIMPTON, S. Fast parallel algorithms for short-range molecular dynamics. **Journal of Computational Physics**, v. 117, n. 1, p. 1 – 19, 1995. ISSN 0021-9991.

REYNOLDS, C. W. Flocks, herds, and schools: A distributed behavioral model. In: ACM SIGGRAPH, 1987, California - USA. [S.l.], 1987. p. 25–34.

SCHERER, C. **Métodos Computacionais da Física**. [S.l.]: Editora Livraria da Física, 2005. ISBN 9788588325357.

SOLON, A. P.; CHATÉ, H.; TAILLEUR, J. From phase to microphase separation in flocking models: The essential role of nonequilibrium fluctuations. **Phys. Rev. Lett.**, American Physical Society, v. 114, p. 068101, Feb 2015.

STEINBERG, M. Reconstruction of tissues by dissociated cells. **Science**, v. 141, 1963.

SZABÓ G. L. SZÖLLÖSKI, B. G. Z. J. D. S. T. V. B. Phase transition in the collective migration of tissue cells: Experiment and model. **Physical Review E**, v. 74, n. 061908, 2006.

TURING, A. M. The chemical basis of morphogenesis. **Phil. Trans. Roy. Soc. London**, v. 237, p. 37–72, 1952.

VICSEK A. CZIRÓK, E. B.-J. I. C. O. S. T. **Phys. Rev. Lett.**, v. 75, n. 1226, 1995.

WAALS, J. D. V. der. The equation of state for gases and liquids. **Nobel Lecture**, 1910.

WILSON, V. H. On some phenomena of coalescence and regeneration in sponges. **J. Exp. Zool.**, v. 5, p. 245–258, 1907.