Simulation of Scale Free Gene Regulatory Networks based on Threshold Functions on GPU

Raphael R. Campos, Ricardo Ferreira, Julio C. Goldner Vendramini, Fábio Cerqueira
Departamento de Informática

Marcelo Lobato Martins
Departamento de Fisica
Universidade Federal de Vicosa, Vicosa, MG, 36570 000
ricardo@ufv.br

Abstract

Gene regulatory networks have been used to study diseases and cell evolution, where Random Boolean graphs are one of computational approaches. A Boolean graph is a simple and effective model, and its dynamic behavior has been used in several works. This article proposes an efficient environment to simulate Boolean graph on GPU (Graphics Processing Units). The dynamic behavior of a Boolean graph is computed by visiting the whole or a subset of state space. The proposed tool is based on statistical approaches to evaluate large graphs. Moreover, it can take into account scale free graphs with threshold functions. The experimental results show a speed-up factor of up to 40 times. In addition, the exploration of state spaces three orders of magnitude greater than previous approaches have been evaluated.

1. Introdução

Um dos grandes desafios, proposto pela Sociedade Brasileira de Computação, é a modelagem computacional de sistemas naturais [4]. Este trabalho se enquadra no desafio ao apresentar uma plataforma heterogênea para solução de um problema de bioinformática com alta demanda de processamento. As aplicações na área de bioinformática são um resultado da evolução do hardware e do software na área de computação, que permitem um alto desempenho para simular modelos teóricos em silício. Ao mesmo tempo, os avanços científicos na área de biologia permitem extrair um grande volume de dados para validar os modelos teóricos. As plataformas heterogêneas de software/hardware e as redes reguladoras de genes são temas atuais abordados em um estudo do estado da arte sobre o futuro dos sistemas eletrônicos [12]. Este propõe o uso de plataformas baseadas em CPU e GPU (*Graphics Processing Units*) na implementação de simuladores para redes reguladoras de genes.

O processo de evolução celular é fundamental para a compreensão dos processos de evolução dos organismos vivos, pesquisa de medicamentos, etc. A disponibilidade de dados experimentais nas duas últimas décadas motivou o estudo de vários modelos para simulação e validação com os dados. Um destes modelos foi proposto por Kauffman [10] em 1969 e revisto na década de 90 [11]. O modelo de Kauffman é baseado em um grafo Booleano aleatório onde a dinâmica do sistema é determinada pela ativação e desativação dos genes que são os vértices do grafo. Cada vértice tem o mesmo número de vizinhos, escolhidos de forma aleatória. Estudos recentes, mostram que modelos de grafos baseados na topologia livre de escala são mais realistas que os grafos aleatórios com um número fixo de vizinhos [2]. Em um grafo livre de escala [3], a maioria dos vértices tem poucos vizinhos e alguns poucos vértices possuem muitos vizinhos. Este artigo apresenta um ambiente de simulação para manipular grafos livres de escala.

Cada vértice armazena uma função Booleana que é responsável pela atualização de seu estado. Segundo o modelo de Kauffman, a função Booleana de cada vértice também é escolhida de forma aleatória. Neste trabalho, o ambiente de simulação possibilita a exploração do espaço de estados do grafo onde as funções de atualização dos vértices são baseadas em limiar que são um sub-conjunto das funções Booleanas. Estas funções possuem propriedades interessantes e vários pontos em aberto que vem motivando estudos recentes [5].

A dinâmica do sistema estuda o diagrama de estados do grafo, os atractores e as suas bacias. Um atractor representa a estabilidade em um ciclo do diagrama de estados no modelo teórico, no caso de uma rede regulatória, eles são mais fáceis de serem observados experimentalmente e podem ter

uma interpretação funcional associada como por exemplo o tipo celular[11].

Este trabalho apresenta três contribuições. Primeiro, avalia a aceleração no tempo de execução na simulação de redes reguladoras com grafos Booleanos implementados em GPU. Segundo, avalia grafos Booleanos com topologia livre de escala com mais de 20 vértices. Em terceiro lugar, apresenta um ambiente eficiente para modelagem e estudo de funções baseadas em limiar com topologia livre de escala para o estudo dos atractores das redes reguladoras.

Um ambiente de simulação com tempo de execução baixo se faz necessário pois para estudar uma classe de grafos para uma determinada classe de redes reguladoras, milhares de grafos devem ser gerados e para cada grafo milhões de estados são explorados. Este trabalho apresenta uma redução da ordem de 20 a 30 vezes no tempo de execução.

2. Conceitos Básicos

2.1. Rede Booleana

Kauffman [11] propôs um grafo Booleano para representar as redes reguladoras de genes, onde cada vértice representa um gene. O gene no estado ativo (inativo) é representado pelo valor 1 (0). As arestas representam a interação entre os genes. No modelo síncrono, a cada instante de tempo, os valores de todos os vértices são recalculados em função dos valores de seus vizinhos ou vértices adjacentes. O valor depende da função local. Na proposta original de Kauffman, os vértices tem funções Booleanas aleatórias.

O modelo de Kauffman possui uma limitação. As estruturas encontradas na natureza possuem uma distribuição irregular de arestas, onde alguns vértices possuem muitos vizinhos e a maioria poucos. Este modelo difere do modelo de Kauffman onde todos os vértices tem o mesmo número de vizinhos [2, 3].

O grafo é muito sensível as funções aleatórias. A dinâmica do grafo tem que ser estável para não ser afetada pela maioria das pertubações. Ao mesmo tempo, o grafo deve reagir para alguns casos específicos e se adaptar para permitir a evolução dos mecanismos. Ou seja, o modelo deve ser robusto e ao mesmo tempo adaptável.

Kauffman propôs a teoria da vida à beira do caos [11]. Os grafos de Kauffman em geral tem um grau pequeno em torno de 2 ou 3. O ajuste dos paramêtros deve ser feito para gerar grafos no estado crítico. O estado crítico é situado entre o estado ordenado e o estado caótico dando origem ao termo "à beira do caos". Aldana mostrou que os grafos livres de escala possuem um espaço de fase onde se pode ajustar com mais flexibilidade a modelagem dos grafos no estado crítico entre o caos e a ordem.

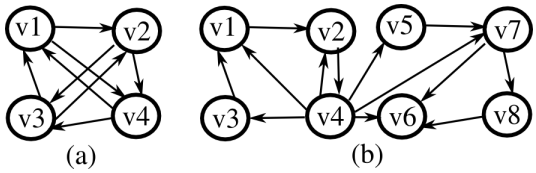

Figura 1. (a) Kauffman (b) Livre de Escala

2.2. Livre de Escala

Um grafo livre de escala possui uma probabilidade de distribuição de arestas por vértice baseada na equação $P(k) = k^{-\gamma}$, onde γ é o coeficiente livre de escala e k o número de vizinhos. Valores de γ mais baixo, próximos de 1, representam grafos em um estado mais caótico. A maioria dos grafos propostos e validados com dados experimentais possuem coeficiente entre 2 e 3 [2]. Em um grafo aleatório é pouco provável que um vértice tem um número grande de vizinhos bem acima da média k de vizinhos do grafo. Em um grafo livre de escala, a função aleatória segue uma lei de potência e existe a possibilidade de ser gerar vértices com muitos vizinhos bem como com poucos vizinhos, independente do tamanho do grafo.

Os grafos livre de escala aparecem na natureza em várias situações [3], seja na biologia, na topologia da internet, nas redes de citações de artigos, nas redes sociais dentre outras.

A Figura 1(a) mostra um grafo de Kauffman com $k = 2$ onde cada vértice tem exatamente 2 vizinhos. No modelo livre de escala, os vértices denominados *hubs* concentram várias ligações e tem um grau de conectividade alto. Por outro lado, a maioria dos outros vértices possuem poucas ligações como pode ser visualizado na Figura 1(b). O vértice v_4 é um hub no grafo que influencia vários outros. A maioria dos vértices tem apenas uma aresta de saída e uma de entrada.

Para se construir aleatoriamente um grafo livre de escala, pode se começar com uma clique [3]. Suponha $k = 2$, então será gerada uma clique de três vértices. Para cada novo vértice a ser inserido no grafo, são sorteados dois vértices para serem seus vizinhos onde a probabilidade de ser sorteado depende do grau do vértice. Assim, durante o processo de geração do grafo, vértices com maior grau tem uma maior probabilidade de serem sorteados. Ou seja, o "rico" fica mais "rico".

Uma das contribuições é propor uma ambiente para estudo de grafos livres de escala. O espaço de soluções é muito grande. Como estudo de caso para validar nossa ferramenta iremos abordar grafos livres de escala com funções baseadas em limiar.

2.3. Função Limiar

O espaço de funções locais é ainda maior que o número de estados. Por exemplo, para um grafo de Kauffman com $k = 2$, teremos $2^{2^k} = 16$ possibilidades de funções Booleanas de duas entradas. A maioria das funções é simples, pois uma única variável determina seu valor com uma função *AND*, onde se uma variável tem o valor 0, a saída será sempre 0 independente das outras variáveis. Uma função pouco explorada é a função XOR.

Agora se considerarmos $k = 5$ teremos 4 bilhões de possíveis funções por vértice. Para simplificar a modelagem e buscar um significado para os genes e sua dinâmica, a maioria dos trabalhos usa funções canalizadoras, onde o valor de uma variável determina com prioridade o valor da função. Por exemplo, suponha que o vértice v_1 depende de seus vizinhos v_2, v_3 e v_4. Se a função é canalizada por v_2, primeiro v_2 irá determinar o valor da saída ou canalizar o seu valor para 1 ou para 0. Apenas se v_2 não estiver ativo que os outros vértices irão influenciar. As funções canalizadoras fazem o grafo ter uma convergência mais rápida.

Outra possibilidade que ainda foi pouco explorada nos grafos Booleanas para redes reguladoras de genes são as funções baseadas em limiar que podem ser calculadas com operações de soma, também chamadas de funções aditivas. A implementação destas funções é direta e eficiente nos processadores e GPU. Um estudo recente [5] mostra que existe um correlação entre as funções aditivas e os dados experimentais. Estas funções se aproximam mais que as soluções baseadas em funções Booleanas.

Seja Viz um conjunto com k vizinhos para um vértice v_i. Se o vizinho v_j tiver o valor 1 (0), irá contribuir com +1 (-1). O valor de v_i será 1 se a soma dos vizinhos for maior que um certo limiar L, para $j = 1, \ldots, k$. Caso contrário o valor será zero. Formalmente, o novo valor será $v_i = (\sum_{j=0}^{k}(v_j \cdot (+1) + \overline{v_j} \cdot (-1)) > L$. Suponha que o limiar seja zero. Ou seja, o vértice irá receber o valor da maioria ou majoritário entre os seus vizinhos. Esta equação pode ser reescrita na forma $v_i = (2 * \sum_{j=0}^{k} v_j) > k$, sendo uma implementação mais eficiente. Outros valores de limiares pode ser incorporados bem como pesos diferentes para cada vizinhos. O espaço de solução é bem amplo. Neste trabalho apresentamos uma ferramenta que pode ser usada para esta exploração. Foge ao espoco deste trabalho um estudo detalhado das funções.

2.4. Diagrama de Estados

Um grafo com N vértices terá 2^N estados. Para simplificar a explicação iremos usar um grafo regular com 2 vizinhos por vértices como ilustrado na Figura 2(a). Suponha a função aditiva. O diagrama de estados do grafo, que também está descrito no formato de um grafo é ilustrado na

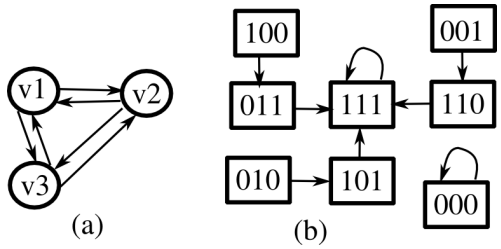

Figura 2. (a) Grafo (b) Diagrama de Estados

Figura 2(b). O grafo possui 3 vértices e um diagrama de $2^3 = 8$ estados. Cada vértice no diagrama é representado pelo estado dos seus três vértices. Por exemplo, o vértice 000 representa o estado no qual todos os vértices tem o valor 0. Para cada estado a notação usada é $v_1v_2v_3$. Portanto 101 significa que $v_1 = 1, v_2 = 0, v_3 = 1$.

O grafo possui dois atractores: 000 e 111. Ambos tem comprimento 1. Ou seja, são atractores com 1 estado ou atractores simples. O atractor 000 não tem outros estados na sua bacia. Já o atractor 111 possui 6 estados na sua bacia. Estes estados convergem para o atractor. Por exemplo, o estado 100 irá evoluir na simulação para o estado 011 que depois irá passar para o estado 111 onde ficará preso no atractor.

Podemos notar que o diagrama de estados é um grafo com vários componentes conexos. Cada atractor e sua bacia formam um componente conexo. Neste modelo o comportamento de uma célula é determinado pelo seu estado dinâmico. Suponha que a célula esteja no estado 000. Se alguma mutação ou pertubação afetar um dos seus genes, por exemplo v_1 passa para 1, a célula irá mudar de comportamento, passando para 001, depois 110 e se estabilizar no atractor 111.

Os estudos dos modelos de grafos Booleanos com correlação com os experimentos de Biologia tem mostrado que os diagramas de estados tem poucos atractores. Além disso, o comprimento dos atractores é pequeno e em poucos passos a partir de um estado inicial aleatório, o grafo evolui para um atractor rapidamente.

3. Algoritmo

3.1. GPU

O uso de GPU vem se ampliando na comunidade científica. Neste trabalho iremos considerar que os leitores estão familiarizados com os conceitos básicos da GPU. Para maiores detalhes sugerimos as seguintes referências [14, 13].

As maiores vantagens da GPU são: a rápida curva de aprendizado com o uso de CUDA ou OpenCL, o grande número de *threads* (milhares) que podem ser instanciados

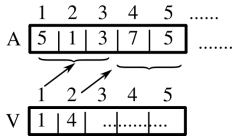

Figura 3. Estrutura Dados do Grafo

em paralelo, a disponibilidade do hardware e seu baixo custo, o alto desempenho em ponto flutuante e o acesso rápido e em paralelo a memória.

Uma das limitações da geração atual de GPUs é o modelo de execução SIMD onde cada *thread* tem que executar a mesma operação. Além disso, se faz necessário o conhecimento de detalhes do hardware para otimizações como o acesso a memória, transferência para memória compartilhada e indexação dos dados em vetores para mapeamento eficiente, além do balanceamento no uso dos *warps*. Outro fator limitante é a baixa taxa de transferência de dados entre a CPU e a GPU. As duas unidades poderiam se complementar mais se o desempenho da comunicação fosse maior.

3.2. Grafo em Tempo de Execução

Um software de geração dos grafos livre de escala foi escrito em C++ e implementado na CPU, seguindo o modelo de Barabasi [3]. A eficiência na execução depende da estrutura de dados para manipulação dos grafos e do modelo de implementação paralela do algoritmo.

A proposta deste trabalho é construir um gerador e simulador de grafos. Ademais, para cada instância de um determinado grafo, vários estados iniciais serão avaliados. Ou seja, o estudo de uma determinada classe de rede reguladora de genes pode demandar a geração de milhares de grafos e simulação de bilhões de estados.

Os principais algoritmos de grafos implementados em GPU podem ser encontrados em [7]. O paralelismo é modelado a nível de vértice. Entretanto, para grafos livres de escala o *hubs* irão desbalancear a execução gerando um uso ineficiente da GPU. A maioria dos vértices irá atualizar o seu valor rapidamente e ficarão a espera dos *hubs* que possuem muitos vizinhos.

A abordagem proposta é que cada *thread* simule uma instância de grafo por inteiro a partir de um estado inicial diferente. Para cada *thread*, um estado inicial aleatório é gerado, o diagrama de estados é percorrido até encontrar um atractor. Como os atractores tem comprimento e bacias semelhantes, os *threads* ficam balanceados.

A estrutura de dados para armazenar o grafo é composta por um vetor de vizinhos como ilustrado na Figura 3. O vetor V armazena ponteiros para a lista de arestas do vértice v_i que está armazenada no vetor A das arestas. Por exemplo, v_1 terá 3 vizinhos, os vértices v_5, v_1 e v_3. Pois os vizinhos

```
For i=1 to N
  soma = 0;
  For j= A[i] to A[i+1]-1
     soma = soma + v[j];
  v[i] = (2*soma > (A[i+1]-A[i]) ) ? 1 : 0;
```

Figura 4. Varredura do Grafo

de v_2 começam na posição 4 no vetor de arestas. Suponha por sua vez que o vértice v_2 tem 2 vizinhos v_7 e v_5. Consultando o vértice atual e o próximo sabemos quantos e quais são os vizinhos. Desta maneira, o grafo pode ser percorrido com dois loops aninhados como ilustrado na Figura 4.

3.3. Grafo em Tempo de Compilação

Uma segunda alternativa foi avaliada neste trabalho. O gerador de grafos gera um kernel específico para cada grafo, com o código das arestas mapeado em tempo de compilação. Por exemplo, se v_1 depende de v_2, v_3 e v_4, então será gerado o seguinte comando condicional em C: $v_1 = (2 * (v_1 + v_2 + v_3 + v_4) > 2)?1 : 0$. Esta opção elimina os dois loops aninhados apresentados na seção anterior.

Esta técnica evita a indireção e representação do grafo nos vetores de arestas. Uma pequena aceleração é obtida. Porém devido ao aumento considerável no tempo de compilação na versão 2.1 do compilador da NVIDIA, para os exemplos avaliados neste trabalho, o tempo total de compilação mais execução é menor na versão com o grafo em memória, quando para cada kernel são avaliados apenas 1 milhão de estados iniciais.

3.4. Fila

Para determinar o tamanho do atractor no modelo síncrono pode-se usar duas abordagens. A primeira é baseada no algoritmo apresentado em [1], que não necessita memória e pode capturar atractores de qualquer tamanho. A desvantagem é que o caminho do estado inicial até achar o atractor no diagrama de estados é percorrido três vezes. A segunda abordagem usa uma fila para armazenar os últimos estados da simulação. A cada passo, todos os estados da fila são verificados. Se o estado atual estiver presente na fila, o atractor é detectado. O caminho do atractor no diagrama é visitado uma única vez. Entretanto, se a fila tiver um tamanho T, esta abordagem só encontrará atractores menores que T.

A maioria dos grafos derivados dos dados experimentais possuem pequenos atractores. Portanto, a versão baseada em fila foi utilizada com um valor máximo entre 10 e 20.

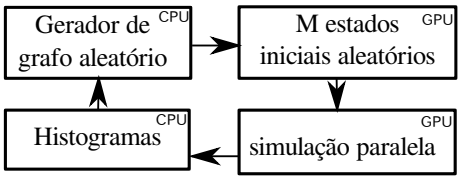

Figura 5. Diagrama Blocos da Simulação

Nos grafos avaliados não foram encontrados valores maiores, resultando em uma aceleração de no mínimo de três vezes em comparação com a abordagem sem uso de memória [1].

3.5. Histogramas

Um estudo das propriedades dos grafos livre de escala para um valor máximo de $N = 20$ foi implementado em [2]. Em cada experimento, 20 mil redes aleatórias foram geradas. Para cada grafo todos os estados foram avaliados, o que limitou o estudo a grafo com 20 vértices que tem 1 milhão de estados.

Nossa abordagem permite funções Booleanas ou baseadas em Limiar. O usuário especifica o número de grafos e para cada grafo quantos estados iniciais serão avaliados. Para grafos maiores que um determinado valor que pode ser especificado, apenas um subconjunto de estados iniciais é usado para explorar o diagrama de estados. O que permite ampliar os grafos estudados com tamanhos maiores que 20 além de avaliar com abordagens estatísticas um subconjunto de estados sem a necessidade de avaliar o diagrama de estados por completo que tem um custo exponencial e proibitivo para grafos com mais de 25 vértices. Neste trabalho foram avaliados grafos com até 60 vértices.

4. Resultados Experimentais

A Figura 5 apresenta a estrutura do ambiente de simulação. O módulo gerador de grafo aleatório executa na CPU, onde X grafos são gerados para uma dada função de distribuição aleatória. O usuário especifica o fator γ e o número n de vértices do grafo. O usuário especifica também o número M de estados iniciais que serão explorados para cada grafo. Como mencionado anteriormente, o estudo de uma classe de funções podem envolver a geração de milhares de grafos e para cada grafo milhões de estados, o que demanda um ambiente de simulação com baixo tempo de execução.

O segundo módulo executa na GPU. Para um dado grafo, M estados iniciais são gerados. Quando $M \leq 2^n$, todos os estados do grafo podem ser explorados. Se $M > 2^n$, o conjunto de M estados é gerado aleatoriamente na GPU. Apenas para grafos com menos de 30 vértices é viável explorar todo o espaço. Entretanto, como o tamanho máximo para os ciclos dos atractores para a classe de funções que iremos explorar é pequeno e o tamanho médio das bacias também, portanto, nossa abordagem baseada em uma amostragem aleatória para gerar um subconjunto de estados é suficiente para avaliar o comportamento da classe de grafos em estudo.

O terceiro módulo executa um thread para cada estado inicial na GPU. O resultado de cada simulação é agrupado em um vetor com as seguintes informações: tamanho do atractor, tamanho da bacia do atractor e identificador do atractor. Finalmente, estes dados são transferidos para a CPU que monta os histogramas para os atractores e suas bacias (quarto módulo na Figura 5).

Iremos comparar os resultados do método proposto com dois outros trabalhos correlatos. O primeiro proposto por Aldana [2] faz um estudo de redes livre de escala mas só apresenta resultados para grafos até 20 vértices. Além disso, este trabalho não apresenta os tempos de execução. O segundo trabalho proposto por Garg *et al.* [6] apresenta uma ferramenta baseada em BDD que é capaz de fazer a exploração de grafos com até 1000 vértices ou mais. Apesar de fazer a exploração implícita de todos os estados, mesmo para 2^{1000}, o desempenho depende do tipo de função a ser simulada. Os BDDs, base da ferramenta, tem custo exponencial para algumas classes de funções. Iremos mostrar que para funções baseadas em limiar, a ferramenta não apresenta um bom desempenho e não é viável a exploração de grafos com mais de 20 vértices.

Iremos mostrar uma série de experimentos para validar nossa proposta. As características das plataformas de execução são: GPU GTX 285 com 1 G byte de memória e 240 núcleos; uma CPU dual-core 2.8GHz com 4 G bytes de RAM. O primeiro experimento compara o tempo de execução para avaliar todos os estados para 1000 grafos livres de escala. Os grafos tem de 20 a 28 vértices. A tabela 1 compara o tempo de execução da nossa abordagem para busca de atractores com a abordagem baseada em BDD proposta em [6]. A primeira coluna mostra o tamanho do Grafo em número de vértices. A segunda coluna o fator γ. Depois são apresentados os tempos de execução para a versão em GPU para a abordagem proposta neste artigo. A coluna BDD apresenta o tempo de execução da ferramenta baseada em BDD [6]. A última coluna mostra o speedup.

O tempo de execução da nossa implementação é de uma a 8600 vezes mais rápido. Apenas para 2 casos é equivalente ao tempo com o BDD [6]. A nossa abordagem também foi implementada em CPU e executa em média 30 vezes mais lento que na GPU. Para grafos com 40 ou 60 vértices, a ferramenta baseada em BDD não terminou a execução no tempo limite de uma hora para um grafo. Nossa implementação em GPU executa em 2 minutos um

Tabela 1. Tempo de Execução para Grafos Livres de Escala

Vértices	γ	GPU (seg)	BDD (seg)	speedup
20	1,1	77,9	3840	49
20	2,5	94,6	80	0,85
24	1,1	80,7	21340	264
24	2,5	121,1	230	2
28	1,1	83,1	719840	8659
28	2,5	118,4	1650	13,9

grafo com 60 vértices, sendo 40 vezes mais rápida que nossa versão em CPU, com o limite de 1 milhão de estados.

Os próximos experimentos são semelhantes aos apresentados por Aldana [2] para grafos livres de escala com 20 vértices e funções Booleanas aleatórias. Entretanto, o trabalho proposto aqui irá explorar grafos maiores com até 60 vértices. Iremos abordar os grafos com a função limiar. Todos os vértices tem o limiar igual a $k/2$, onde k é o número de vértices. Não é possível comparar o tempo de execução pois estes dados não são apresentados em [2]. Além disso, diferente da ferramenta de BDD [6] que só calcula o número e tamanho do atractor, nossa abordagem irá medir também o tamanho da bacia dos atractores. A ferramenta baseada em BDD também é limitada pois não viabiliza a exploração das funções livres de escala para $N > 20$, mesmo que estas funções gerem grafos com poucos atractores e ciclos pequenos.

Para $n = 28$ que tem mais de 500 bilhões de estados, a GPU executa em 10 horas para exploração completa de 1000 grafos. Na CPU, o tempo de execução é mais de 20 vezes maior. Ou seja, é possível explorar completamente grafos com um espaço de estado com três ordens de grandeza superior ao trabalho de Aldana [2]. Nos próximos experimentos consideramos apenas 1 milhão de estados iniciais aleatórios.

Serão apresentados histogramas com a distribuição do tamanho dos atractores, do número de atractores por grafo e do tamanho da bacia de atractores. O usuário especifica o número de vértices, o fator γ, o número máximo de estados iniciais por vértices e o número de grafos a serem explorados.

A Figura 6 apresenta a distribuição do tamanho dos atractores l_a para 1000 grafos com 20 vértices e $\gamma = 1,1$. Podemos observar que existe uma forte concentração de atractores com ciclo igual a 1. A probabilidade $P(l_a)$ para tamanho 1 é de 70%, 25% tem tamanho 2 e o restante 3 ou 4. Para este experimento a GPU executou 26 vezes mais rápido que a CPU.

Primeiro iremos observar se o tamanho dos atractores é sensível ao fator γ. Ao executar para $\gamma = 2.5$, vemos que 77% tem tamanho 1 e 14% tem tamanho 2. Outros valores

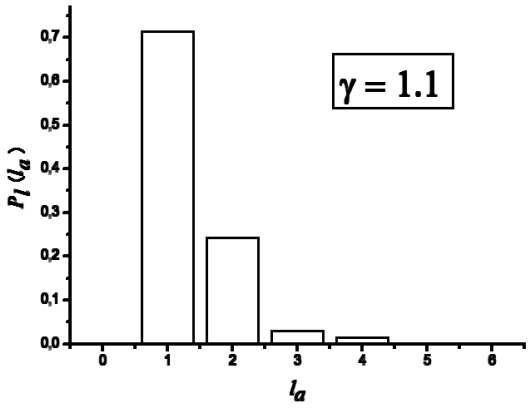

Figura 6. Tamanho do ciclo para $N = 20$

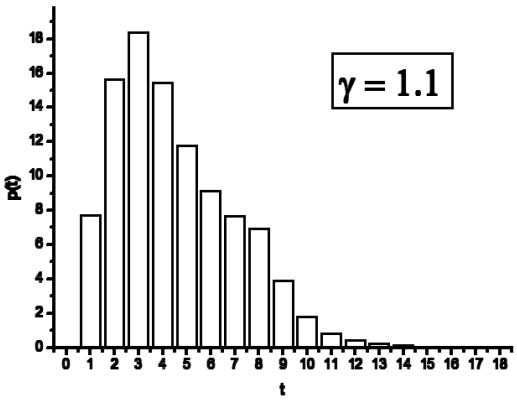

Figura 7. Tamanho do transiente para $N = 20$

de γ devem ser obsevados, o objetivo aqui é mostrar que a ferramenta permite esta exploração. Não é o escopo deste trabalho fazer esta análise, apenas apresentar os recursos onde podemos aplicar um ambiente eficiente em tempo de execução. Neste exemplo, a simulação executa em 1 minuto para cada valor de γ. Os próximos experimentos também servirão para ilustrar os pontos que podem ser explorados em trabalhos futuros.

As figuras 7 e 8 apresentam as distribuições dos tamanhos dos transientes dos atractores t_a para 1000 grafos com 20 vértices e $\gamma = 1,1$ e $2,5$. Podemos observar que existe uma forte concentração no transiente igual a 3, porém com o fator $\gamma = 1,1$, a distribuição é mais suave.

Outro experimento é verificar quantos attractores existem em cada grafo. Para o caso estudado temos 1000 grafos. A figura 9 apresenta as distribuições número de atractores n_a para 1000 grafos com 20 vértices e $\gamma = 1,1$. Podemos observar que existe uma forte concentração no valor 2, ou seja, a maioria dos grafos tem 2 atractores. Juntando todos os dados, vemos que para $\gamma = 1,1$ e $n = 20$, funções

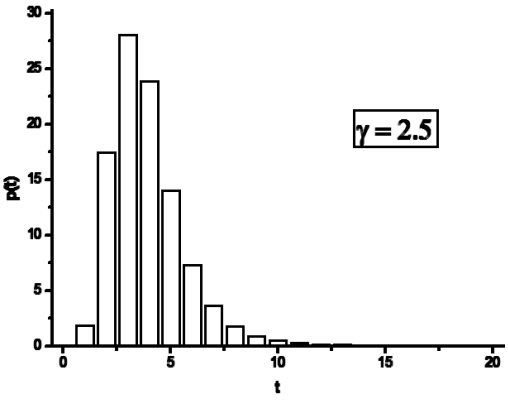

Figura 8. Tamanho do transiente para $N = 20$

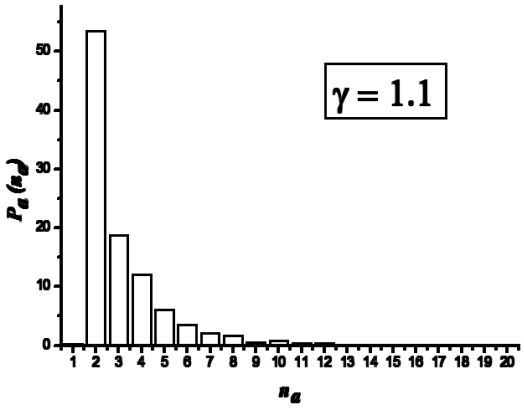

Figura 9. Número de atractores para $N = 20$

limiares $k/2$, topologia livre de escala temos em média 2 atractores com 70% de probabilidade de ser de tamanho 1 e com transiente médio em torno de 3.

Finalmente na figura 10 verificamos a exploração com 1 milhão de estados iniciais para um grafo com 60 vértices. Podemos observar que o comportamento é similiar para o tamanho médio dos transientes com valores em torno de 3. Porém com uma distribuição mais equilibrada com as ocorrências de valores 2 e 4. O número de atratores do grafo gira em torno de 2 para 65% dos casos e existe 95% de probabilidade do attractor tem comprimento 1.

5. Trabalhos Correlatos

Vários trabalhos fazem um estudo dos atractores em rede Booelanas [1, 2, 8, 9]. Porém poucos mostram o tempo de execução. No estudo das redes reguladores com topologia livre de escala proposto inicialmente em [2], não são apresentados tempo de execução e o tamanho máximo avaliado foi 20. Um algoritmo $O(n^3)$ é apresentado em [9]. porém a

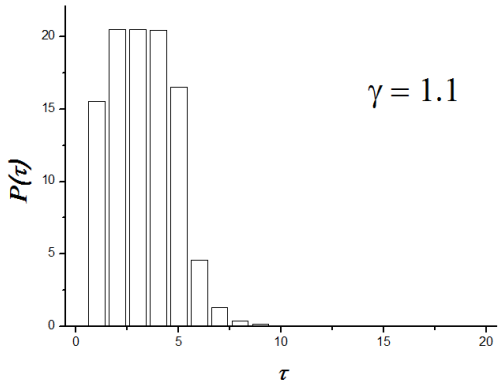

Figura 10. Transiente para $N = 60$

execução de um grafo com 80 vértice gasta em torno de 100 segundos. Nossa abordagem simula em torno de 1000 grafos no mesmo tempo de execução. Uma implementação baseada em grafos probabilísticos em FPGA foi apresentada em [17]. Entretanto não são tratados grafos livre de escala e é restrito a funções Booleanas probabilísticas. O tempo de compilação e configuração do FPGA não é considerado e a comparação é feita com uma versão ineficiente implementada em Matlab. Outras abordagens paralelas foram propostas baseadas em FPGA [16, 15]. Porém uma propõe construir um novo FPGA que possui um grande custo. A outra abordagem é baseada em redes bayesianas e e esta restrita a grafos com menos de 10 vértices.

Calcular o tamanho do atractor é um problema NP-completo[18] Uma solução eficiente que visita implicitamente todos os estados foi proposta em [6]. O tempo de execução é pequeno pois a abordagem é baseada em diagramas de decisão binária (BDD) que manipula implicidamente todos os estados. Esta abordagem é derivada das ferramentas de modelagem de máquinas de estados finitos em verificação formal de circuitos. Apesar da ferramenta ser rápida para a maioria dos grafos, o desempenho para funções com limiar pode demorar horas nos casos avaliados neste artigo. O gerador desenvolvido neste trabalho é compatível com o formato da ferramenta baseada em BDD [6]. A grande contribuição desta ferramenta baseada em BDD é o estudo completo de todos os estados mesmo para grafos com 1000 vértices que é impossível de ser realizado com a exploração explicita dos estados. Entretanto a limitação é a restrição a classes de funções Booleanas. Além disso, a maioria dos diagramas tem uma convergência rápida onde uma avaliação estatística é suficiente. Outros pontos são os dados de histograma que a ferramenta de [6] não fornece. Ela fornece apenas o número de atractores e seu comprimento. Nossa abordagem deriva dados sobre as bacias dos atractores e seus transientes. Ademais, as duas ferramentas se complementam e podem ser usadas como um ambiente

de exploração e validação dos diagramas.

6. Conclusão

Este trabalho apresenta o uso de GPU para a modelagem e simulação de redes reguladoras de genes com o modelo livre de escala, grafos Booleanos e funções com limiares. As principais contribuições foram a formulação e implementação eficiente tanto em CPU quanto em GPU em relação aos trabalhos anteriores. O estudo de grafos com a avaliação completa dos estado foi realizado para grafos com até 28 vértices que é três ordens de grandeza maior que o avaliados completamente pelo [2]. Uma amostragem de um milhão de estados é usada para avaliação estatística de grafos maiores. A solução usa apenas um computador pessoal equipado com uma placa de GPU. A grande disponibilidade de *clusters* com centenas de placas de GPU e a paralelização direta da nossa abordagem pode possibilitar até o estudo completo de grafos com mais de 40 vértices. Em se tratando de uma exploração parcial, grafos com centenas de vértices podem ser validados bem como a convergência da simulação.

Na prática, as redes reguladores tem de 20 a 2000 vértices. Muitas bases de dados de biologia apresentam redes com o número de vértices em torno de 50 a 100.

Os resultados mostraram um ganho de 20 a 40 vezes no tempo de execução da GPU em relação a CPU e de até 8000 vezes em relação a implementação com BDD [6]. Como trabalhos futuros juntamente com os professores do Departamento de Física e Biologia da Universidade Federal de Viçosa, se pretende ampliar os modelos e a ferramenta para avaliar uma ampla ganha de funções limiares e suas topologias.

7. Agradecimentos

Este artigo teve apoio das seguintes instituições: UFV, CAPES PROAP, FAPEMIG (CEX APQ 00472-10), O estudante Raphael Rodrigues Campos é bolsista de Iniciação Científica do projeto FAPEMIG.

Referências

[1] A.Bhattacharjya and S. Liang. Median attractor and transients in random boolean nets. *Physica D: Nonlinear Phenomena*, 95:29–34, 1996.

[2] M. Aldana. Boolean dynamics of networks with scale-free topology. *Physica D*, 185:45–66, 2003.

[3] A.-L. Barabasi and Z. N. Oltvai. Network biology: Understanding the cell's function organization. *Nature Reviews - Genetics*, 5:101–113, 2004.

[4] A. P. L. Carvalho, L. Ribeiro, and et alli. Grandes desafios da pesquisa em computação no brasil 2006-2016. Technical report, Sociedade Brasileira de Computação, 2006.

[5] C. Darabos, F. Di Cunto, M. Tomassini, J. Moore, P. Provero, and M. Giacobini. Validating a threshold-based boolean model of regulatory networks on a biological organism. In C. Pizzuti, M. Ritchie, and M. Giacobini, editors, *Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics*, volume 6623 of *Lecture Notes in Computer Science*, pages 59–68. Springer Berlin / Heidelberg, 2011.

[6] A. Garg, A. D. Cara, I. Xenarios, L. Mendoza, and G. D. Micheli. Synchronous versus asynchronous modeling of gene regulatory networks. *Bioinformatics Systems Biology*, 24(17):1917–1925, 2008.

[7] P. Harish and P. Narayanan. Accelerating large graph algorithms on the gpu using cuda. In S. Aluru, M. Parashar, R. Badrinath, and V. Prasanna, editors, *High Performance Computing – HiPC 2007*, volume 4873 of *Lecture Notes in Computer Science*, pages 197–208. Springer Berlin / Heidelberg, 2007.

[8] K. Iguchi, S. Kinoshita, and H. Yamada. Boolean dynamics of kauffman models with scale-free networks. *Journal of theorical biology*, 247:138–151, 2007.

[9] D. J. Irons. Improving the efficiency of attractor cycle identification in boolean networks. *Physica D*, 217:7–21, 2006.

[10] S. A. Kauffman. Metabolic stability and epigenesis in randomly constructed genetic nets. *J. Theoret. Biology*, 22:437–467, 1969.

[11] S. A. Kauffman. *The Origins of Order: Self-Organization and Selection in Evolution*. Oxford University Press, USA, 1 edition, June 1993.

[12] G. D. Micheli. An outlook on design technologies for future integrated systems. *IEEE Trans. Comp.-Aided Des. Integ. Cir. Sys.*, 28(6):777–790, 2009.

[13] J. Nickolls and W. J. Dally. The gpu computing era. *IEEE Micro*, 30(2):56–69, 2010.

[14] F. Pereira. *Atualizações em informática 2011*, pages 259–302. Sociedade Brasileira de Computação, Editora PuC Rio, Natal, Brasil, 2011.

[15] I. Pournara, C. Bouganis, and G. Constantinides. Fpga-accelerated bayesian learning for reconstruction of gene regulatory networks. In *IEEE Field Programmable Logic Conference*, pages 323–328. IEEE Computer Society, 2005.

[16] I. Tagkopoulos, C. Zukowski, G. Cavelier, and D. Anastassiou. A custom fpga for the simulation of gene regulatory networks. In *13th ACM Great Lakes Symposium on VLSI*, 2003.

[17] M. Zerarka, J. David, and E. M. Aboulhamid. High speed emulation of gene regulatory networks using fpgas. In *47th IEEE International MidWest Symposium on Circuits and Systems*, pages 545–548, 2004.

[18] S.-Q. Zhang, M. Hayashida, T. Akutsu, W.-K. Ching, and M. Ng. Algorithms for finding small attractors in boolean networks. *Eurasip Journal on Bioinformatics and System Biology*, pages 1–13, 2007.