# Simulação de Redes Reguladoras de Genes com Lógica Booleana e Limiar em Plataformas Alto Desempenho*

**Wallace F. Rosa[1], Hector Baranda [1], Michael Canesche[1], Marcelo Menezes[1], Lucas Bragança[1], Salles Magalhães[1], José A. Nacif[2], Ricardo Ferreira[1]**

[1]Depto de Informática – Universidade Federal de Viçosa, Brasil

***Abstract.*** *Gene regulatory networks are graph-based models widely used to study cell behavior, cell differentiation processes, or the treatment and evolution of the disease. A network is a graph where each node is a gene, and the gene behavior is described by boolean equations. The network simulations evaluate these equations several times throughout the execution. The network states' transitions are the basic step in gene regulatory algorithms. This paper proposes a study of the CPU, GPU and FPGA implementations of the basic operation which is the calculation of the next state. We explore OpenMP and AVX vectorization compiler-based approaches for processors. Moreover, we propose a new dynamic overlay architecture to simplify the use of FPGA solutions. In addition, we transform the Boolean equation in threshold logic. Finally, we evaluate 16 gene regulatory networks used in the literature. The experimental results show a speed-up from 3× to 90× with respect to the CPU times. The AVX/OpenMP, GPU and FPGA implementations report a speed-up of 3x, 57.3x, and 86.7x, respectively.*

***Resumo.*** *As redes reguladoras de genes são modelos baseados em grafos muito utilizadas para estudar o comportamento de células, processos de diferenciação celular ou tratamento e evolução de doenças. Uma rede pode ser implementada por um grafo com equações booleanas. Os algoritmos usados nas simulações das redes avaliam estas equações várias vezes ao longo da execução. Este artigo propõe um estudo das implementações em CPU, GPU e FPGA da operação básica que é o cálculo do próximo estado. Exploramos as técnicas de vetorização e paralelização com AVX e OpenMP para os processadores e uma nova arquitetura dinâmica é proposta para simplificar o uso das soluções com FPGA. Além do modelo booleano, mostramos como as redes podem ser transformadas em equações com somas de peso e limiares. Finalmente, 16 redes biológicas usados na literatura foram avaliadas, onde as implementações em CPU com OMP apresentaram uma aceleração de 3x em comparação com a CPU, as implementações em GPU foram em média 57,3x mais rápidas que a CPU e finalmente as implementações em FPGA foram em média 86,7x mais rápidas que a CPU.*

*Financiamento: FAPEMIG, PIBIC/CNPq, Nvidia, Funarbe. O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001. Apoio dos laboratórios: Intel Academic Compute Environment e Paderborn Center for Parallel Computing.

## 1. Introdução

A simulação em silício de modelos computacionais de sistemas biológicos vem sendo usada na produção de novos medicamentos e desenvolvimento de tratamentos para doenças como câncer [De Jong 2002]. Muitos problemas como, por exemplo, diferenciação celular [Aldana 2003] e morte celular programada [Haupt et al. 2003], podem ser modelados com equações booleanas, conhecidas pelo termo redes reguladoras de gene [Kauffman 1993, Davidson and Levin 2005, Kauffman 1969]. Essas redes são grafos direcionados, nos quais um vértice representa um gene, podendo ser ativo ou inativo. As arestas do grafo representam as interações entre os genes. O número de estados de uma rede de $N$ vértices é $2^N$, ou seja, para uma rede com 100 vértices, teremos um espaço de $2^{100} = 10^{30}$ estados a serem exploradas pelas simulações. Supondo um desempenho computacional capaz de avaliar 1 Tera estados por segundo, seriam necessários 36 bilhões de anos para explorar todo o espaço, o que motiva a busca por heurísticas e soluções eficientes. Buscando implementações com desempenho e eficiência energética, este trabalho propõe avaliar arquiteturas *multi-core*, GPU e FPGA para implementação eficiente da avaliação dos estados. Apesar da eficiência demonstrada pelos FPGAs para avaliação de redes de genes [Purandare et al. 2017, Pournara et al. 2005, da Silva et al. 2017], o tempo de compilação/síntese é longo (minutos a horas) e a complexidade das ferramentas de projeto restringem o uso de FPGAs. Neste trabalho, apresentamos uma alternativa simples com uma camada virtual sobre um FPGA comercial semelhante à abordagem proposta em [Ferreira and Vendramini 2010], transparente para o usuário que não necessita ter nenhum conhecimento de FPGAs e suas ferramentas. A arquitetura proposta juntamente com as outras soluções foram avaliadas para um conjunto de 16 redes de genes da literatura [UFV 2019].

A contribuição maior desde trabalho foi propor e comprovar que uma solução em FPGA dinâmica é competitiva com as soluções aceleradas com GPU. Além disso, os aceleradores com GPU e FPGA tem um desempenho bem superior aos processadores, da ordem de 50-90x, mesmo utilizando a vetorização de código ou paralelização gerada pelos compiladores com AVX e OpenMP.

Este artigo está estruturado da seguinte forma. A Seção 2 apresenta o problema do cálculo do próximo estado e o desempenho esperado nas plataformas avaliadas: CPU, GPU e FPGA. A Seção 3 introduz uma arquitetura reconfigurável sobre o FPGA em tempo de execução e mostra como ela foi derivada das redes biológicas e como se faz o mapeamento. A Seção 4 apresenta os resultados experimentais. A Seção 5 compara o trabalho com o estado da arte e, finalmente, a Seção 6 apresenta as conclusões e os trabalhos futuros.

## 2. Cálculo do Próximo Estado em uma Rede Reguladora

Uma rede reguladora pode ser modelada por um conjunto de equações booleanas [Aldana 2003]. A dinâmica da rede consiste em acompanhar a evolução dos estados. Dado um estado, o próximo estado é calculado avaliando todas as equações. A Figura 1(a) mostra um exemplo de grafo, no qual a cor preta significa ativo e a cor branca inativo. A Figura 1(b) ilustra as equações de cada vértice e a Figura 1(c) mostra o próximo estado da rede quando todas as equações são reavaliadas.

O cálculo do próximo estado é a operação básica para os algoritmos de

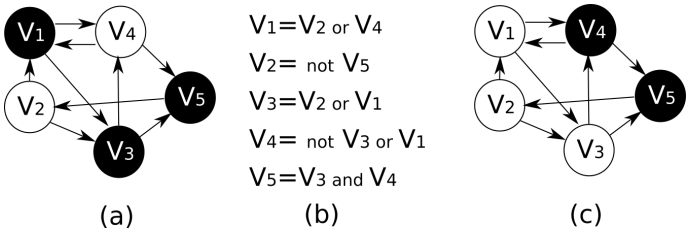


**Figura 1. (a) Rede Reguladora (estado Inicial); (b) Equações; (c) Próximo estado**

manipulação de redes reguladoras de genes [Aldana 2003]. As Seções irão estimar o custo computacional do cálculo de próximo estado para as implementações em CPU, GPU e FPGA, que serão posteriormente validados pelos experimentos.

### 2.1. CPU

Em uma CPU, o cálculo de um conjunto de equações gera uma sequência de instruções com operações booleanas, máscaras e deslocamento de bits. Por exemplo, para a rede reguladora Apoptosis com 78 equações booleanas são necessárias 458 instruções de máquina gerados pelo compilador GCC versão 5.4.0 compilado com -O3. Supondo um cenário ideal com frequência de 3 GHz, pipeline perfeito e 4 instruções por ciclo, uma CPU poderia avaliar 26 Mega estados por segundo. Supondo ainda uma versão *multi-core* ideal com 8 núcleos e um ganho ideal de 8x, o desempenho máximo poderá chegar à 208 Mega estados por segundo.

### 2.2. GPU

As GPUs atuais tipicamente possuem de 2.000 a 5.000 núcleos, podendo ter um desempenho de pico de 6 a 10 Tera instruções por segundo, executando com uma frequência de 1,6 GHz. Considerando a mesma rede reguladora Apoptosis com 78 equações, são necessárias 412 instruções de máquina geradas pelo compilador NVCC CUDA 10.1. Portanto, a GPU em um cenário ideal pode ter um desempenho de 7,7 Giga estados por segundo.

### 2.3. FPGA

Um FPGA tem a estrutura ideal para implementação de avaliação paralela de funções booleanas, como já for demonstrado por trabalhos anteriores [Pournara et al. 2005]. Entretanto, a implementação fica restrita a uma rede específica, o tempo de compilação é custoso (minutos/horas) e a complexidade das ferramentas e da integração do acelerador são grandes barreiras para popularizar o uso dos FPGAs [Pournara et al. 2005]. A implementação proposta em [da Silva et al. 2017] resolve o problema de entrada e saída de dados e acoplamento em plataformas de alto desempenho, porém, o tempo de compilação é da ordem de horas. A implementação proposta em [Ferreira and Vendramini 2010] elimina a etapa de compilação com uma camada virtual. Porém, apenas uma rede é avaliada por vez e os resultados apresentados foram com redes aleatórias. Este trabalho utiliza redes mapeadas por biólogos. Para ser competitivo com outras plataformas, além da virtualização proposta em [Ferreira and Vendramini 2010], a solução em FPGA deve processar várias cópias da rede e ser integrada em uma plataforma de computação de alto desempenho. Supondo uma frequência de relógio de 200 Mhz, a avaliação das equações em apenas 1 ciclo e 10 cópias, um FPGA poderá avaliar na ordem de 2 Giga estados por segundo.

## 3. Arquitetura Reconfigurável

Além do uso de CPU e GPU, este trabalho apresenta uma opção de virtualização em FPGA para modelagem de rede. Primeiro, a maioria das redes disponíveis na literatura são booleanas. Algumas equações são simples (2 a 4 variáveis) e podem ser implementadas por pequenas memórias reconfiguradas dinamicamente em uma camada virtual no FPGA. Entretanto, para equações de maior complexidade (9 ou mais variáveis), outras alternativas podem ser avaliadas. A solução proposta neste trabalho é transformar as equações com lógica limiar, permitindo assim a criação de operadores genéricos com soma de pesos nas unidades de processamento reconfiguráveis na camada virtual sobre o FPGA. Diferente de trabalhos anteriores que exploraram equações sintéticas com lógica limiar em GPUs [Jacob et al. 2012, Campos et al. 2011], a proposta deste trabalho é mapear modelos derivados de redes biológicas.

### 3.1. Lógica Limiar

**Tabela 1. Tabela verdade da Função $f = \overline{x_3} \cdot (x_1 \vee x_2)$ e pesos da Função Limiar.**

| | Entrada | | | Saída | Relação entre |
|---|---|---|---|---|---|
| linha | $x_1$ | $x_2$ | $x_3$ | y | P e L |
| 0 | 0 | 0 | 0 | 0 | 0 <L |
| 1 | 0 | 0 | 1 | 0 | $P_3$ <L |
| 2 | 0 | 1 | 0 | 1 | $P_2 \geq$ L |
| 3 | 0 | 1 | 1 | 0 | $P_2 + P_3$ <L |
| 4 | 1 | 0 | 0 | 1 | $P_1 \geq$ L |
| 5 | 1 | 0 | 1 | 0 | $P_1 + P_3$ <L |
| 6 | 1 | 1 | 0 | 1 | $P_1 + P_2 \geq$ L |
| 7 | 1 | 1 | 1 | 0 | $P_1 + P_2 + P_3$ <L |
| ocorrências($x_i$) | 2 | 2 | 0 | | |
| ocorrências($\overline{x_i}$) | 1 | 1 | 3 | | |

Nesta seção iremos ilustrar, com um simples exemplo, como uma função booleana pode ser transformada em lógica limiar (*Threshold Logic Function – TLF*). A função limiar requer o armazenamento apenas dos pesos no lugar de uma tabela completa. Uma função lógica limiar booleana utiliza um peso por variável e pode ser completamente representada por um vetor compacto $[p_1, p_2, ..., p_n; L]$. A função booleana limiar pode representar uma função booleana complexa, como $f = x_1x_2 \vee x_1x_3 \vee x_2x_3x_4 \vee x_2x_3x_5$ que corresponde com $f = [4, 3, 3, 1, 1; 7]$, que representa a equação $4x_1+3x_2+3x_3+x_4+x_5 \geq 7$. Para cada padrão de entrada, a função é 1 somente quando a soma for maior que (ou igual) ao valor do peso limite $L$ [Muroga 1971, Neutzling et al. 2013]. A função booleana limiar pode ser formalmente definida por:

$$f(x_1, x_2, \cdots, x_n) = \begin{cases} 1 & se \sum_{i=1}^{n} p_i \cdot x_i \geq L \\ 0 & \text{caso contrário} \end{cases} \quad (1)$$

Dada uma função $f(x_1, x_2, ..., x_n)$, o peso de cada variável é:

$$P_i = 2 * (Ocorrencias(x_i) - Ocorrencias(\overline{x_i})), \quad quando : f(x_i) = 1 \quad (2)$$

onde a função $Ocorrencias(x_i)$ irá contar quantas vezes a variável $x_i$ é verdadeira quando a função $f$ é verdadeira. A função $Ocorrencias(\overline{x_i})$ irá contar quando a variável é falsa. Por exemplo, para a função Booleana $f = \overline{x_3}(x_1 \vee x_2)$ na Tabela 1, a função $f$ é verdadeira nas linhas 2, 4 e 6. Nestas linhas a variável $x_1$ é verdadeira 2 vezes (linhas 4 e 6) e falsa uma vez (linha 2), portanto $P_1 = 2 * (Ocorrencias(x_1) - Ocorrencias(\overline{x_1})) = 2 * (2 - 1) = 2$. As duas últimas linhas da Tabela 1 mostram as ocorrências de $x_i$ verdadeiro e falso para $f$.

A última coluna da Tabela 1 representa a relação que tem os pesos ($P$) e o limite ($L$) da função limiar. O valor de $L$ é obtido solucionando as inequações. Para facilitar o cálculo, apenas as linhas 2, 4 e 6 onde a função $f$ é verdadeira podem ser usadas, calculando o mínimo do somatório dos pesos $P_n$ no lado esquerdo das inequações em cada uma destas linhas, definido-se como:

$$L = \min_{j=1}^{linhas} (\sum_{i=1}^{n} P_i), \quad \forall \quad j \quad \text{onde} \quad f(x) = 1 \tag{3}$$

Para a função $f(x_1, x_2, ..., x_n)$ com as três condições e os valores de $P_2 = 2$, $P_1 = 2$ e $P_1 + P_2 = 4$, o valor mínimo é 2, ou seja $L = 2$, então a função limiar será $f = [2, 2, -6; 2]$. Na linha 4, temos $2x_1 + 2x_2 - 6x_3 = 2 * 1 + 2 * 0 - 6 * 0 = 2 \geq 2$, portanto $f(4 = 100_2) = 1$.

### 3.2. Interconexões

Além do cálculo local das equações, a camada virtual precisa implementar a comunicação dos vértices de forma eficiente e flexível. Neste trabalho usamos uma abordagem quantitativa para definir a estrutura de interconexão da rede. Um conjunto de 16 redes biológicas foi primeiro categorizado para avaliar a distribuição do grau de entrada e do grau de saída de cada vértice. Este estudo teve o objetivo de modelar uma arquitetura que pudesse mapear os diferentes grafos dinamicamente, uma vez que o grau de entrada e saída das unidades de processamento do acelerador devem ser definidas em tempo de projeto.

A Tabela 2 enumera as 16 redes utilizadas neste trabalho. Maiores detalhes sobre as redes e como foram modeladas estão disponíveis no material complementar [UFV 2019].

A Figura 2 mostra o histograma de distribuição do grau de entrada dos vértices para as redes avaliadas. Podemos observar que existe uma grande quantidade de vértices com um grau de entrada pequeno (0, 1 ou 2), onde $11,47 + 38,20 + 19,28 = 68,95\%$ dos vértices tem grau de entrada inferior a 3, como já mencionado na literatura da área [Aldana 2003]. Duas arquiteturas foram derivadas baseadas no histograma do grau de entrada dos vértices.

A Tabela 3 apresenta a distribuição do grau de entrada para a $Arq_1$ com uma rede de 1.024 conexões e para a $Arq_2$ com 256 conexões. Por exemplo, a $Arq_1$ possui 34 vértices com grau zero (o que corresponde a 8.5% dos vértices), que é próximo do valor médio de 11% apontado pelos dados da Figura 2. Estes vértices irão implementar genes que permanecem constantes durante a evolução dos estados. São 183 vértices (ou 46% dos vértices) com grau 1 para $Arq_1$ e 59 (47%) para a $Arq_2$.

**Tabela 2. As 16 Redes Reguladores usadas nos Experimentos**

| Nome | G | Nome | G |
|---|---|---|---|
| Cholesterol Regulatory Pathway | 1 | Bortezomib R in U266 Human Myelo | 9 |
| Apoptosis Network | 2 | HGF Signaling in Keratinocytes | 10 |
| Guard Cell Abscisic Acid Signaling | 3 | Glucose Repression Signaling 2009 | 11 |
| Differentiation of T lymphocytes | 4 | Yeast Apoptosis | 12 |
| B Bronchiseptica and T R Coinfection | 5 | Lymphopoiesis Regulatory Network | 13 |
| MAPK Cancer Ce | 6 | IL-6 Signalling | 14 |
| T-LGL Survival Network 2011 | 7 | EGFR & ErbB Signaling | 15 |
| PC12 Cell Differentiation | 8 | Signal Transduction in Fibroblasts | 16 |

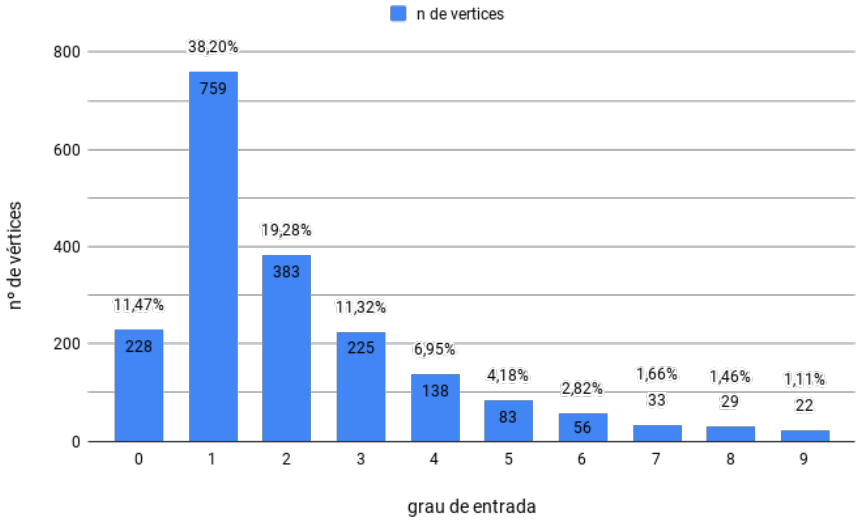


**Figura 2. Histograma do grau de entrada das 16 redes reguladoras.**

A Figura 3 ilustra um exemplo de uma arquitetura que pode suportar duas redes. A primeira rede tem 2 vértices com grau 1 (cor branca), 4 vértices com grau 2 (cor preta) e 1 vértice com grau 3 (cor cinza). A segunda rede tem 3, 2 e 2 vértices com grau 1, 2 e 3, respectivamente. A Figura 3(c) ilustra um exemplo de uma arquitetura que suporta o mapeamento das duas redes com 2, 3 e 2 unidades que suportam 1, 2 e 3 entradas, respectivamente. Podemos observar que um vértice de duas entradas é mapeado em uma unidade de três entradas para rede 1, pois não tem mais unidades de duas entradas livres. O mesmo ocorre para rede 2 que tem um vértice de uma entrada mapeado em uma unidade de duas entradas. O posicionamento dos vértices busca alocar unidades com o grau igual ou superior. Caso não seja possível, um vértice será alocado em uma unidade com grau menor e dois ou mais ciclos de relógio são necessários para realizar a avaliação das equações.

### 3.3. Mapeamento das Redes Biológicas

A Tabela 4 caracteriza as 16 redes reguladoras de genes avaliadas. As colunas Rede, v e arestas mostram o identificador da rede (correspondente a Tabela 2), o número

**Tabela 3. Distribuição do grau de entrada para as arquiteturas reconfiguráveis.**

| Arquitetura | Rede | Unidades | Grau | 0 | 1 | 2 | 3 | 4 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| $Arq_1$ | 1.024 | 395 | Qte | 34 | 183 | 78 | 20 | 29 | 51 |
| $Arq_2$ | 256 | 125 | Qte | 14 | 59 | 23 | 14 | 8 | 7 |

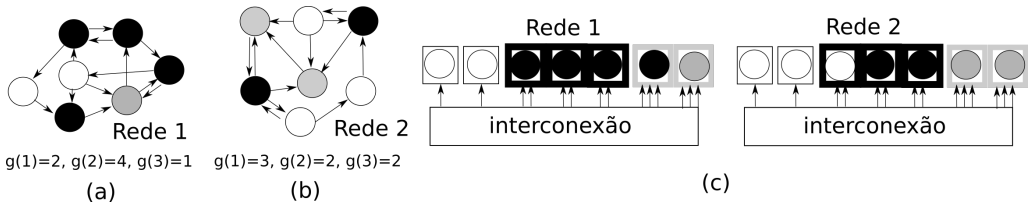


**Figura 3. (a) Rede 1; (b) Rede 2; (c) Mapeamento na Arquitetura Personalizada.**

de vértices e o número de arestas, respectivamente. A segunda, terceira, quarta e quinta coluna mostram a quantidade de vértices agrupados pelo tamanho do grau de entrada. A coluna $0-1$ mostra o total de vértices com grau de entrada 0 ou 1. Podemos observar que a maioria dos vértices possui grau de entrada igual ou inferior a três. Entretanto, como as redes reguladoras de genes são grafos livre de escala [Aldana 2003], existe a presença de *hubs* que possuem um grau alto de entrada mesmo para um grafo pequeno como as redes 4 e 5, onde encontramos vértices com mais de 8 entradas.

A Tabela 4 mostra também os resultados do mapeamento das redes nas arquiteturas $Arq_1$ e $Arq_2$, onde a sub-coluna $1$ indica se é possível mapear todas as arestas para executarem sincronicamente em um único ciclo de relógio. Como $Arq_1$ suporta 1.024 arestas é possível mapear todas as redes. Já para $Arq_2$, devido aos vértices de alto grau, não é possível por exemplo mapear as redes 4 e 7 com um ciclo, são necessários pelo menos 2 ciclos. O impacto no desempenho é significativo, pois irá ser reduzido à metade. A coluna $cp$ mostra quantas cópias podem ser simultaneamente mapeadas na arquitetura. Por exemplo, a rede 1 que tem 34 vértices pode ter duas cópias mapeadas na $Arq_2$ que possuim 256 conexões e 125 unidades. Poderíamos até mapear três cópias, porém para reduzir o custo da rede, utilizamos uma rede bloqueante multiestágio como proposto em [Ferreira and Vendramini 2010], onde para a Rede 1 só foi possível rotear duas cópias sem conflito. A redução do custo da rede de interconexões irá permitir instanciar mais de uma acelerador nas arquiteturas. Portanto, além de $n$ cópias por arquitetura, podemos ter $m$ instâncias da arquitetura a 200 Mhz, gerando um desempenho de pico de $n \cdot m \cdot 200$ Mega estados por segundo.

## 4. Resultados

Todos os experimentos estão disponíveis no material complementar [UFV 2019]. O primeiro experimento mostra o custo em instruções em linguagem de montagem necessários para avaliar o conjunto de equações para cada umas das arquiteturas alvo. Na Tabela 5, as colunas $BooCPU$ e $LimCPU$ mostram o número de instruções para cada conjunto de equações executadas no processador Intel(R) Xeon(R) CPU E5-2630v3 2.4 GHZ com 8 núcleos. O código foi compilado com GCC versão 5.4.0. com -O3 para as colunas $CPU$. A coluna $AVX$ apresenta o total de instruções quando o código é vetorizado automaticamente pelo compilador GCC com as otimizações O3, *unroll-loops*, *omit-*

**Tabela 4. Caracterização dos Grafos das Redes Reguladores.**

| | | Grau de Entrada | | | | Total | $Arq_1$ | | | | $Arq_2$ | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Rede | v | 0-1 | 2-3 | 4-7 | 8+ | Arestas | cp | 1 | 2 | 3 | cp | 1 | 2 | 3 |
| 1 | 34 | 26 | 8 | - | - | 43 | 8 | x | | | 2 | x | | |
| 2 | 41 | 23 | 13 | 5 | - | 73 | 5 | x | | | 1 | x | | |
| 3 | 44 | 27 | 11 | 6 | - | 78 | 5 | x | | | 1 | x | | |
| 4 | 50 | 30 | 11 | 8 | 1 | 97 | 4 | x | | | 1 | - | x | |
| 5 | 53 | 15 | 28 | 9 | 1 | 135 | 2 | x | | | 1 | x | | |
| 6 | 55 | 26 | 22 | 7 | - | 103 | 3 | x | | | 1 | x | | |
| 7 | 60 | 6 | 30 | 24 | - | 195 | 2 | x | | | 1 | - | x | |
| 8 | 62 | 31 | 30 | 1 | - | 108 | 5 | x | | | 1 | x | | |
| 9 | 67 | 35 | 22 | 10 | - | 130 | 1 | x | | | 1 | x | | |
| 10 | 68 | 41 | 25 | 2 | - | 103 | 2 | x | | | 1 | x | | |
| 11 | 73 | 40 | 32 | 1 | - | 106 | 3 | x | | | 1 | x | | |
| 12 | 73 | 55 | 11 | 5 | 2 | 112 | 4 | - | x | | 1 | - | x | |
| 13 | 81 | 47 | 19 | 13 | 2 | 158 | 2 | x | | | 1 | - | x | |
| 14 | 87 | 45 | 30 | 12 | - | 149 | 3 | x | | | 1 | x | | |
| 15 | 104 | 56 | 33 | 6 | 9 | 227 | 2 | - | x | | 1 | - | x | |
| 16 | 139 | 30 | 39 | 53 | 17 | 546 | 1 | - | x | | 0 | - | - | - |

*frame-pointer*, *inline*, com as opções *arch=native*, *tune=native*, *no-zero-upper* e com o alvo AVX tanto para as equações booleanas quanto para as equações com limiares. A coluna GPU mostra as instruções aferidas com a ferramenta *nvprof* na versão CUDA 10.1 em uma GPU GTX 1070 da Nvidia com 1.920 núcleos e 1,683 GHz.

Para comparar o desempenho das arquiteturas de CPU, GPU e FPGA realizamos um segundo experimento. Para cada rede um conjunto de 10 milhões de estados foi avaliado. O tempo de execução das soluções com CPU (incluindo OpenMP e AVX) foi aferido com a biblioteca *chrono*. O tempo de execução da GPU foi medido com nvprof CUDA versão 10.1.168. O tempo de execução do FPGA foi medido para uma frequência de 200 MHz. Ademais, foram instanciados 5 e 20 aceleradores das $Arq_1$ e $Arq_2$, respectivamente. Um acelerador, dependendo do tamanho do grafo, pode mapear mais de uma cópia do grafo.

A Tabela 6 mostra o número de estados avaliados em Mega estados por segundo. A coluna $R$ mostra o identificador da rede reguladora. Podemos observar que o pior desempenho é o da CPU com apenas 1 *thread* nas colunas $BooCPU$ e $LimCPU$. A versão booleana é ligeiramente mais rápida que a versão limiar. As duas últimas linhas da Tabela 6 mostram o ganho de desempenho de todas as implementações em relação a versão CPU booleana. Foram calculadas as médias geométricas dos ganhos de desempenho (linha G) e a média aritmética dos ganhos de desempenho (linha A). A versão limiar é 30% pior que a booleana. Entretanto, a versão limiar pode ser facilmente adaptada para ter um comportamento dinâmico no qual pode-se explorar variações nos pesos para ajustar uma rede biológica que está sendo modelada e comparada com dados experimentais [Davidson and Levin 2005]. A versão limiar é vetorizada pelo compilador e tem um desempenho semelhante à versão booleana sem vetorização, enquanto a versão

**Tabela 5. Número de instruções geradas.**

| Rede | Número de instruções | | | | | |
|---|---|---|---|---|---|---|
| | Ares-tas | Boo CPU | Lim CPU | Lim AVX | Boo GPU | Lim GPU |
| 1 | 43 | 185 | 255 | 84 | 195 | 590 |
| 2 | 73 | 384 | 419 | 162 | 356 | 783 |
| 3 | 78 | 458 | 392 | 135 | 412 | 791 |
| 4 | 97 | 418 | 473 | 184 | 289 | 1.080 |
| 5 | 135 | 584 | 678 | 262 | 635 | 1.305 |
| 6 | 103 | 493 | 590 | 213 | 502 | 1.116 |
| 7 | 195 | 630 | 872 | 351 | 432 | 1.396 |
| 8 | 108 | 1.693 | 499 | 203 | 557 | 831 |
| 9 | 130 | 647 | 575 | 248 | 591 | 1.241 |
| 10 | 103 | 445 | 453 | 238 | 563 | 1.308 |
| 11 | 106 | 464 | 356 | 186 | 650 | 1.016 |
| 12 | 112 | 615 | 435 | 233 | 785 | 1.204 |
| 13 | 158 | 664 | 623 | 322 | 618 | 1.724 |
| 14 | 149 | 623 | 502 | 229 | 761 | 1.270 |
| 15 | 227 | 914 | 584 | 291 | 946 | 1.839 |
| 16 | 546 | 2.387 | 2.063 | 997 | 2.612 | 6.303 |

booleana não foi automaticamente vetorizada pelo compilador GCC. No segundo grupo de implementações temos a paralelização do cálculo com OMP para 8 *threads*. Podemos ver um ganho médio de 3x tanto na média aritmética, quanto na geométrica. Finalmente, a versão GPU apresenta um ganho de 75-80x com a versão booleana e de 50x com a versão limiar. Vale ressaltar novamente que a versão limiar pode ser ajustada em tempo de execução. O acelerador em FPGA tem um desempenho com números exatos pois realiza o cálculo em um ciclo de relógio na frequência de 200 Mhz com $n \cdot m$ cópias, onde $n$ é número de instâncias por arquitetura e $m$ é número de cópias que podem ser implementadas na camada virtual do FPGA. Além do desempenho similiar, o acelerador virtual em FPGA tem uma maior eficiência energética pois consome 6x menos energia que a GPU.

O gráfico da Figura 4 apresenta a comparação de desempenho das implementações. A escala de desempenho é logarítmica, quanto maior, melhor o resultado. Podemos observar três grupos em desempenho. O primeiro grupo formado pelas soluções em GPU e FPGA, seguindo pelo grupo das implementações com OpenMP e com o desempenho pior as implementações em CPU, mesmo com a vetorização gerada pelo compilador GCC.

## 5. Trabalhos Relacionados

Nesta seção são apresentados os trabalhos relacionados com as rede reguladoras. Esses trabalhos demonstram que esse tema vem sendo estudado nas últimas décadas [Kauffman 1969, Kauffman 1993] e ainda possui relevância na área de pesquisa. Contudo, a maioria dos trabalhos não detalha e nem disponibiliza as implementações.

A maioria das redes é descrita por equações booleanas [De Jong 2002,

**Tabela 6. Cálculo de Estados nas diversas plataformas: CPU, GPU, FPGA.**

| R | Mega Estados por Segundo | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Boo CPU | Lim CPU | Boo OMP | Lim OMP | Lim AVX OMP | Lim AVX | Boo GPU | Lim GPU | $Arq_1$ | $Arq_2$ |
| 1 | 74 | 49 | 174 | 197 | 174 | 76 | 8.569 | 2.685 | 8.000 | 8.000 |
| 2 | 37 | 25 | 101 | 103 | 133 | 48 | 3.625 | 2.644 | 5.000 | 4.000 |
| 3 | 28 | 31 | 95 | 112 | 87 | 55 | 3.591 | 2.370 | 5.000 | 4.000 |
| 4 | 24 | 20 | 75 | 82 | 67 | 23 | 3.989 | 1.890 | 2.000 | 2.000 |
| 5 | 21 | 17 | 68 | 51 | 51 | 31 | 2.221 | 1.569 | 2.000 | 2.000 |
| 6 | 25 | 20 | 97 | 60 | 60 | 37 | 3.090 | 1.871 | 3.000 | 4.000 |
| 7 | 17 | 13 | 51 | 40 | 59 | 24 | 3.760 | 1.436 | 1.000 | 1.333 |
| 8 | 58 | 17 | 194 | 74 | 74 | 24 | 2.251 | 1.915 | 2.500 | 4.000 |
| 9 | 20 | 18 | 78 | 75 | 96 | 18 | 945 | 907 | 1.000 | 2.000 |
| 10 | 30 | 17 | 86 | 79 | 72 | 17 | 930 | 860 | 1.000 | 2.000 |
| 11 | 26 | 19 | 98 | 78 | 83 | 20 | 857 | 898 | 1.500 | 1.333 |
| 12 | 21 | 18 | 83 | 60 | 60 | 18 | 546 | 635 | 2.000 | 1.333 |
| 13 | 17 | 13 | 56 | 42 | 42 | 14 | 701 | 617 | 2.000 | 2.000 |
| 14 | 20 | 17 | 64 | 74 | 74 | 18 | 923 | 680 | 1.500 | 1.333 |
| 15 | 13 | 15 | 61 | 77 | 51 | 15 | 693 | 370 | 1.000 | 1.333 |
| 16 | 4 | 5 | 18 | 26 | 26 | 5 | 275 | 171 | 333 | - |
| A | 1,0 | 0,71 | 3,2 | 2,80 | 2,76 | 1,00 | 67,6 | 46,1 | 79,5 | 94,2 |
| G | 1,0 | 0,76 | 3,4 | 2,99 | 2,98 | 1,00 | 84,3 | 49,3 | 88 | 91,5 |

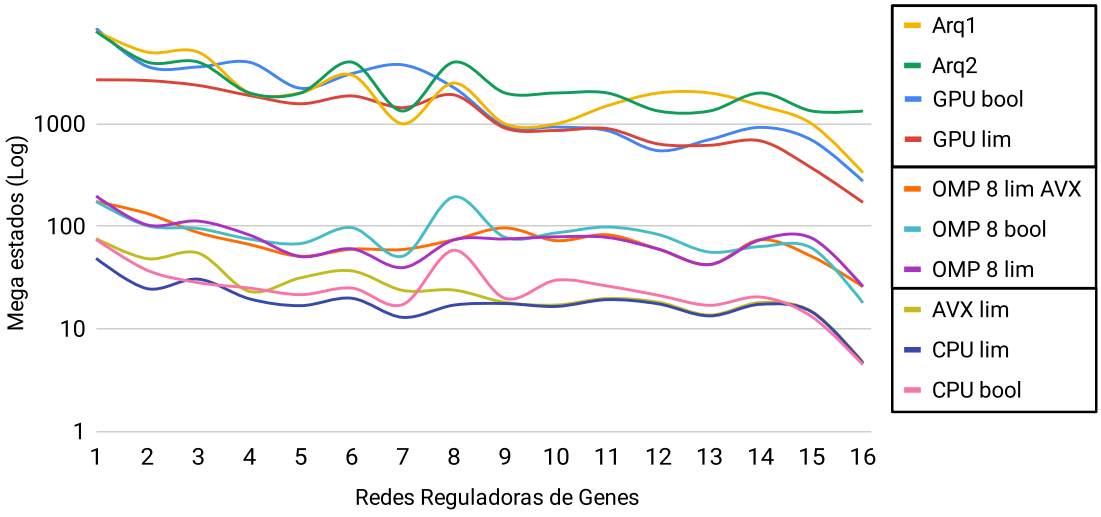


**Figura 4. Comparação de Desempenho medido em Mega Estados por Segundo para as implementações em CPU, GPU e FPGA.**

Davidson and Levin 2005]. Alguns trabalhos propõem o uso de lógica limiar [Darabos et al. 2011], ou exploram propriedades de grafos livres de escala nos quais alguns vértices concentram a maioria das ligações [Aldana 2003]. A abordagem pro-

posta em [Borelli et al. 2013] implementa um acelerador em GPU para as redes com aceleração de 50-200 vezes, porém não foram avaliadas redes biológica e sim redes aleatórias sintéticas geradas para os testes. A abordagem proposta em [Trinh et al. 2014] propõe uma técnica para analisar a robustez das redes e a estrutura de retroalimentação das redes em CPU e GPU, mostrando um ganho de 2x na versão implementada em GPU.

Já nas abordagens propostas com FPGA, o maior gargalo é o tempo de compilação e a complexidade do projeto que é específico para cada rede [Purandare et al. 2017, Pournara et al. 2005, da Silva et al. 2017]. Apesar da solução apresentada em [Ferreira and Vendramini 2010] utilizar uma camada virtual sobre o FPGA, o sistema só foi avaliado com redes sintéticas e só avalia uma cópia da rede por vez.

Este trabalho difere dos anteriores ao explorar três arquiteturas para implementação da operação básica, o cálculo do próximo estado. Trabalhos futuros poderão avaliar o desempenho para execução de algoritmos que envolverão a evolução dos estados [Purandare et al. 2017, da Silva et al. 2017, Trinh et al. 2014, Borelli et al. 2013].

## 6. Conclusão

Este artigo apresenta uma comparação de desempenho para avaliação de estados de redes reguladoras de genes em três arquiteturas: CPU, GPU e FPGA. Apesar da paralelização e vetorização dos processadores com múltiplos núcleos com diretivas de compilação AVX e pragmas em OpenMP, o ganho é de apenas 3x mesmo com a disponibilidade de 8 núcleos. A GPU oferece um alto desempenho com ganhos de até 80x em relação à versão CPU. A opção com equações limiares no lugar de booleanas tem uma perda de desempenho mas pode ser útil em cenários nos quais os pesquisadores estão ajustando os pesos dinamicamente sem a necessidade de recompilação. A implementação em FPGA com a virtualização elimina o problema de recompilar as redes para mapear em um FPGA, oferece desempenho em relação às GPUs e é transparente para o usuário que não precisa de conhecimento do fluxo de desenvolvimento de aplicações em FPGA. Ademais, neste artigo avaliamos um cenário ideal para GPU com a mesma carga e operações para todos os *threads*. Em algoritmos de redes reguladores como o cálculo de atratores [da Silva et al. 2017], a carga é variável resultando em uma perda de desempenho, já as implementações em FPGA ou CPU não são afetadas pela carga variável. Trabalhos futuros incluem a execução de algoritmos como o cálculo de atratores, comportamento de variáveis [Fumia and Martins 2013], dentre outros.

## Referências


Aldana, M. (2003). Boolean dynamics of networks with scale-free topology. *Physica D: Nonlinear Phenomena*, 185(1):45–66.

Borelli, F. F., de Camargo, R. Y., Martins, D. C., and Rozante, L. C. (2013). Gene regulatory networks inference using a multi-gpu exhaustive search algorithm. *BMC bioinformatics*, 14(18):S5.

Campos, R. R., Ferreira, R., Vendramini, J. C. G., Martins, M. L., et al. (2011). Simulation of scale free gene regulatory networks based on threshold functions on gpu. In *Simposio de Sistemas Computacionais de Alto Desempenho (WSCAD)*. IEEE.

da Silva, L. B., Almeida, D., Nacif, J. A. M., Sánchez-Osorio, I., Hernández-Martínez, C. A., and Ferreira, R. (2017). Exploring the dynamics of large-scale gene regulatory networks using hardware acceleration on a heterogeneous cpu-fpga platform. In *IEEE Int. Conf. on ReConFigurable Computing and FPGAs (ReConFig)*.

Darabos, C., Di Cunto, F., Tomassini, M., Moore, J. H., Provero, P., and Giacobini, M. (2011). Additive functions in boolean models of gene regulatory network modules. *PloS one*, 6(11):e25110.

Davidson, E. and Levin, M. (2005). Gene regulatory networks. *Proceedings of the National Academy of Sciences*, 102(14):4935–4935.

De Jong, H. (2002). Modeling and simulation of genetic regulatory systems: a literature review. *Journal of computational biology*, 9(1):67–103.

Ferreira, R. and Vendramini, J. (2010). Fpga-accelerated attractor computation of scale free gene regulatory networks. In *IEEE Field Programmable Logic and Applications FPL*.

Fumia, H. F. and Martins, M. L. (2013). Boolean network model for cancer pathways: predicting carcinogenesis and targeted therapy outcomes. *PloS one*, 8(7):e69008.

Haupt, S., Berger, M., Goldberg, Z., and Haupt, Y. (2003). Apoptosis-the p53 network. *Journal of cell science*, 116(20):4077–4085.

Jacob, V. V., de Resende Ferreira, C., and Ferreira, R. (2012). Gpu optimization techniques applied to scale free gene regulatory networks based on threshold function. In *Simposio de Sistemas Computacionais de Alto Desempenho (WSCAD)*, pages 57–64. IEEE.

Kauffman, S. A. (1969). Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of theoretical biology*, 22(3):437–467.

Kauffman, S. A. (1993). *The origins of order: Self-organization and selection in evolution*. OUP USA.

Muroga, S. (1971). Threshold logic and its applications.

Neutzling, A., Martins, M., Ribas, R., and Reis, A. (2013). An efficient method to threshold logic functions identification. In *SOUTH SYMP. ON MICROELECTRONICS*.

Pournara, I., Bouganis, C.-S., and Constantinides, G. A. (2005). Fpga-accelerated bayesian learning for reconstruction of gene regulatory networks. In *Field Programmable Logic and Applications (FPL)*. IEEE.

Purandare, M., Polig, R., and Hagleitner, C. (2017). Accelerated analysis of boolean gene regulatory networks. In *IEEE Field Programmable Logic and Applications (FPL)*.

Trinh, H.-C., Le, D.-H., and Kwon, Y.-K. (2014). Panet: a gpu-based tool for fast parallel analysis of robustness dynamics and feed-forward/feedback loop structures in large-scale biological networks. *PloS one*, 9(7).

UFV (2019). Redes reguladoras. `https://github.com/Wallace-F-Rosa/redes_reg_genes`.