# Aplicação de Evolução Diferencial em GPU Para o Problema de Predição de Estrutura de Proteínas com Modelo 3D AB Off-Lattice

**André E. P. Dias[1], Mateus Boiani[2], Rafael S. Parpinelli[1]**

[1]Departamento de Ciência da Computação / Mestrado em Computação Aplicada
Universidade do Estado de Santa Catarina (UDESC) – Joinville – SC – Brasil

[2]Instituto de Informática
Universidade Federal do Rio Grande do Sul (UFRGS) – Porto Alegre – RS – Brasil

andreeduardo1997@gmail.com, mateus.boiani@inf.ufrgs.br

rafael.parpinelli@udesc.br

***Abstract.*** *The function that a protein performs is directly related to its three-dimensional structure. However, for most of the proteins currently sequenced, their native structural form is not yet known. This article proposes using the Differential Evolution (DE) algorithm developed on the NVIDIA CUDA platform applied to the 3D AB Off-Lattice model for protein structure prediction. A niche and crowding strategy is added in the DE algorithm combined with parameter self-tuning techniques, routines for resetting the population, two levels of optimization, and local search. Four real proteins were used for experimentation, and the results obtained are competitive with state-of-the-art algorithms. The use of GPU's massive parallelism highlights its applicability to this class of problems, reaching accelerations of 708.78x for the largest protein chain.*

***Resumo.*** *A função que uma proteína exerce está diretamente relacionada com a sua estrutura tridimensional. Porém, para a maior parte das proteínas atualmente sequenciadas ainda não se conhece sua forma estrutural nativa. Este artigo propõe a utilização do algoritmo de Evolução Diferencial (DE) desenvolvido na plataforma NVIDIA CUDA aplicado ao modelo 3D AB Off-Lattice para Predição de Estrutura de Proteínas. Uma estratégia de nichos e crowding foi implementada no algoritmo DE combinada com técnicas de autoajuste de parâmetros, rotinas para reinicialização da população, dois níveis de otimização e busca local. Quatro proteínas reais foram utilizadas para experimentação e os resultados obtidos se mostram competitivos com o estado-da-arte. A utilização de paralelismo massivo através da GPU ressalta a aplicabilidade desses recursos a esta classe de problemas atingindo acelerações de 708.78x para a maior cadeia proteica.*

## 1. Introdução

Proteínas são macromoléculas formadas por cadeias de aminoácidos encontradas em todos organismos biológicos, sendo considerada a base da vida celular e molecular. São responsáveis por diversas funções celulares, como transporte, regulação, proteção, mobilidade, suporte estrutural, etc. [Alberts B 2002]. A função que uma proteína realiza está associada com sua estrutura tridimensional.

Esforços em biologia computacional buscam descobrir a estrutura de proteínas utilizando informação de estruturas já conhecidas ou as propriedades químico-física presentes na formação das mesmas. O método *ab initio* propõe um modelo matemático que representa a energia livre de possíveis conformações de proteínas a partir da sua cadeia de aminoácidos e suas propriedades físico-químicas. O modelo AB Off-Lattice, considera a propriedade de hidrofobicidade dos aminoácidos de maneira que os aminoácidos hidrofóbicos tendem a ficar no centro das proteínas e os hidrofílicos na parte externa [Jana and Sil 2018]. Nesse modelo o ângulo entre aminoácido adjacentes é considerado. Possui duas variantes: no plano de duas dimensões e três dimensões. A última é conhecida como 3D AB Off-Lattice e se aproxima mais das proteínas do mundo real e é o modelo abordado neste trabalho.

Vale ressaltar que todos os modelos de abstração do problema de predição de estrutura de proteínas, são considerados problemas da classe NP-Difícil [Fraenkel 1993]. Por se tratar de um problema desta natureza, não é possível encontrar um algoritmo exato que resolva o problema em tempo computacional viável. Portanto, algoritmos de aproximação são comumente adotados e, dentre estes, as meta-heurísticas bio-inspiradas se destacam [Siarry 2016]. Dentre estas, o algoritmo de Evolução Diferencial (DE) vem se mostrando bastante eficiente [Opara and Arabas 2019].

Pelo fato do DE ser um algoritmo populacional, ele é naturalmente indicado para desenvolvimento em ambientes que propiciem aplicação de técnicas de paralelismo. Nele, um conjunto de soluções é modificada e avaliada de maneira independente a cada iteração [Opara and Arabas 2019]. A plataforma CUDA desenvolvida pela NVIDIA permite escrever aplicações que executam em dispositivos massivamente paralelos chamados de Unidades de Processamento Gráfico de Propósito Geral (GPGPUs) [Soyata 2018].

Na literatura, diversas meta-heurísticas foram propostas para otimizar o modelo 3D AB Off-Lattice. Os autores [Bošković and Brest 2016] propuseram o $DE_{PFO}$, que consiste no algoritmo DE com autoajuste de parâmetros (jDE) e com técnica de reinicialização global para poder contornar a perda de diversidade. Em [Bošković and Brest 2018], o algoritmo $DE_{LSRC}$ aperfeiçoa a rotina de reinicialização e adiciona um método de busca local que desloca aminoácidos e preserva o restante da conformação da proteína. Em [Bošković and Brest 2019], o algoritmo $DE_{2L}$ elabora uma função objetivo auxiliar que compacta as conformações em núcleos hidrofóbicos. O algoritmo cuHjDE-3D proposto em [Boiani and Parpinelli 2020] constrói um otimizador híbrido do jDE com o algoritmo de busca local Hookie-Jeeves, onde o jDE é executado em GPU e alcança speedup de até 273x. A meta-heurística de Otimização por Enxame de Abelhas (ABC) também se mostrou popular na literatura [Li et al. 2015, Li et al. 2018] mas com resultados menos relevantes quando comparado aos trabalhos mais recentes.

Este trabalho explora o potencial dos dispositivos gráficos com a execução do otimizador em GPU a partir da programação em CUDA C++. O presente trabalho implementa em GPU uma variante do algoritmo DE conhecida como DSMDE (*Dynamic Speciation-based Mutation Strategies*) que divide a população em nichos [Deng et al. 2019]. Uma estratégia de *crowding* também é adotada, aumentando a competição entre indivíduos e mantendo a diversidade de soluções. O algoritmo proposto, chamado de cuDSMjDE, também faz uso de rotinas para autoajuste de parâmetros, reinicialização da população, dois níveis de otimização e busca local.

O trabalho está organizado da seguinte maneira. A próxima seção apresenta a fundamentação teórica com o contexto biológico do problema, o detalhamento do modelo 3D AB Off-Lattice, conceitos do DSMDE, jDE e GPU. A abordagem proposta para o problema se dá na seção 3. Os experimentos, resultados e análises são apresentados na seção 4. Por fim, as considerações finais e sugestões para trabalhos futuros se encontram na seção 5.

# 2. Fundamentação Teórica

## 2.1. Aminoácidos, Proteínas e o Modelo 3D AB Off-Lattice

Proteínas são compostos orgânicos construídos por uma cadeia de aminoácidos. Um aminoácido é constituído de um conjunto de átomos do grupo carboxílico (COOH), um grupo amina ($NH_2$) e um grupo R conhecido como cadeia lateral. Esse último grupo é responsável diferenciar cada um dos aminoácidos e fornecer suas características físico-químicas próprias como hidrofobia ou hidrofilia, carregamento elétrico positivo ou negativo, reatividade, entre outras [Alberts B 2002]. Todos os grupos de um aminoácido estão ligados em um átomo de carbono central, chamado de carbono alfa ($C_\alpha$). Dois aminoácidos se conectam para formar uma ligação peptídica. Isso acontece quando o grupo carboxílico de um aminoácido se conecta com o grupo amina de outro [Jana and Sil 2018].

O modelo 3D AB Off-lattice parte da premissa que o principal fator para a formação da estrutura de uma proteína são as interações de hidrofobicidade entre seus aminoácidos [Stillinger et al. 1993], caracterizando-os como A os hidrofóbicos e B os hidrofílicos. Este modelo considera a ligação entre os aminoácidos com uma unidade de comprimento e um par de ângulos de ligação e torção.

A posição de cada aminoácido de uma possível conformação de uma proteína é dada pela Equação 1 onde $i$ representa um aminoácido, $L$ o número total de aminoácidos que compõe a proteína, $\theta$ o ângulo de ligação entre dois aminoácidos e $\beta$ o ângulo de torção. O intervalo de valores que um ângulo $\theta$ ou $\beta$ pode assumir é $(-180^o, 180^o]$.

$$p_i = \begin{cases} (0,0,0), & \text{if i = 1} \\ (0,1,0), & \text{if i = 2} \\ (cos(\theta_1), sin(\theta_1)+1, 0), & \text{if i = 3} \\ \big(x_{i-1} + cos(\theta_{i-2}) \cdot cos(\beta_{i-3}), & \\ y_{i-1} + sin(\theta_{i-2}) \cdot cos(\beta_{i-3}), & \\ z_{i-1} + sin(\beta_{i-3})\big), & \text{if } 4 \leq \text{i} \leq \text{L} \end{cases} \quad (1)$$

O ângulo de ligação entre os dois primeiros aminoácidos é sempre $90^o$ e portanto, é desconsiderado. Como os três primeiros aminoácidos estão no plano $z = 0$, estes não possuem ângulo de torção. Por isso, o número total de ângulos que constitui uma conformação neste modelo é $L \times 2 - 5$. Uma possível conformação é representada pelo vetor $\{\theta_1, \theta_2, ..., \theta_{L-2}, \beta_1, \beta_2, ..., \beta_{L-3}\}$. Então, a estrutura nativa de uma proteína pelo modelo 3D AB *Off-Lattice* é representada pela combinação de ângulos que minimiza a função de energia da Equação 2. A equação do modelo é composta por dois termos onde o primeiro (Equação 3) é referente ao custo energético para dobrar a proteína e o segundo (Equação 4) pelas relações intramoleculares. O segundo termo apresenta um

fator de atratividade $c$ que é igual a 1.0 para dois aminoácidos hidrofóbicos, 0.5 para dois hidrofílicos ou -0.5 caso contrário. A distância Euclidiana $d$ também é considerada.

$$E(s, \theta, \beta) = E_1(\theta) + E_2(s, \theta, \beta) \tag{2}$$

$$E_1(\theta) = \frac{1}{4} \sum_{i=1}^{L-2} [1 - cos(\theta_i)] \tag{3}$$

$$E_2(s, \theta, \beta) = 4 \sum_{i=1}^{L-2} \sum_{j=i+2}^{L} [d(p_i, p_j)^{-12} - c(s_i, s_j) \times d(p_i, p_j)^{-6}] \tag{4}$$

Proposto por [Bošković and Brest 2019], um componente a mais é adicionado a função de energia da Equação 2 com o objetivo de compactar a proteína em núcleos hidrofóbicos. Este termo calcula a distância entre os aminoácidos hidrofóbicos e o seu respectivo centroide $c$ (Equação 5).

$$E_3 = \sum_{i=1}^{L} d(p_i, c) \tag{5}$$

### 2.2. Algoritmos DSMDE e jDE

O algoritmo DSMDE (*Dynamic Speciation-based Mutation Strategies*) insere o conceito de nichos no algoritmo DE, elabora duas novas mutações que passam a considerar o nicho do indivíduo e elimina a necessidade de ajustar os parâmetros de controle[Deng et al. 2019]. O parâmetro $NP$ que define o tamanho da população passa a ser ajustado em relação a dimensão $D$ do problema, onde $NP = D \times 5$, aumentando a capacidade de exploração em instâncias mais complexas. Neste trabalho, $D$ representa o número de ângulos da conformação de uma proteína. Os demais parâmetros $F$ e $CR$ são atualizados em cada iteração a partir de um número aleatório gerado com a distribuição Levy, com média $0.6$ e $0.8$, respectivamente, e desvio $0.1$.

A etapa de inicialização ocorre criando $NP$ soluções aleatórias, também chamadas de indivíduos, dentro de uma distribuição Uniforme. No ciclo evolutivo é construída uma população *trial* de tamanho $2 \times NP$, sendo que cada indivíduo *target* dá origem a dois novos indivíduos *trial*. A seleção acontece substituindo o melhor indivíduo *trial* caso este possua valor de função objetivo melhor que o valor de seu *target*. A evolução continua até que o critério de parada definido seja alcançando.

No DSMDE, a primeira etapa do ciclo evolutivo é criação de nichos. Para isso, dois parâmetros $C_{min}$ e $C_{max}$ precisam ser determinados para definir o tamanho mínimo e máximo de cada nicho. $C_{min}$ é adotado como 3 e $C_{max}$ como $\frac{NP}{10}$. Para criar os nichos é necessário calcular a distância euclidiana dos indivíduos, de todos para todos. A rotina de criação de nichos inicia escolhendo melhor indivíduo da população para ser o $seed$ do primeiro nicho. Então, o tamanho do nicho é calculado pela Equação 6, onde $S$ determina

o tamanho do nicho $n$, $f(seed)$ representa o valor da função objetivo do $seed$, $f(min)$ e $f(max)$ os valores de energia do melhor e pior indivíduo deste problema de minimização.

$$S(n) = \lceil \frac{f(seed) - f(max)}{f(min) - f(max)} \times (C_{max} - C_{min}) + C_{min} \rceil \quad (6)$$

Uma vez calculado o tamanho do nicho, os $(S - 1)$ indivíduos mais próximos do $seed$ são escolhidos para fazer parte do primeiro nicho. Então, o processo se repete com o restante dos indivíduos até que toda a população faça parte de um nicho.

Diferentemente do DSMDE, as rotinas de autoajuste de parâmetros do algoritmo jDE associa para cada indivíduo $i$ da população um par de parâmetros $F^i$ e $CR^i$ próprios. A motivação é de que os melhores pares darão origem aos melhores indivíduos e portanto sobreviverão nas próximas gerações. Os parâmetros são ajustados de acordo com a Equação 7 onde: $rand_1$, $rand_2$, $rand_3$ e $rand_4$ são quatro número aleatórios no intervalo $[0, 1]$; $F_{lower}$ e $F_{upper}$ representam os limites que o parâmetro F pode ser ajustado, adotados como $0.1$ e $1.0$, respectivamente; $\tau_1$, $\tau_2$ controlam a frequência de ajuste e são definidos inicialmente como $0.1$.

$$\begin{aligned} F^i &= \begin{cases} F_{lower} + (F_{upper} - F_{lower}) * rand_1, & \text{if } rand_2 < \tau_1 \\ F^i, & \text{otherwise} \end{cases} \\ CR^i &= \begin{cases} rand_3, & \text{if } rand_4 < \tau_2 \\ CR^i, & \text{otherwise} \end{cases} \end{aligned} \quad (7)$$

### 2.3. Unidades de Processamento Gráfico de Propósito Geral

As Unidades de Processamento Gráfico de Propósito Geral (GPGPUs ou simplesmente GPUs) são dispositivos, ou *devices*, que disponibilizam arquiteturas massivamente paralelas para realização de computação de alto-desempenho [Soyata 2018]. A plataforma CUDA C++ estende a linguagem de programação C++, com adição de elementos sintáticos para manipulação apropriada do hardware gráfico. Uma função executada em GPU é denominada *kernel* e este precisa ter sua configuração explicitamente determinada. Uma *device* é composto por múltiplos SMs (*Streaming Multiprocessor*) que são construídos por vários SPs (*Streaming Processor*). Por isso, existe uma divisão lógica de *grids*, blocos e *threads*. Um *kernel* invoca uma *grid* que é composta por um ou mais blocos, que são compostos por uma ou mais *threads*. Cada bloco executa independentemente em um SM e todas as *threads* de um bloco executam uma mesma instrução. Durante cada ciclo de *clock* de um SM um conjunto de 32 *threads*, chamado *warp*, executam uma mesma instrução. Por esse motivo, é ideal que cada bloco possua um número de *threads* múltiplo de 32 e que existam poucas divergências no fluxo de execução, para otimizar a execução da aplicação [Soyata 2018]. O contexto em CPU de um código CUDA é denominado *host*.

Existem diferentes tipos de memória na GPU, com diferentes tamanhos e tempos de acesso. A maior e mais lenta é a memória global. Dentro dela também são armazenadas outras duas: memória constante e memória local. A memória constante só pode ser escrita

no *host* mas cada SM possui uma *cache* para leitura otimizada. A memória local é de escopo da *thread* e é alocada quando uma *thread* excede seu limite de registrador. Cada SM possui uma memória compartilhada de escopo de um bloco com latência e tamanho reduzidos. Por fim, existe a memória de registrador com escopo de uma *thread*. O uso otimizado destas diferentes memória pode causar um impacto significativo no tempo de execução total de uma aplicação.

## 3. Abordagem Proposta

O otimizador proposto neste trabalho, chamado de cuDSMjDE, representa a implementação em GPU do algoritmo DSMDE com os parâmetros de controle autoajustados pela lógica do algoritmo jDE. Alguns pontos de destaque do algoritmo proposto são elencados a seguir e explicados em detalhes nesta seção. São eles: a rotina de seleção possui uma estratégia de *crowding*; os melhores novos indivíduos de cada geração são escolhidos para participar de uma técnica de busca local; o algoritmo conta com uma reinicialização parcial ou global da população em caso de estagnação da otimização; e uma função auxiliar para compactação dos núcleos hidrofóbicos também é empregada.

Durante a execução do algoritmo, dez *kernels* são invocados. Nos próximos parágrafos, a notação <*A, B*> representa um *kernel* com a configuração de $A$ blocos e $B$ *threads*. O diagrama que ilustra o fluxo de execução do algoritmo é exibido na Figura 1. As etapas sinalizadas com $SM$ e $C$ representam que o *kernel* está utilizando a memória compartilhada e constante, respectivamente. $R$ representa a geração de números aleatórios.

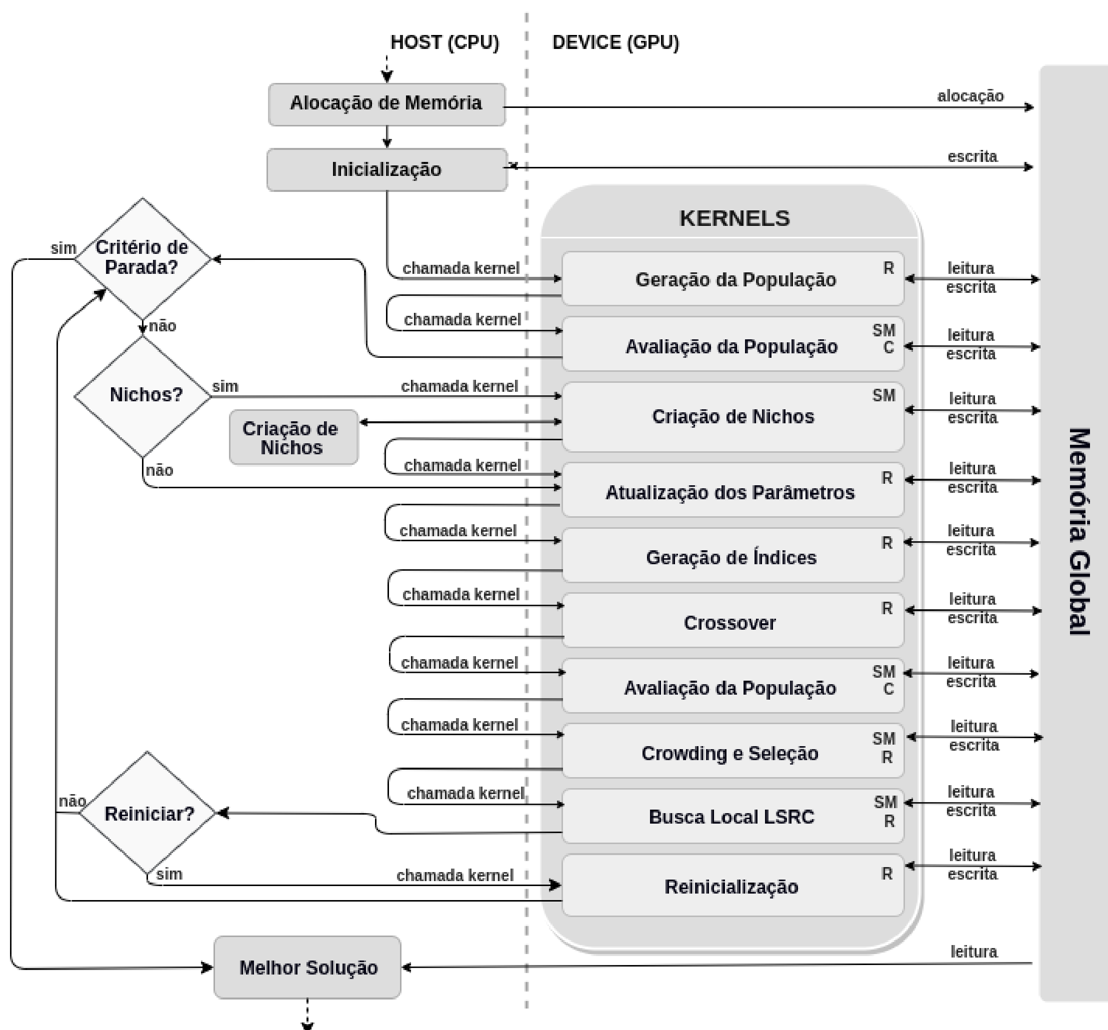
Na etapa de **Alocação de Memória** o *host* reserva toda a memória global que vai ser utilizada durante a execução do algoritmo. Na etapa **Inicialização**, as estruturas de dados auxiliares são configuradas.

A etapa **Geração da População** gera as soluções candidatas iniciais na forma <*NP, D*>, onde $NP$ representa o tamanho da população (número de indivíduos) e $D$ representa o número de dimensões do problema que corresponde ao total de ângulos de uma conformação. Cada *thread* gera um valor aleatório para cada dimensão de cada indivíduo, dentro do domínio $(-180^o, 180^o]$.

O algoritmo possui duas funções objetivos: a função do modelo 3D AB e a função auxiliar para a compactação em núcleos hidrofóbicos, ambas descritas na seção 2.1. Com isso, existem dois níveis de otimização: o primeiro nível que utiliza a função auxiliar e o segundo com a função do modelo 3D AB. A execução começa no primeiro nível e a troca de níveis acontece em sinergia com a rotina de reinicialização. A rotina **Avaliação da População** <*NP, L*>, onde $L$ representa o comprimento da proteína sendo otimizada, ocorre com cada bloco avaliando um indivíduo e cada *thread* representando uma parcela do somatório de cada termo de uma função objetivo, de acordo com o nível de otimização atual. Após os cálculos, uma técnica de redução da operação soma acontece no escopo de um bloco, para acumular o valor final de energia.

A rotina **Criação de Nichos** realiza o cálculo da distância dos indivíduos de todos para todos <*NP, NP*>, com os índices de blocos e *threads* representando os indivíduos. A cada cálculo de distância, todas as *threads* realizam o cálculo para um mesmo indivíduo. Sendo assim, a cada cálculo, todas as dimensões deste indivíduo são carregadas na memória compartilhada para leitura otimizada. Após a construção da matriz de distâncias,

**Figura 1. Diagrama de Fluxo de Execução**




esta é ordenada em GPU com auxilio da biblioteca *THURST*, já que os indivíduos mais próximos são considerados para a criação dos nichos. Além disso, o vetor que armazena os valores da função objetivo também são ordenados em GPU. Então, a criação de nichos do algoritmo DSMDE ocorre em CPU e as informações são copiadas para a memória global da GPU após a conclusão. No algoritmo aqui proposto, os nichos são criados a cada $\frac{Pb \times D}{1.25}$ gerações, com objetivo de permitir a convergência interna dos indivíduos de cada nicho. O parâmetro $Pb$ será mencionado posteriormente.

Os parâmetros do algoritmo são autoajustados na rotina **Atualização dos Parâmetros** $< \lceil \frac{NP}{32} \rceil, 32 >$ com cada *thread* atualizando os parâmetros de um indivíduo. O número de *threads* de um bloco é definido como 32 devido ao tamanho do *warp* da GPU. A **Geração de Índices** para mutação $< \lceil \frac{NP}{32} \rceil, 32 >$ também acontece com cada *thread* representando um indivíduo. As Equações 8 e 9 são utilizadas para mutação e todos os índices pertencem a indivíduos de um mesmo nicho. Ambas são inspiradas em mutações do algoritmo DE padrão.

$$V_1^{i,n} = X^{seed_n} + F^i \times (V^{r1_n} - V^{r2_n}) \tag{8}$$

$$V_2^{i,n} = X^{r3_n} + F^i \times (V^{r4_n} - V^{r5_n}) \quad (9)$$

A rotina **Crossover** *<NP, D>* acontece com cada bloco gerando dois novos indivíduos *trial* compostos por dimensões do indivíduo pai ou por cálculos de cada equação de mutação, dependendo da geração de um número aleatório. Caso o número aleatório seja maior que a probabilidade de crossover $CR$, a informação vem do indivíduo pai. Caso contrário, vem da equação de mutação.

Durante o ciclo evolutivo, a rotina **Avaliação da População** $< 2 \times NP, L >$ acontece novamente mas agora com o dobro de blocos devido ao fato de cada indivíduo ter gerado dois novos filhos a partir das mutações.

A etapa de **Crowding e Seleção** *<NP, K>*, onde $K = \lceil \frac{S_{max}}{32} \times 32 \rceil$, ou seja, o múltiplo de 32 mais próximo do tamanho do maior nicho da população $S_{max}$, pode adotar dois fluxos diferentes em cada bloco. Para cada bloco, se um número aleatório satisfazer a probabilidade $CWD_{prob}$, o indivíduo de índice do bloco participa do processo de *crowding*. Caso contrário, o melhor indivíduo *trial* substitui seu pai caso apresente função objetivo superior. O *crowding* acontece com cada *thread* representando um individuo *target* do nicho e calculando a distância para o indivíduo trial do bloco. Como todas as *threads* calculam a distância para um mesmo indivíduo, este é carregado na memória compartilhada. O indivíduo mais próximo é substituído pelo melhor dos dois *trial* caso haja melhoria no valor de energia.

Na rotina **Busca Local LSRC** *<M, L – 5>*, a variável $M$ representa o número de novos indivíduos que são superiores a seus pais. O número de *threads* é referente ao número de aminoácidos que serão deslocados e reavaliados. Os cálculos geométricos podem ser encontrados em [Bošković and Brest 2018]. Em cada bloco, a melhor das novas conformações compete com a conformação original para a próxima geração.

Antes de avançar para a próxima geração do algoritmo, uma checagem é efetuada para verificar a necessidade da rotina de **Reinicialização** da população. Esta rotina é composta por dois tipos de reinicialização: local e global, e possui três parâmetros para controlar a frequência de reinicialização: $Pb$, $Hb$ e $Lb$. Além disso, as informações de dois indivíduos são armazenadas em memória: $X_{best-local}$ e $X_{best-global}$. O $best - global$ representa o melhor indivíduo de todo o processo evolutivo. O $best - local$ corresponde ao melhor indivíduo entre o início da execução do algoritmo e uma reinicialização global, ou entre duas reinicializações globais. Uma reinicialização local acontece caso a melhor solução da população não apresente melhora durante $Pb \times D$ gerações dentro do segundo nível ou $Hc \times D$ no primeiro. Essa rotina consiste em copiar a informação do $X_{best-local}$ para todos indivíduos da população e atribuir um novo valor aleatório dentro dos limites do problema para $C$ componentes de todos os indivíduos. Na etapa de cópia com estrutura *<D, NP>*, cada bloco carrega a respectiva dimensão do $best - local$ na memória compartilhada para então cada *thread* realizar a substituição. Então, os componentes são reiniciados *<C, NP>* onde cada *thread* reinicia um componente de um indivíduo. Sempre que uma reinicialização local acontece, o nível de otimização é alterado. Já a rotina de reinicialização global é invocada caso haja $Lb \times D$ reinicializações locais mal-sucedidas, onde o indivíduo $X_{best-local}$ não é atualizado. Então, a rotina de inicialização da população é invocada, gerando uma população aleatória.

Ao final do algoritmo, a melhor solução da população é retornada como resultado da otimização.

## 4. Experimentos, Resultados e Análises

Os experimentos foram realizados no Laboratório de Inteligência Computacional (LABICOM) da UDESC em um servidor com sistema operacional Ubuntu 18.04, processador Intel i9 de 3.6 Ghz e placa de vídeo NVIDIA GeForce TITAN V, que conta com 5120 CUDA *Cores* (80 *Streaming Multiprocessors* e 64 *CUDA Cores* por SM), 12GB de memória global e *clock* de 1455 MHz.

Os parâmetros da rotina de reinicialização $Pb$, $Hc$, $Lb$ e $C$ foram definidos como 15, 15, 1 e $0.05 \times D$, respectivamente. A probabilidade de *crowding* $CWD_{prob}$ foi escolhida como 25%. Estes parâmetros foram definidos de maneira empírica. Os valores dos parâmetros do jDE são os sugeridos pelo autor de referência da estratégia, com $F_{lower}$, $F_{upper}$, $\tau_1$ e $\tau_2$ igual a 0.1, 1.0, 0.1 e 0.1, respectivamente. O parâmetro $NP$ é determinado conforme propõe o DSMDE, com $NP = D \times 5$.

Quatro proteínas do repositório PDB (*Protein Data Bank*[1]) foram utilizadas. São elas: 1CRN com 46 aminoácidos; 1HVV com 75 aminoácidos; 1PCH com 88 aminoácidos; e 2EWH com 98 aminoácidos. Foram empregadas 50 bilhões de avaliação de função objetivo para o algoritmo cuDSMjDE. O algoritmo cuDSMjDE foi executado 10 vezes para cada proteína e os valores de média de energia ($x$), desvio padrão ($s$) e melhor valor de energia $f^*$ são apresentados na Tabela 1. A comparação dos resultados obtidos foi realizada com os trabalhos apresentados por [Bošković and Brest 2016, Bošković and Brest 2018, Bošković and Brest 2019] e que foram brevemente descritos na introdução deste artigo. Fato a ser destacado é que os algoritmos utilizados para comparação são considerados o estado-da-arte para esta abstração do problema de predição de estrutura de proteínas e que estes utilizaram um tempo limite de 4 dias para cada execução de cada proteína, não contabilizando a quantidade de avaliações de função. No caso do trabalho em tela, considerando 50 bilhões de avaliação, o tempo médio de execução para cada proteína para o cuDSMjDE foi de 2h e 46min.

Com base nos resultados apresentados na Tabela 1, é possível observar que o cuDSMjDE foi superior tanto na média quanto na melhor conformação quando comparado aos algoritmos $\text{DE}_{\text{pfo}}$ e $\text{DE}_{\text{lsrc}}$. Em relação ao $\text{DE}_{\text{2L}}$, a abordagem proposta obteve resultados próximos para as três primeiras proteínas e resultados melhores para média e melhor conformação para a proteína 2EWH.

Uma experimentação para verificar o desempenho do algoritmo proposto quando comparado com sua versão sequencial executada completamente em CPU também foi realizada. Este experimento analisou as mesmas proteínas com novas execuções em GPU. Dessa vez, um total de 10 milhões de avaliações para executar nas versões em CPU e em GPU. Os resultados do tempo médio em segundos e respectivos *speedups* são apresentados na Tabela 2.

Em questão de performance, a implementação em GPU teve um ganho bastante expressivo, alcançando cerca de 708x de *speedup* para a maior proteína (2EWH). É possível observar que o tempo em GPU escala de maneira significativamente mais lenta que

[1]PDB website: https://www.rcsb.org/

**Tabela 1. Comparação de Energia Livre**

| Proteína | L | | $DE_{pfo}$ | $DE_{lsrc}$ | $DE_{2L}$ | cuDSMjDE |
|---|---|---|---|---|---|---|
| **1CRN** | **46** | x | -86.0390 | -89.8223 | -93.7138 | -92.6620 |
| | | $s$ | 1.45 | 0.65 | 0.55 | 1.09 |
| | | $f^*$ | -89.2001 | -92.9853 | -95.3159 | -94.5862 |
| **1HVV** | **75** | x | -68.8332 | -91.4531 | -98.0730 | -96.6665 |
| | | $s$ | 4.08 | 1.92 | 1.30 | 1.14 |
| | | $f^*$ | -82.1427 | -95.4475 | -101.6018 | -98.4507 |
| **1PCH** | **88** | x | -117.7610 | -153.1003 | -161.4182 | -161.2535 |
| | | $s$ | 6.26 | 2.71 | 2.13 | 1.90 |
| | | $f^*$ | -131.7790 | -156.5250 | -166.7194 | -164.0593 |
| **2EWH** | **98** | x | -203.6810 | -240.2247 | -250.2833 | -253.6498 |
| | | $s$ | 7.18 | 2.14 | 3.18 | 2.90 |
| | | $f^*$ | -225.0968 | -245.5190 | -257.0741 | -257.3538 |

**Tabela 2. Comparação de Tempo Médio de Execução em segundos**

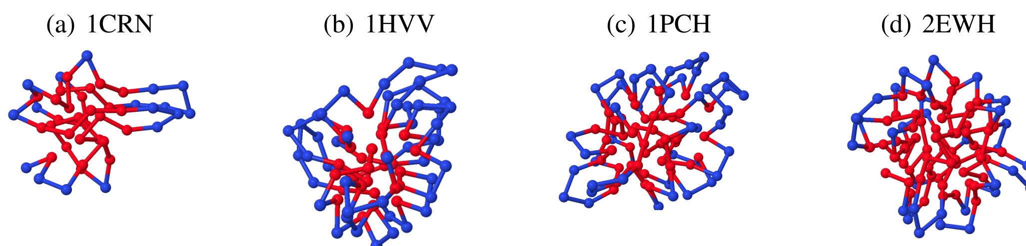
| Proteína | L | Tempo CPU | Tempo GPU | Speedup |
|---|---|---|---|---|
| **1CRN** | **46** | 382.60 | 2.12 | 180.36 |
| **1HVV** | **75** | 881.15 | 2.10 | 419.43 |
| **1PCH** | **88** | 1253.71 | 2.14 | 585.57 |
| **2EWH** | **98** | 1550.70 | 2.18 | 708.78 |

em CPU. Isto se deve ao custo computacional da rotina de avaliação, que tem complexidade $O(L^2)$ devido ao seu segundo termo. Nesta rotina $NP$ indivíduos são avaliados e este número depende da dimensão do problema, onde $NP = D \times 5$. A dimensão cresce de acordo o comprimento da proteína, com $D = L \times 2 - 5$. Portanto, a relação entre $NP$ e $L$ é de $NP = 10 \times L - 25$. Dessa forma, o custo total de rotina é de $O([10 \times L - 25] * L^2) = O(L^3)$, ou seja, escala ao cubo do tamanho da cadeia proteica. Quando considerando a aplicação em GPU, o *kernel* de avaliação tem seu número de blocos igual a $NP$, crescendo de maneira linear. Já o número de *threads* também cresce linearmente de acordo com $L$, sendo a tarefa mais custosa de cada *thread* o somatório interno do segundo termo da equação de energia, que é composto por $L$ parcelas. Em outras palavras, a placa gráfica utilizada, que possui 5120 *CUDA Cores*, consegue tratar as instâncias deste trabalho em escala linear. Para instâncias maiores do problema, o crescimento do tempo em GPU tende a adotar um comportamento semelhante ao em CPU, no momento em que a carga computacional necessária para cada núcleo seja mais elevada. Por fim, observa-se que a execução em CPU das 50 bilhões de avaliação de função objetivo promovida pelo algoritmo cuDSMjDE na obtenção dos resultados comparados com a literatura se torna inviável, justificando o uso de processamento massivamente paralelo.

A ilustração tridimensional da melhor conformação obtida para cada proteína é exibida na Figura 2, onde as esferas vermelhas representam os aminoácidos hidrofóbicos e azuis os hidrofílicos. Como é possível observar, as conformações obtidas conseguiram replicar o fenômeno físico onde os aminoácidos hidrofóbicos se concentram no centro da

conformação.

**Figura 2. Ilustração das melhores conformações das proteínas otimizadas**




## 5. Considerações Finais e Trabalhos Futuros

Este trabalho abordou o problema de predição de estrutura de proteínas com o modelo *ab initio* 3D AB Off-Lattice. A abordagem proposta, chamada de cuDSMjDE, implementa em GPU uma adaptação do algoritmo DSMDE com os parâmetros autoajustados pela lógica do algoritmo jDE. Uma estratégia de *crowding* foi adicionada a etapa de seleção, com o objetivo de manter a diversidade da população. Uma rotina que reinicia parcialmente ou completamente a população é executada em momentos de estagnação do algoritmo. Duas estratégias específicas para o modelo de representação proteico abordado neste trabalho foram implementadas. A primeira introduz um componente na função objetivo com a finalidade de compactar as conformações das proteínas em núcleos de aminoácidos hidrofóbicos. A segunda consiste em deslocar dois aminoácidos preservando o restante da conformação. Com excessão de parte da rotina de criação de nichos, todas as etapas do otimizador proposto são executadas em GPU.

Os resultados alcançados com este trabalho foram bastante competitivos com a literatura. Quatro proteínas reais foram otimizadas, sendo que três delas alcançaram valores de energia livre próximos aos da literatura tanto de média quanto de melhor resultado. A otimização da proteína com a maior cadeia de aminoácidos conseguiu obter a maior média e melhor conformação dentre todos os trabalhos comparados. Um ponto de destaque da comparação realizada foi o tempo de execução. Os trabalhos estado-da-arte utilizaram um tempo limite de 4 dias de otimização para cada execução de cada proteína. O presente trabalho minimizou cada proteína em menos de 3 horas. O desempenho do algoritmo implementando, quando comparado com sua versão completamente sequencial executada em CPU, alcançou um *speedup* de 708x para a maior proteína. Isso evidencia o potencial da GPGPU para o problema de predição de estrutura de proteínas com o modelo *ab initio* 3D AB Off-Lattice.

Como trabalhos futuros, propõe-se o uso de outras técnicas para autoajuste de parâmetros e análise de sensibilidade dos mesmos [Parpinelli et al. 2019]. Uma análise do tempo de execução de cada rotina (*profiling*) pode ser realizada para possíveis otimizações e análise do algoritmo proposto. Além disso, pretende-se desenvolver sistemas que integram diferentes níveis de complexidade do problema acelerados com o uso de GPUs, visto que este problema se mostra escalável com o uso de arquiteturas massivamente paralelas. Uma

## Agradecimentos

Os autores agradecem a Universidade do Estado de Santa Catarina e a agência FAPESC pelo auxílio financeiro, e agradecem também a empresa NVIDIA pela doação da GPU GeForce Titan V.

## Referências

Alberts B, Johnson A, L. J. e. a. (2002). *Molecular Biology of the Cell*. Garland Science, 4th edition edition.

Boiani, M. and Parpinelli, R. S. (2020). A GPU-based hybrid jDE algorithm applied to the 3d-AB protein structure prediction. *Swarm and Evolutionary Computation*, 58:100711.

Bošković, B. and Brest, J. (2016). Differential evolution for protein folding optimization based on a three-dimensional AB off-lattice model. *Journal of Molecular Modeling*, 22(10).

Bošković, B. and Brest, J. (2018). Protein folding optimization using differential evolution extended with local search and component reinitialization. *Information Sciences*, 454-455:178–199.

Bošković, B. and Brest, J. (2019). Two-level protein folding optimization on a three-dimensional ab off-lattice model.

Deng, L., Zhang, L., Sun, H., and Qiao, L. (2019). DSM-DE: a differential evolution with dynamic speciation-based mutation for single-objective optimization. *Memetic Computing*, 12(1):73–86.

Fraenkel, A. (1993). Complexity of protein folding. *Bulletin of Mathematical Biology*, 55(6):1199–1210.

Jana, N. D., D. S. and Sil, J. (2018). *A Metaheuristic Approach to Protein Structure Prediction*. Springer International Publishing.

Li, B., Chiong, R., and Lin, M. (2015). A balance-evolution artificial bee colony algorithm for protein structure optimization based on a three-dimensional AB off-lattice model. *Computational Biology and Chemistry*, 54:1–12.

Li, T., Zhou, C., Wang, B., Xiao, B., and Zheng, X. (2018). A hybrid algorithm based on artificial bee colony and pigeon inspired optimization for 3d protein structure prediction. *Journal of Bionanoscience*, 12(1):100–108.

Opara, K. R. and Arabas, J. (2019). Differential evolution: A survey of theoretical analyses. *Swarm and Evolutionary Computation*, 44:546 – 558.

Parpinelli, R., Felippe Plichoski, G., Silva, R., and Narloch, P. (2019). A review of techniques for online control of parameters in swarm intelligence and evolutionary computation algorithms. *International Journal of Bio-Inspired Computation*, 13:1–20.

Siarry, P., editor (2016). *Metaheuristics*. Springer International Publishing.

Soyata, T. (2018). *GPU Parallel Program Development Using CUDA*.

Stillinger, F. H., Head-Gordon, T., and Hirshfeld, C. L. (1993). Toy model for protein folding. *Physical Review E*, 48(2):1469–1477.