O impacto da computação paralela em Algoritmos Genéticos

1st Alex V. Echeverria U. Silva
Bacharelado em Inteligência Artificial
Universidade Federal de Goiás
Goiânia, Brasil
ugarte_alex@discente.ufg.br

2rd Heloisy Pereira Rodrigues
Bacharelado em Inteligência Artificial
Universidade Federal de Goiás
Goiânia, Brasil
heloisyrodrigues@discente.ufg.br

3rd Jonas Gomes da Silva Junior
Bacharelado em Inteligência Artificial
Universidade Federal de Goiás
Goiânia, Brasil
jonasjunior@discente.ufg.br

***Resumo*—Este trabalho tem por objetivo demonstrar, a partir de testes realizados, qual o impacto no tempo de processamento de um problema de otimização resolvido a partir de algoritmos genéticos ao explorar os conceitos de paralelismo. Tanto os processos de paralelismo em CPU e GPU são demonstrados em uma análise profunda dos resultados de cada teste, isso tudo feito na linguagem de alto nível Python, com frameworks otimizados em C como Numpy e Numba.**

***Index Terms*—Genetic Algorithms, Evolutionary computing, Parallel computing, High performance computing**

I. Introdução

A. Algoritmos genéticos

Os algoritmos genéticos [1] (AG) são uma técnica de otimização de problemas inspiradas no processo evolutivo natural dos seres vivos, ou seja, leva em consideração os conceitos apresentados na obra de Charles Darwin, "A origem das espécies", mas também em conceitos da genética moderna. Foi uma técnica desenvolvida na década de 70 por John Holland [2], mas que foi popularizada por David Goldberg [3] no fim da década de 80. Desde então, computação e genética foram fundidas e amplamente utilizadas na solução de diversos problemas de otimização.

Seu funcionamento [4] consiste basicamente em 6 passos:

- Inicialização da população;
- Verificação do fitness (aptidão) de cada indivíduo;
- Seleção dos indivíduos mais aptos;
- Cruzamento dos indivíduos;
- Mutação dos indivíduos;
- Verificação do critério de parada.

Inicialização da população

A inicialização da população envolve cuidados além apenas de incializar um vetor gênico, pois é fundamental que o projetista saiba bem como representar o problema a ser resolvido em que o vetor genético (ou cromossomo) faça sentido, já que as operações são realizadas de forma que os cromossomos evoluam e consigam atingir o melhor estado possível. Por isso, é imprescindível que cada cromossomo gerado represente bem as variáveis do problema para que as evoluções realmente busquem a melhor solução. Um exemplo é que cada gene represente os pesos de uma rede neural artificial, sendo que a cada evolução é esperado que a *Loss Function* seja minimizada.

Verificação do fitness

Nessa etapa, para cada indivíduo da população é calculado sua aptidão por meio de uma *fitness funtion*, isto é, uma função que mensura a aptidão de acordo com algum critério matemático ou que o projetista julgue necessário. Seguindo o exemplo de redes neurais citado no tópico anterior, a fitness function pode ser dada pelo próprio valor da Loss Function que calcula o quão bem a rede neural está aprendendo e generalizando. Também é possível modelar a fitness function de forma que tenha multiobjetivos, ou seja, que leve em consideração o custo de vários objetivos reunidamente. Um exemplo para essa questão é um problema em que a quantidade de *features* e a acurácia de um classificador deve ser levado em consideração, ou seja, deseja-se obter um modelo que não só tenha a maior acurácia como também use a menor quantidade de features possíveis.

Seleção de indivíduos

A seleção dos indivíduos podem ser feitas de 3 maneiras diferentes. Uma forma mais simples, mas também mais ingênua é selecionar aleatoriamente a quantidade desejada de indivíduos em meio à população apresentada. Essa forma, como já dito, é ingênua e não leva em consideração a aptidão individual dos indivíduos, o que implica em sua baixa utilização. Outra forma mais utilizada de seleção é o torneio, em que obtém-se apenas os melhores indivíduos da geração que está sendo analisada. Essa é uma forma de se obter os mais aptos sempre, mas que pode excluir segmentos gênicos que seriam interessantes de se manterem no pool genético, mesmo o cromossomo não estando entre os melhores. A forma mais utilizada de selecionar indivíduos é a roleta, em que é realizado um sorteio aleatório dos indivíduos, no entanto cada um tem probabilidade de ser selecionado proporcionalmente ao seu fitness. Isso implica que aqueles mais aptos têm mais chance de serem escolhidos, no entanto há a possibilidade de ser excluído. Aqueles que tem baixa aptidão também têm chances de serem escolhidos. Assim, não há o problema de

sequências gênicas interessantes serem perdidas.

Cruzamento dos indivíduos

Nessa etapa, ocorre a troca de informação genética entre dois indivíduos, de forma que os descendentes gerados tenham características herdadas dos pais. Como hiper-parâmetro, é necessário que o projetista defina uma probabilidade de ocorrer o cruzamento P_cross, a qual indica que, mesmo um indivíduo muito apto, pode não deixar descendentes. Os indivíduos selecionados para reproduzir também seguem a ideia de roleta, pois são sorteados dois indivíduos aleatoriamente com probabilidade de sorteio proporcional ao fitness de cada um. Com isso, as reproduções ocorrem até que haja uma proporção de 7 filhos por pai. A população que será retornada depende do critério que o projetista escolha, se será *Mi + Lambda* ou *Mi, Lambda*, pois assim os pais serão considerados na população seguinte ou se somente os filhos serão considerados na geração seguinte. Considerar os pais é uma estratégia elitista, ou seja, uma boa solução não será perdida ao longo da evolução.

Mutação dos indivíduos

A mutação é a principal forma de gerar diversidade na população de indivíduos, pois garante que novas características sejam inseridas ao longo da evolução, de forma que não ocorra platôs e todos os indivíduos tornem-se iguais. Como o cruzamento, o projetista também deve estipular de antemão uma probabilidade de mutação P_mut, que indica qual a chance de ocorrer mutações. Geralmente é um valor entre 0.01 e 0.2 [5] Ela pode ocorrer de duas formas, a "bit-flip"(para cromossomos binários) ou a Gaussiana (para valores reais). Como o próprio nome sugere, a bit-flip é apenas trocar um gene binário para o seu oposto. Já a gaussiana explora uma distribuição normal que gera valores baseado em um desvio-padrão determinado de acordo com a taxa de sucesso de mutações, isto é, de acordo com quantas mutações levaram a valores de fitness maiores se comparados com os fitness antes da mutação.

Critério de parada

Após todos esses processos, é verificado se o melhor fitness da população atingiu um valor mínimo desejado pelo projetista, caso tenha atingido, retorna-se o melhor indivíduo, caso contrário uma nova iteração é realizada com todas as sequências acima explicadas, excetuando a geração de uma nova população.

B. Computação Paralela e seus benefícios

Em um breve resumo, computação paralela é uma área da ciência da computação composta por diversas outras subáreas, entre elas a arquitetura de computadores, as linguagens de programação, a de compiladores e de teoria da computação, qual objetivo é diminuir o tempo de processamento necessário para lidar com problemas de alto custo computacional por meio da ampliação do número de unidades computacionais [6]. Com isso, programas antes limitados a computadores com apenas uma unidade computacional podem usufruir do processamento simultâneo de várias unidades processadoras.
As formas mais comuns de se paralelizar um programa computacional é em relação aos dados em que se processa e às tarefas que se realiza.
Paralelismo de dados é a técnica utilizada quando se divide as entradas do programa entre as unidades computacionais disponíveis, no entanto todas elas executarão a mesma tarefa sobre cada entrada recebida. Já o paralelismo de tarefas é a técnica que permite que cada unidade processadora receba uma tarefa diferente para realizar sobre os mesmos dados, só que cada entrada que elas recebem estão em instantes diferentes do mesmo dado, ou seja, as tarefas são diferentes e a "maturidade"dos dados também, isso segue uma ideia muito próxima de *pipelines*.
Os algoritmos genéticos, por se tratarem de uma meta-heurística populacional, têm o espaço de busca naturalmente grande e complexo. Embora as operações de mutação e cruzamento sejam relativamente simples, a fitness function e a grande quantidade de indivíduos em uma população junto ao possível grande número de genes torna o processo de evolução lento para uma máquina sequencial single-core. Justamente nesse cenário que esse artigo busca explorar as diferenças de desempenho ao aplicar técnicas de paralelização de tarefas e de dados sobre o processo evolutivo dos AG's. Explicamos em detalhes diversos testes e resultados nas próximas seções.

II. METODOLOGIA

A. Modelagem do problema

Para esse trabalho, desejamos usar os algoritmos genéticos para treinar uma rede neural que reconheça dígitos manuscritos, sendo essa a rede de número 2 do artigo LeNet de Yann LeCun [7], uma rede com duas camadas ocultas, com 12 e 10 neurônios respectivamente, totalmente conectados em si e nas entradas. Cada imagem contém dimensões quadradas de 28x28 e representam um dígito de zero a nove. Então, cada cromossomo (vetor de características) representa os pesos da rede neural, os quais, quando somados, resultam em 9550 pesos. Isso significa que cada indivíduo da população de indivíduos do AG carrega essa grande quantidade de informações. Com as iterações de cada geração, é esperado que entre os crossovers e as mutações os melhores pesos sejam descobertos e passados adiante.

B. Fitness funtion

A fitness function utilizada para avaliar a aptidão de cada indivíduo é a *loss* que cada indivíduo gera ao ser aplicado na rede neural e testado em 500 imagens. Aqueles que geram a menor loss são mais aptos, mas como algumas escolhas dependem de um valor que represente a probabilidade de sucesso de cada indivíduo, utilizamos o inverso da loss, ou seja, 1 dividido pela loss, de tal forma que quanto maior seja esse valor resultante, mais chances o indivíduo terá de ser escolhido para o cruzamento.

C. Códigos

Em relação à implementação do algoritmo genético, desenvolvemos nós mesmos o software, parte essa que demandou um grande esforço e tempo, pois até o pleno funcionamento foram muitas linhas de código, *debugging* e otimizações. Consideramos um código voltado à implementação do AG, outro que implementa a rede neural artificial e outro que faz a união dos dois, que é onde se declara a fitness function.

D. Hiperparâmetros

O que de fato utilizamos para realizar as medidas de desempenho entre os diversos experimentos foi o tamanho da população que o algoritmo utilizaria, isto é, quantos indivíduos cada núcleo computacional receberia para ser avaliado em cada iteração. Mantivemos o valor de 1600 indivíduos para as avaliações, alterando esse valor para as abordagens de Amdahl, em que a quantidade de dados mantém-se inalterada para as avaliações de melhoras do desempenho, quanto de Gustafson, em que é esperado que a quantidade de dados a ser processado pela máquina paralela aumente arbitrariamente.

Deixamos valores padrões para os outros hiperparâmetros do algoritmo genético, como taxa de cruzamento (crossover, que indica a chance de dois indivíduos escolhidos produzirem descendentes) em 80%, taxa de mutação (a probabilidade de um indivíduo ter seus genes mutados) em 25% e limitamos em 10 gerações apenas para tornar mais rápida a execução dos algoritmos várias vezes.

E. Paralelismo

Paralelizamos o código levando em consideração Amdahl, em que cada núcleo execute todo o pipeline do algoritmo genético para uma quantidade menor de indivíduos em cada população, isto é, se é desejado que haja 1000 indivíduos em cada população, é distribuído de forma uniforme essa quantidade entre os núcleos disponíveis para processamento. Ou seja, se há 1000 cromossomos e 10 núcleos, cada núcleo computacional ficará responsável por evoluir 100 cromossomos.

Já para o modelo de Gustafson, a cada teste aumentamos a quantidade de indivíduos na mesma proporção em que aumentávamos o número de processadores a ser utilizado. Isto é, se dobramos o número de processadores, também dobramos o número de indivíduos na população.

Para o paralelismo em GPU, paralelizamos a execução da rede neural na GPU, para aproveitar o poder que elas oferecem em multiplicação paralela de matrizes, pois todos os pesos estão organizados na forma matricial, juntamente às imagens da base.

III. Experimentos

A máquina em que conduzimos os experimentos foi uma octa-core, 16 threads, rodando Windows 11 e com processos em segundo plano acontencedo normalmente. Especificamente rodamos em um processador Ryzen 7 5800H e em uma GPU Nvidia RTX 3060. Executamos diversos experimentos, abaixo exemplificamos a saída de um:

1) 1 núcleo e 1600 indivíduos na população:

- Core: 0, Iteração: 0, Fitness atual: 6.05, Tempo de execução: 19.60 seg
- Core: 0, Iteração: 1, Fitness atual: 5.83, Tempo de execução: 19.08 seg
- Core: 0, Iteração: 2, Fitness atual: 5.81, Tempo de execução: 19.11 seg
- Core: 0, Iteração: 3, Fitness atual: 5.79, Tempo de execução: 20.06 seg
- Core: 0, Iteração: 4, Fitness atual: 5.75, Tempo de execução: 20.10 seg
- Core: 0, Iteração: 5, Fitness atual: 5.78, Tempo de execução: 21.39 seg
- Core: 0, Iteração: 6, Fitness atual: 5.68, Tempo de execução: 21.19 seg
- Core: 0, Iteração: 7, Fitness atual: 5.69, Tempo de execução: 22.17 seg
- Core: 0, Iteração: 8, Fitness atual: 5.59, Tempo de execução: 20.93 seg
- Core: 0, Iteração: 9, Fitness atual: 5.59, Tempo de execução: 22.68 seg

Como podemos notar, a cada iteração temos o feedback de diversas informações, como o fitness atual e o tempo de execução. Baseado nessas informações de cada experimento, conseguimos montar a relação de speedup ao somarmos o tempo das iterações e fazer os devidos cálculos.
Abaixo fica o resultado de cada experimento realizado:

2) Tempo Total:

- (Amdahl) 1 núcleo e 1600 indivíduos: 206.35 seg
- (Amdahl) 2 núcleos e 1600 indivíduos: 108.42 seg
- (Amdahl) 4 núcleos e 1600 indivíduos: 87.72 seg
- (Amdahl) 8 núcleos e 1600 indivíduos: 111.94 seg
- (Amdahl) 16 núcleos e 1600 indivíduos: 116.84 seg
- (Gustafson) 2 núcleos e 2 x 1600 indivíduos: 269.55 seg
- (Gustafson) 4 núcleos e 4 x 1600 indivíduos: 492.69 seg
- (Gustafson) 8 núcleos e 8 x 1600 indivíduos: 1304.00 seg
- (Gustafson) 16 núcleos e 16 x 1600 indivíduos: 803.53 seg
- GPU: 220.47 seg

IV. Speedup's

Sabe-se que para avaliar o desempenho de algo, é necessário o uso de métricas pré-estabelecidas, no processamento paralelo isso não é diferente. Diante disso, há duas métricas importantes e abordadas nesse artigo: speedup e eficiência. Os conceitos são básicos, o speedup mede o ganho de desempenho entre o tempo de execução sequencial e o tempo de execução em paralelo de um algoritmo, enquanto a eficiência mede o quão eficiente é o speedup em relação aos recursos computacionais (processadores) utilizados (speedup ideal contra o real). Para o cálculo do speedup, basta dividirmos o tempo gasto pelo algoritmo executado sequencialmente pelo tempo gasto paralelamente. Já para a eficiência, basta achar a razão entre o speedup obtido e o número de núcleos usados para atingir

esse objetivo. Para melhor compreensão do desempenho do algoritmo usado neste trabalho, apresentamos a seguir os resultados das métricas de desempenho:

1) Speedup Absoluto (razão entre sequencial e paralelo):

- Referente às 2 CPU's: 1.90332116646
- Referente às 4 CPU's: 2.35225125542
- Referente às 8 CPU's: 1.84336579268
- Referente às 16 CPU's: 1.76611564753
- Referente à GPU: 0.93596415843

2) Speedup Relativo:

- Referente à comparação entre 1 e 2 CPU's: 1.90332116646
- Referente à comparação entre 2 e 4 CPU's: 1.23586670336
- Referente à comparação entre 4 e 8 CPU's: 0.78366024396
- Referente à comparação entre 8 e 16 CPU's: 0.95809288343
- *Observa-se que o speedup decrescente para 8 e 16 processadores pode ser explicado pelo fato de a máquina de testes possuir processador octa-core, com 16 threads, ou seja, consumimos todos os 8 núcleos disponíveis, mas o Sistema Operacional e todos os outros processos continuaram a executar ao mesmo tempo que nosso progama, o que deve ter causado lentidão e gargalos ocasionados pelo SO.*

3) Speedup segundo a lei de Amdhal:

- Referente às 2 CPU's: 1.90332116646
- Referente às 4 CPU's: 2.35225125542
- Referente às 8 CPU's: 1.84336579268
- Referente às 16 CPU's: 1.76611564753
- *Vale ressaltar que, como não há o quantitativo da parte sequencial e parte paralela do programa, baseou-se apenas no tempo de execução, logo o resultado equivale ao do Speedup Absoluto.*

4) Speedup segundo a lei de Gustafson:

- Referente às 2 CPU's: 1.53112684922
- Referente às 4 CPU's: 1.67533809224
- Referente às 8 CPU's: 1.26599243904
- Referente às 16 CPU's: 4.10902422618
- *Ressalta-se que fixou-se a quantidade de dados que cada processador recebe, ou seja, a quantidade de dados foi multiplicada pelo número de CPU's a cada teste realizado, o que revela um speedup muito interessante.*

5) Eficiência:

- Referente às 2 CPU's: 0.95166058323
- Referente às 4 CPU's: 0.58806281385
- Referente às 8 CPU's: 0.23042072408
- Referente às 16 CPU's: 0.11038222797

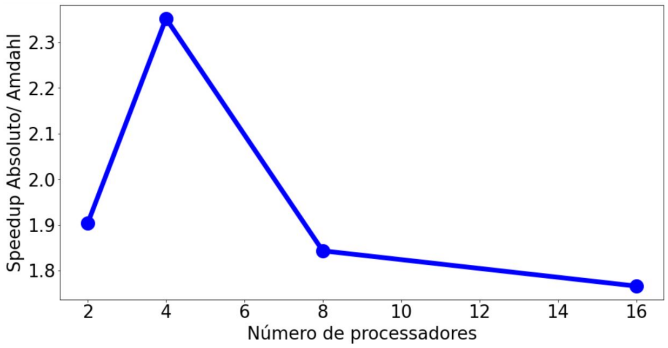

Figura 1. Gráfico dos Speedups Absolutos (coincidente com de Amdahl).

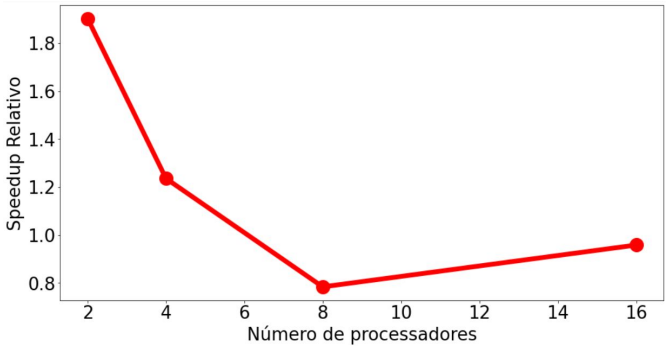

Figura 2. Gráfico dos Speedups Relativos.

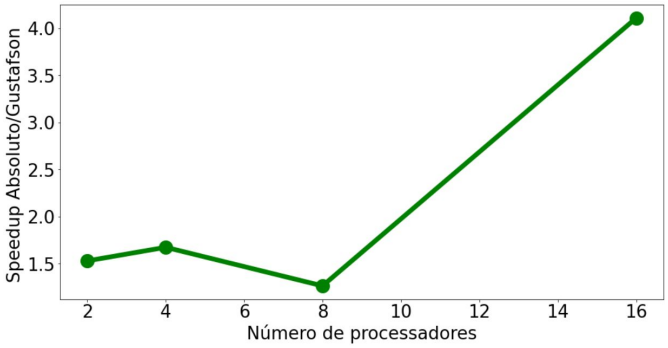

Figura 3. Gráfico dos Speedups segundo a lei de Gustafson.

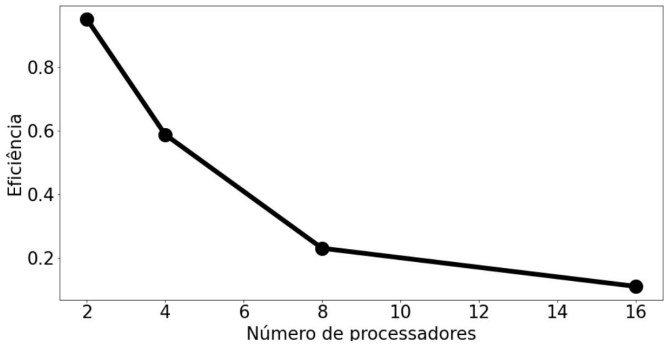

Figura 4. Gráfico da Eficiência.

V. Conclusões

Como podemos perceber, houveram situações em que o speedup foi bem abaixo do esperado, principalmente nos experimentos que seguiam a lei de Amdahl e que foram executados em 8 ou 16 núcleos. Pensamos que esse fato ocorre por conta de estarmos explorando todos os núcleos disponíveis da máquina de testes, o que deve levar a interrupções do sistema para executar seus processos fundamentais, que não temos controle de desligar.

Já em questão da GPU, deve-se ao fato da tarefa de Algoritmos genéticos aproveitar os benefícios do paralelismo em GPU apenas para a fitness function que é basicamente multiplicação de matrizes. Em nosso caso a rede não é tão complexa, por isso o desempenho na CPU e GPU foi similar, sendo da placa de vídeo um pouco mais lento pelo fato de haver trocas de mensagens entre dois dispositivos fisicamente distantes no computador, sofrendo com atrasos e diversos subprocessos que são abertos de fundo para que ocorram essas trocas de dados.

Contudo, ao observarmos o desempenho voltado para Gustafson, percebemos que há um grande ganho computacional ao realizar o paralelismo aumentando os dados, pois cada núcleo recebeu a mesma quantidade de indivíduos que os experimentos com menos núcleos, mas mesmo assim completou a tarefa mais rapidamente.

Portanto, a computação paralela é um forte aliado de algoritmos genéticos pois possibilita acelerar e impulsionar a evolução de uma grande quantidade de indivíduos simultaneamente, abrangendo, assim, uma grande área do espaço de buscas pela melhor solução.

Referências

[1] DARWIN, Charles. A Origem das Espécies, no meio da seleção natural ou a luta pela existência na natureza, 1 vol., tradução do doutor Mesquita Paul.

[2] HOLLAND, John H. Adaptation in natural and artificial systems. University of Michigan Press, 1975.

[3] GOLDBERG, David E. Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Longman Publishing Co., Inc., 1989.

[4] AURÉLIO, M.; PACHECO, C.; PACHECO, C. ALGORITMOS GENÉTICOS: PRINCÍPIOS E APLICAÇÕES. [s.l: s.n.]. Disponível em: <http://www.inf.ufsc.br/ mauro.roisenberg/ine5377/Cursos-ICA/CE-intro_apost.pdf>.

[5] mealpy.evolutionary_based package — MEALPY 2.4.1 documentation. Disponível em: <https://mealpy.readthedocs.io/en/latest/pages/models/mealpy.evolutionary_based.html?module-mealpy.evolutionary_based.GA>. Acesso em: 3 set. 2022.

[6] ATHAIDE COELHO, S.; INTRODUÇÃO A COMPUTAÇÃO PARALELA COM O OPEN MPI. [s.l: s.n.]. Disponível em: <https://www.ufjf.br/getcomp/files/2013/03/Introdu%C3%A7%C3%A3o-a-Computa%C3%A7%C3%A3o-Paralela-com-o-OpenMPI.pdf>. Acesso em: 3 set. 2022.

[7] LECUN, Y. Generalization and Network Design Strategies. jun. 1989. Disponível em: <http://yann.lecun.com/exdb/publis/pdf/lecun-89.pdf>. Acesso em: 11 set. 2022.