# Comparação entre métodos para computar algoritmos genéticos simples em GPU

**Vinícius C. Oliveira de Andrade**[1], **Wagner M. Nunan Zola** [1]

[1]Departamento de Informática – Universidade Federal do Paraná (UFPR)
Curitiba – PR – Brasil

{vcoandrade, wagner}@inf.ufpr.br

***Resumo.*** *Algoritmos genéticos simples podem ser usados para busca de soluções de diversos problemas. Neste trabalho são apresentados quatro métodos diferentes de implementação paralela em GPU para esses algoritmos. Foram obtidos speedups máximos entre 6x e 12,5x em relação à implementação serial.*

## 1. Introdução

Algoritmos genéticos são métodos de busca inspirados na teoria da evolução das espécies de Darwin que podem ser aplicados para encontrar soluções para diversos problemas. Devido a natureza aleatória do processo é necessária a computação de várias gerações até que seja encontrada uma solução satisfatória, o que motiva a busca pela diminuição do tempo de cada uma delas. Neste trabalho são comparadas quatro implementações paralelas e uma serial de modo a evidenciar qual o melhor método em cada situação.

## 2. Fundamentos teóricos

O modelo de seleção dos algoritmos genéticos segue o modelo natural da sobrevivência dos mais bem adaptado para garantir que as gerações futuras de uma determinada espécie tenham membros com valor de adaptação médio maior que as gerações passadas [Osyczka and Kundu 1995]. Nos algoritmos genéticos simples cada indivíduo tende a receber um número de descendentes proporcional ao valor da sua adaptação, sendo adaptação um número que indica o quão boa uma solução é.

## 3. Resultados e discussão

Serão comparados os tempos de execução das gerações em cinco modelos diferentes. O modelo serial será utilizado como referência para os *kernels* paralelos em CUDA, sendo três *s-Persistentes (simples)*: um modelo de *threads* persistentes, as quais calculam os valores necessários em cada geração do algoritmo genético, uma segunda com valores aleatórios pré-calculados, ambas com apenas um bloco, e uma terceira versão utilizando um máximo de blocos ativos possível para o dispositivo, e finalmente o *mt-Kernel (muitas threads)* chamando *threads* em todas as oportunidades presentes. Foram realizados testes para a minimização das funções F1 de De Jong [Jong 1975] e Styblinski–Tang [Styblinski and Tang 1990] variando o número de iterações, utilizando respectivamente 896 e 188 indivíduos devido as limitações do modelo *s-Persistentes*, e o tamanho da população. Para os experimentos foi utilizado um Intel core i5-4460 de 4 núcleos de 3.2GHz, 8GB de RAM e uma GeForce RTX 2070 com arquitetura SMX 7.5 (Turing), 8GB de memória e 2304 núcleos utilizando a versão 10.1 de CUDA.

A Figura 1 apresenta os resultados para a função F1 de De Jong. Os métodos *s-Persistentes* apresentam resultados inferiores quando comparados aos demais pois mesmo fazendo uso de *shared memory* não obtém paralelismo suficiente. Não existem diferenças consideráveis entre as versões persistentes com e sem valores aleatórios pré-calculados.

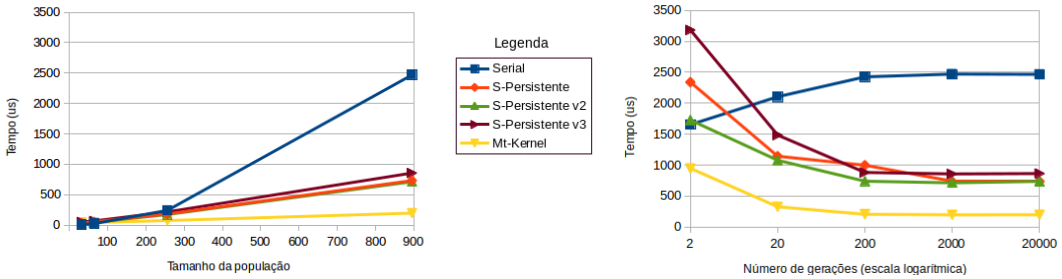


**Figura 1. Tempo por geração da função F1 de De Jong variando: (à esquerda) tamanho da população e (à direita) número de gerações.**

A Figura 2 apresenta os resultados para a função de Styblinski–Tang. Nestes testes é possível ver que os modelos *s-Persistentes* apresentam resultados piores que os demais, mostrando que, para funções um pouco mais complexas a divergência de *threads* já afeta de forma bastante considerável o tempo de computação. A versão 3 em particular apresenta baixo desempenho por não utilizar memória compartilhada.

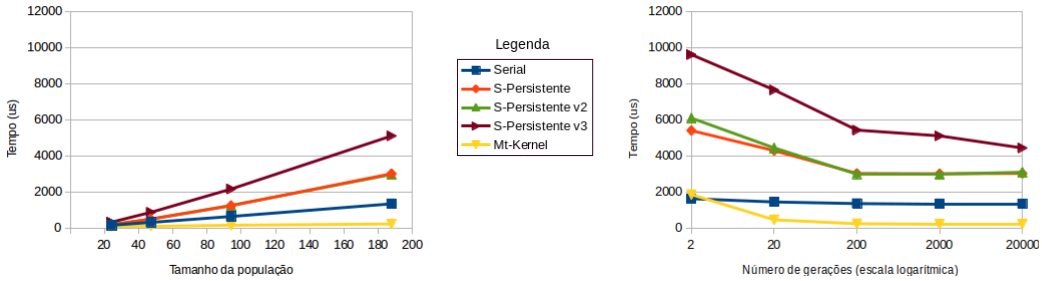


**Figura 2. Tempo por geração da função de Styblinski–Tang Jong variando: (à esquerda) tamanho da população e (à direita) número de gerações.**

## 4. Conclusão

Conforme se pode verificar pelos resultados os modelos de *threads s-Persistentes* não são indicados para esses algoritmos, uma vez que sofrem com as limitações de tamanho da memória compartilhada, e com o acesso à memória global. O modelo *mt-Kernel* apresentou melhor desempenho com *speedups* máximos entre 6x e 12,5x.

## Referências


Jong, K. A. D. (1975). An analysis of the behavior of a class of genetic adaptive systems.

Osyczka, A. and Kundu, S. (1995). A new method to solve generalized multicriteria optimization problems using the simple genetic algorithm. In *Structural optimization*, pages 94–99.

Styblinski, M. A. and Tang, T.-S. (1990). Experiments in nonconvex optimization: Stochastic approximation with function smoothing and simulated annealing. In *Neural Networks 3*, pages 467–483.