Uso de Operações SIMD em uma Biblioteca de Algoritmos Bio-inspirados*

Natiele Lucca[1], **Claudio Schepke** [1]

[1]Ciência da Computação – Universidade Federal do Pampa (UNIPAMPA)
Alegrete – RS – Brazil

natielelucca@gmail.com, claudioschepke@unipampa.edu.br

Resumo. *Uma estratégia para modelar algoritmicamente um problema é utilizar conceitos de computação natural, também conhecida como computação bio-inspirada. Neste trabalho é proposta uma biblioteca open source bio-inspirada que utiliza instruções SIMD. Com os resultados dos testes, comprovou-se que a versão paralela dos algoritmos desenvolvidos mantém a qualidade da solução e reduz o tempo de execução.*

1. Introdução

A resolução de um problema pode não ser alcançada de forma exata devido a complexidade dada por um número elevado de variáveis e/ou soluções potenciais. Um algoritmo que descreve um problema sequencialmente pode demandar de uma grande capacidade de processamento para encontrar a solução ótima [Ignácio and Ferreira 2002]. O tempo exigido para a computação exata e sequencial varia de horas à anos, o que destaca a importância de abordagens e técnicas que otimizem a execução de algoritmos [Nievergelt et al. 1995]. Os problemas citados anteriormente podem ser identificados em diversas áreas de pesquisa, como em simulações de atividades reais que envolvem engenharia, ciência e economia.

Nesse contexto, algoritmos bioinspirados são uma estratégia para reduzir o tempo de execução, de maneira que a solução seja mantida ou melhorada. Estes aplicam técnicas de inteligência de enxames ou inteligência de colônias, ou ainda inteligência coletiva, que simulam o comportamento coletivo de sistemas auto-organizados, distribuídos, autônomos, flexíveis e dinâmicos [Serapiao 2009]. Os elementos que integram o enxame ou colônia são capazes de otimizar um objetivo global através da busca colaborativa em um espaço seguindo regras específicas [Kennedy and Eberhart 2001].

O objetivo deste trabalho é aplicar a concorrência em unidades vetoriais com operações SIMD (*Single Instruction Multiple Data*) na biblioteca bio-inspirada[1] desenvolvida pelos autores, a fim de diminuir o tempo de execução e auxiliar os desenvolvedores na solução de problemas. A biblioteca implementa os algoritmos clássicos: Enxame de Partículas (PSO – *Particle Swarm Optimization*); Colônia Artificial de Abelhas (ABC – *Artificial Bee Colony*); e Colônias de Formigas (ACO – *Ant Colony Optimization*).

O artigo está organizado da seguinte forma. A Seção 2 traz os métodos e filtros utilizados para a realização deste trabalho. Na Seção 3 são apresentados os resultados. Por fim, na Seção 4, são destacadas as considerações finais sobre a pesquisa realizada.

*O presente trabalho foi desenvolvido no Laboratório de Estudos Avançados (LEA) com apoio do PDA Iniciação Científica/Unipampa 2018 e PROBIC/FAPERGS 2018 e 2019.

[1]Disponível em https://github.com/NatiLucca/Bio-inspired-library

2. Metodologia

Para validar a implementação é preciso verificar as características dos algoritmos através de testes. Os testes são realizados por funções que possuem propriedades diversas, para garantir que o algoritmo possa ou não resolver certos tipos de otimização com eficiência [Yang 2010]. Para testar os algoritmos implementados na biblioteca bio-inspirada foi desenvolvido um *benchmark*, com uma aplicação base composta por sete funções de teste clássicas[2]: *Alpine*, *Booth*, *Easom*, *Griewank*, *Rastrigin*, *Rosenbrock* e *Sphere*.

Os algoritmos foram desenvolvidos a partir do respectivo pseudocódigo disponível na literatura. Em seguida, diretivas da API OpenMP foram investigadas a fim de verificar qual ou quais são mais adequadas para cada algoritmo. Para comparar as versões sequenciais e paralelas foram gerados arquivos que contém os valores das entradas. Dessa forma, ambas as versões foram submetidas ao mesmo caso de teste. Uma execução compreende a geração dos arquivos de entrada, a execução da função sequencialmente e a execução da função paralela.

A versão sequencial e paralela leem dos arquivos de entrada a população inicial e os demais valores aleatórios do código. A versão paralela contém as mesmas instruções da versão sequencial, tendo apenas o acréscimo das diretivas OpenMP. Ambas as versões retornam o melhor valor encontrado pelo algoritmo e o tempo de execução em segundos. A biblioteca também dispõem de uma versão que retorna além da solução e do tempo, todos os valores utilizados na execução dos testes.

Para medir o desempenho das versões dos algoritmos paralelos, foi utilizado o conceito de *speedup*(S). O *speedup* é definido como a razão entre o tempo de computação do algoritmo serial (T_{serial}) e o tempo de computação do algoritmo paralelo ($T_{paralelo}$). Uma outra forma de mensurar o quanto uma versão paralela é melhor que a versão sequencial é considerar o percentual de ganho de desempenho apresentado na Equação 1.

$$S = \frac{T_{serial} - T_{paralelo}}{T_{paralelo}} * 100 = (\frac{T_{serial}}{T_{paralelo}} - 1) * 100 \quad (1)$$

Todos os testes foram realizados na workstation da Universidade Federal do Pampa (UNIPAMPA), campus Alegrete que possui a seguinte configuração: dois processadores Intel® Core™ Xeon CPU E5-2650, com 2,0 GHz de frequência, 8 núcleos e 16 *threads*, 128 GB de memória RAM e com sistema operacional Ubuntu na versão 16.04 de 64 bits. Os resultados são calculados a partir da média de 30 execuções.

3. Resultados

Após a implementação sequencial da biblioteca e verificação do código, foram feitos testes para avaliar o tempo de execução das implementações. Desta forma, foi possível obter um *baseline*, o qual foi tomado como referência para efeito de comparação dos tempos de execução das implementações paralelas.

A diretiva `#pragma omp simd` foi utilizada, pois, essa diretiva permite usar unidades vetoriais, onde um bloco de instruções é executado simultaneamente. Como uma instrução é executada sobre diferentes dados, para garantir a eficiência desse método

[2]As funções e as suas respectivas características estão disponíveis em http://benchmarkfcns.xyz/fcns

Tabela 1. Casos de Teste - Melhores Resultados.

Função	ABC			PSO		
	Solução	T. Desemp.	Des. Padrão	Solução	T. Desemp.	Des. Padrão
1. Alpine	0,00E+00	35,59	0,0062	1,95E+00	301,01	0,0195
2. Booth	1,30E-17	30,38	0,0017	1,47E-05	602,84	0,0002
3. Easom	-1,00E+00	41,32	0,0094	-9,64E-01	289,20	0,0013
4. Griewank	1,11E-13	53,63	0,0088	1,20E+02	224,88	0,1064
5. Rastrigin	5,35E-13	63,34	0,0131	3,05E+02	276,75	0,1123
6. Rosenbrock	9,16E+00	140,83	0,0118	1,30E+05	2007,38	0,0711
7. Sphere	1,04E-17	35,65	0,0054	1,31E-02	1141,37	0,0027

de paralelismo os endereços acessados no laço de repetição devem estar consecutivos em memória. Desse modo, a alocação de memória contígua é aplicada na implementação dos algoritmos sobre os vetores dos dados. Dos resultados obtidos pela diretiva `#pragma omp simd`, nenhum resultado foi negativo, ou seja, o tempo de execução paralelo foi sempre inferior ao tempo de execução sequencial em todos os casos.

Os testes realizados [3] retornaram a mesma solução ótima para as versões sequencial e paralela, pois ambas eram submetidas ao mesmo caso de teste. A Tabela 1 apresenta para cada algoritmo as soluções médias obtidas a taxa de desempenho em (%) e o desvio padrão para cada respectiva função de teste.

A Tabela 1 apresenta a média dos valores de cada uma das 30 execuções para cada função e algoritmo. Como são realizados testes sobre execuções sequenciais e paralelas os resultados são comparados, pois como os ambas as versões são submetidas aos mesmos valores o resultados deve ser igual. Para todos os testes realizados os resultados obtidos foram iguais. Os valores ótimos para as funções são $-1,00E+00$ para a função Easom e $0,00E+00$ para as demais funções.

O algoritmo ABC obteve as soluções ótimas para as funções 1 e 3, Alpine e Easom, respectivamente. As funções *Booth* e *Sphere* obtiveram bons resultados, sendo que a primeira é bidimensional e a segunda possui 6 dimensões. As funções *Griewank* e *Rastrigin* também obtiveram boas soluções. Essas funções possuem 50 dimensões. A pior solução encontrada pelo algoritmo foi a *Rosenbrock* que também possui 50 dimensões. O algoritmo ABC obteve soluções ótimas e soluções médias próximas à ótima. Dessa forma, analisando as soluções, é um algoritmo que obtém bons resultados aplicado a diferentes funções, sendo que essas funções possuem características diferentes.

O algoritmo PSO obteve soluções próximas as ótimas. As funções *Alpine*, *Booth*, *Easom* e *Sphere* possuem resultados mais próximos ao ótimo, as funções *Booth* e *Easom* são bidimensionais e as funções *Alpine* e *Sphere* são n-dimensionais de 10 e 6, respectivamente. As funções *Griewank*, *Rastrigin* e *Rosenbrock* que possuem resultados distantes do ideal são n-dimensionais todas com 50 dimensões. Dada as características dos algoritmos bio-inspirados, certas casos de teste levam mais iterações para encontrar o ótimo global. Logo, as soluções podem ser melhoradas com o aumento do número de iterações do algoritmo.

[3] Disponível em: https://drive.google.com/drive/folders/ 1sFq1T2sKbkmPp1DGrz1ISooR6uqy2Vng?usp=sharing

No algoritmo ABC, as funções bidimensionais *Booth* (30,30%) e *Easom* (41,32%) possuem as menores taxas de ganho de desempenho, seguidas pelas funções *Alpine* (35,59%) e *Sphere* (36,65%). Os maiores ganhos de desempenho são para as funções *Griewank* (53,63%), *Rastrigin* (63,34%) e *Rosenbrock* (140,83%). Todas possuem 50 dimensões. Observa-se que o ganho de desempenho aumenta conforme a dimensionalidade aumenta, proporcional a granularidade e as características do algoritmo.

O algoritmo PSO obteve taxas de ganho de desempenho significativas para as funções *Alpine* (301,01%), *Booth* (602,84%), *Easom* (289,20%), *Griewank* (224,88%), *Rastrigin* (276,75%) e *Rosenbrock* (2007,388%) e *Sphere* (1141,37%).

É observável que as melhores médias variam o valor da população que a atingiu, tanto de soluções quanto de ganho de desempenho. Cada problema tem um algoritmo e uma população que é melhor aplicado ao problema.

4. Considerações Finais

Os algoritmos bio-inspirados correspondem a uma gama de estratégias eficientes para a solução de problemas de diversas áreas, desde uma simples análise de dados até problemas complexos como de geração de energia distribuída ou simulações de engenharia. O objetivo principal deste trabalho foi alcançado com a paralelização da biblioteca e o ganho de desempenho obtido nos testes dos algoritmos.

O ganho de desempenho obtido no PSO é superior ao ABC devido as características dos algoritmos. Enquanto o PSO é mais simples, com poucos saltos no código, o ABC possui mais estruturas condicionais que limitam a área paralela. A função *Rosenbrock* obteve a maior taxa de ganho de desempenho em ambos os algoritmos devido a granularidade. As características dessa função permitem que ela possa ser paralelizada.

As operações SIMD permitem explorar dos recursos da arquitetura vetorial. Dessa forma, é possível executar uma instrução sobre múltiplos dados utilizando os registradores vetoriais. Assim obtém-se um ganho de desempenho significativo como o apresentado neste trabalho.

Como trabalhos futuros buscamos, com o auxílio da comunidade de desenvolvedores, complementar a biblioteca paralela bio-inspirada com novos algoritmos paralelos.

Referências

Ignácio, A. A. V. and Ferreira, V. J. M. F. (2002). Mpi: uma ferramenta para implementação paralela. *Pesquisa Operacional*, 22:105 – 116.

Kennedy, J. and Eberhart, R. C. (2001). *Swarm Intelligence*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Nievergelt, J., Gasser, R., Mäser, F., and Wirth, C. (1995). *All the needles in a haystack: Can exhaustive search overcome combinatorial chaos?* Springer Berlin Heidelberg, Berlin, Heidelberg.

Serapiao, A. (2009). Fundamentos de Otimização por Inteligência de enxames: Uma Visão Geral. *Controle y Automacao*, 20:271–304.

Yang, X.-S. (2010). Test problems in optimization. *Engineering Optimization: An Introduction with Metaheuristic Applications*.