Operações Vetoriais Aplicadas em uma Biblioteca de Algoritmos Bio-inspirados*

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

[1]Programa de Pós-Graduação em Engenharia de Software
Universidade Federal do Pampa (UNIPAMPA) - Campus Alegrete – RS – Brasil

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

Abstract. *Bio-inspired algorithms are based on the collective behavior of social organisms and are used to solve optimization problems. In this work, concurrency is applied in vector units using Single Instruction Multiple Data (SIMD) architectures in the bio-inspired library developed by the authors, in order to decrease the execution time of the algorithms. An evaluation was made using a test function applied to each of the implemented algorithms. The test allowed to show that parallel implementations obtained performance in all cases, reaching up to 5 times in ACO algorithm.*

Resumo. *Algoritmos bio-inspirados são baseados no comportamento coletivo de organismos sociais e utilizados para a resolução de problemas de otimização. Neste trabalho é aplicada a concorrência em unidades vetoriais na biblioteca bio-inspirada desenvolvida pelos autores, a fim de diminuir o tempo de execução dos algoritmos. Uma avaliação foi feita utilizando uma função de teste aplicada a cada um dos algoritmos implementados. O teste permitiu mostrar que as implementações paralelas obtiveram ganhos de desempenho em todos os casos, chegando até 5 vezes no algoritmo ACO.*

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]. Nesse contexto, algoritmos bio-inspirados são uma estratégia para reduzir o tempo de execução, de maneira que a solução seja mantida ou melhorada. Estes algoritmos 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].

Este artigo avalia uma biblioteca[1] para a resolução de problemas utilizando algoritmos bio-inspirados. A biblioteca implementa os algoritmos clássicos: Enxame de

*O presente trabalho foi desenvolvido pelo Laboratório de Estudos Avançados (LEA) com apoio das bolsas de iniciação científica PDA/Unipampa 2018 e PROBIC/FAPERGS 2018 e 2019.

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

Partículas (PSO – *Particle Swarm Optimization*); Colônia Artificial de Abelhas (ABC – *Artificial Bee Colony*); e Colônias de Formigas (ACO – *Ant Colony Optimization*). Para a avaliação, foi proposto o paralelismo utilizando operações do tipo SIMD (*Single Instruction Multiple Data*) através da interface de programação OpenMP, com o intuito de oferecer desempenho na execução dos algoritmos da biblioteca, diminuindo ainda mais o tempo de execução para a solução de problemas.

O objetivo deste artigo é mensurar o desempenho paralelo da biblioteca para um cenário com diferentes tamanhos de população. A contribuição deste artigo é avaliar e o desempenho da execução concorrente em arquiteturas paralelas.

O artigo está organizado da seguinte forma. A Seção 2 apresenta de forma sistematizada os trabalhos relacionados. A Seção 3 descreve a metodologia de desenvolvimento e testes utilizados para a realização deste trabalho. Na Seção 4 são apresentados os resultados. Por fim, na Seção 5, são destacadas as considerações finais sobre o trabalho.

2. Trabalhos Relacionados

[Silva et al. 2018] apresenta uma análise comparativa 25 bibliotecas de otimização disponíveis na literatura, identificando os pontos fortes e fracos de cada uma. O artigo verifica possíveis carências e possibilidades de trabalhos futuros, uma vez que o mesmo busca disponibilizar um material completo sobre as ferramentas de otimização. Das ferramentas avaliadas em [Silva et al. 2018], apenas seis possuem implementações bio-inspiradas: ECJ[2], EvA2[3], FOM[4], ParadisEO[5], MALLBA[6], e Opt4[7]. Além dessas bibliotecas, foram ainda encontradas na literatura PaGMO[8] e PyGMO[9].

Em relação às bibliotecas paralelas implementadas em C++, MALLBA possui algoritmos implementados nas interfaces de programação OpenMP e PThreads, PaGMO usa a interface MPI e ParadisEO explora os recursos das interfaces MPI e PThreads.

Em nosso trabalho, nós propomos uma biblioteca implementada em C++ que fornece as implementações dos algoritmos PSO, ABC e ACO. Além disso, o paralelismo é explorado através da granularidade fina, ideal para as novas arquiteturas de computadores e suas instruções vetoriais. Para tanto, são utilizadas instruções SIMD das diretivas da interface de programação paralela OpenMP, ainda não exploradas na literatura.

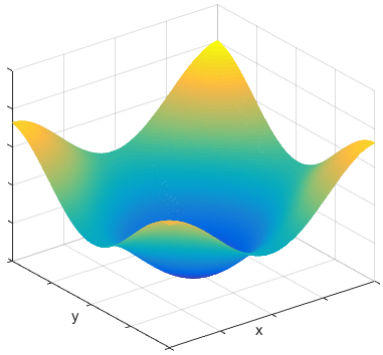
3. Aspectos Metodológicos

3.1. Implementação da Biblioteca

A biblioteca bio-inspirada foi desenvolvida na linguagem de programação C++, com o objetivo de facilitar a programação e dispor de recursos que permitam solucionar problemas de forma eficiente. Essa linguagem é orientada a objetos logo, possibilita abstração, encapsulamento, herança e polimorfismo o que é essencial para as boas práticas de geração

[2]Disponível em http://cs.gmu.edu/∼eclab/projects/ecj/
[3]Disponível em http://www.ra.cs.uni tuebingen.de/software/EvA2/
[4]Disponível em http://www.isa.us.es/fom
[5]Disponível em http://paradiseo.gforge.inria.fr/
[6]Disponível em http://neo.lcc.uma.es/mallba/easy-mallba/
[7]Disponível em http://opt4j.sourceforge.net/
[8]Disponível em: http://esa.github.io/pagmo/index.html
[9]Disponível em: http://esa.github.io/pygmo/index.html

Figura 1. Gráfico Função Alpine



[Ardeh 2016]

de código. A linguagem também oferece bibliotecas de manipulação de vetores e matrizes e para geração de valores aleatórios. Esses contribuem com a implementação dos algoritmos de inteligência de enxame.

Os algoritmos de computação bio-inspirada selecionados para este estudo pertencem a classe de inteligência de enxame, logo simulam o comportamento de agentes diante da colônia. Foram escolhidos os algoritmos ABC, ACO e PSO, pois possuem características desejáveis para a otimização de problemas que não podem ser resolvidos ou solucionados em tempo polinomial. A estrutura dos algoritmos permite que certos blocos de código possam ser paralelizados aumentando o desempenho da aplicação. O paralelismo dos algoritmos da biblioteca foi implementado pela API OpenMP, pois é compatível com C++ e possui uma interface simples para o desenvolvimento de aplicações paralelas.

A diretiva `#pragma omp simd` foi utilizada, pois, essa diretiva permite usar as unidades vetoriais, onde um bloco de instruções é executado simultaneamente. Como uma instrução é executada sobre diferentes dados, os endereços acessados no laço de repetição devem estar consecutivos em memória para garantir a eficiência desse método de paralelismo. Desse modo, a alocação de memória contígua é aplicada na implementação dos algoritmos sobre os vetores dos dados.

3.2. Função Alpine

Para validar a implementação de um algoritmo é preciso verificar as características do mesmo através de testes. Neste trabalho, um teste foi realizado, ou seja, os algoritmos ABC, ACO e PSO foram submetidos a uma função de teste chamada Alpine. A função Alpine é expressa na 1.

$$f(\mathbf{x}) = \sum i = 1^{n} |x_i sin(x_i) + 0.1x_i| \tag{1}$$

As características da função são 10 para dimensão do agente, [0,10] para espaço de busca, 3000 número máximo de iterações e f(0,...,0) = 0 para mínimo global que foram estabelecidos de acordo com [Serapiao 2009] e [Ardeh 2016]. O gráfico que expressa o comportamento da função é apresentado na Figura 1.

3.3. Execução dos Testes

Os resultados finais compreendem a média de 30 execuções, assim como adotado no trabalho de [Couto et al. 2015]. Uma execução compreende a geração dos arquivos de entrada, a execução sequencial e, por último, a execução paralela. Os arquivos de entrada são gerados através do método *runDetails*. Esse método cria uma pasta chamada *Inputs*, que armazena os arquivos de entrada gerados durante a execução do método. Em seguida a versão sequencial é executada pelo método *run*, sendo que a população e os demais valores aleatórios do código são inicializados com os valores dos arquivos gerados anteriormente no método *runDetails*. Por fim, a versão paralela é calculada pelo método *runParallel*. A versão paralela contém as mesmas instruções da versão sequencial, tendo apenas o acréscimo das diretivas OpenMP. Os métodos *runDetails*, *run* e *runParallel* retornam a melhor solução encontrada pelo algoritmo e o tempo de execução em segundos.

Os testes foram realizados em uma workstation composta por 2 processadores Intel(R) Xeon(R) Silver 4116 CPU 2.10GHz (48 threads) e 96 GB de memória RAM.

3.4. Automatização dos Testes

O lançamento dos testes são automatizados através de instruções em *shell script*. Essa linguagem permite a criação de pastas para armazenar os arquivos de saída de cada teste e a criação de *loops* sobre uma linha de comando, permitindo que ocorra o lançamento de 30 repetições sobre cada passo da execução. A linguagem também possibilita o redirecionamento da saída dos métodos para um arquivo. Dessa forma os resultados das execuções são armazenados em um arquivo do tipo `.csv`, com o nome da função, onde cada linha do arquivo corresponde a solução e o tempo de execução mensurado.

De acordo com os estudos de [Talukder 2011] e [Abraham et al. 2010] o número de agentes responsáveis por determinar o ótimo global de uma função corresponde ao intervalo de 20 a 60 agentes. Dessa forma, as funções foram testadas para uma população de 10 até 100 agentes variando de 10 em 10 unidades.

4. Resultados Experimentais

A Tabela 1 apresentam respectivamente os resultados obtidos nos testes para os algoritmos PSO, ABC e ACO, aplicando-os na função de teste Alpine variando o tamanho da população. As tabelas apresentam o tempo médio de execução sequencial e paralelo em segundos, o ganho de desempenho em porcentagem e a média numérica das soluções. Para cada algoritmo é destacado em negrito qual foi o melhor ganho de desempenho e qual foi a melhor média das soluções.

Tabela 1. Resultados Obtidos nos Testes da função Alpine

Algoritmo	Pop.	Sequencial (Segundos)	Paralelo (Segundos)	Ganho de Desemp.(%)	Média das Soluções
ABC	10	0,079604	0,068103	16,89	0,00E+00
	20	0,156575	0,115475	**35,59**	0,00E+00
	30	0,233024	0,173081	34,63	0,00E+00
	40	0,300065	0,226886	32,25	0,00E+00
	50	0,378131	0,290092	30,35	0,00E+00
	60	0,448755	0,346298	29,59	0,00E+00

	70	0,525266	0,403139	30,29	0,00E+00
	80	0,604550	0,459817	31,48	0,00E+00
	90	0,689727	0,516879	33,44	0,00E+00
	100	0,753482	0,568900	32,45	0,00E+00
ACO	10	0,799142	0,134525	494,05	4,07E-07
	20	1,680333	0,276251	508,26	2,37E-15
	30	2,679626	0,439530	509,66	9,01E-12
	40	3,697194	0,607278	508,81	1,99E-19
	50	4,773935	0,772923	517,65	3,49E-01
	60	5,854582	0,940861	522,26	2,94E-01
	70	6,971155	1,112537	526,60	5,88E-03
	80	8,100047	1,351876	499,17	**2,07E-27**
	90	9,370560	1,477861	**534,06**	3,13E-20
	100	10,468643	1,667490	527,81	3,43E-25
PSO	10	0,062815	0,021716	189,25	7,99E+00
	20	0,123000	0,034748	253,97	5,56E+00
	30	0,167739	0,056752	195,56	4,85E+00
	40	0,213916	0,053447	300,24	3,47E+00
	50	0,268318	0,070593	280,09	3,03E+00
	60	0,318456	0,092839	243,02	**1,95E+00**
	70	0,370068	0,091106	**306,19**	3,44E+00
	80	0,420565	0,104877	301,01	4,02E+00
	90	0,469736	0,121351	287,09	3,51E+00
	100	0,520913	0,135383	284,77	2,09E+00

As versões sequencial e paralela foram submetidas aos mesmos casos de teste[10], retornando as mesmas soluções ótimas.

Em relação ao desempenho, primeiramente foi considerada a implementação sequencial da biblioteca. Desta forma, foi possível obter um *baseline*, o qual foi tomado como referência para efeito de comparação dos tempos de execução das implementações paralelas. Conforme pode ser visto na Tabela 1, houve um ganho de desempenho nas versões paralelas em todos os casos testados.

A função *Alpine* obteve os maiores ganhos de desempenho com o algoritmo ACO em todos os testes (cerca de 5×). Entretanto, o algoritmo ABC obteve para a função *Alpine* as soluções ótimas em todos os testes. O algoritmo ACO foi mais eficiente quanto ao ganho de desempenho para os algoritmos *Alpine*. Quando avaliado a solução, o algoritmo ABC obteve as melhores soluções para o algoritmo *Alpine*.

O resultados dos algoritmos estão relacionados com a capacidade de vetorização dos dados a serem processados. Embora o trabalho de implementação tenha levado isso em consideração, mantendo os dados contíguos em memória, o algoritmo ABC possuem menos ou menores trechos que possam ser vetorizados, o que já não acontece nos algoritmos PSO e ACO. Por outro lado, o algoritmo ABC foi o melhor na maioria dos casos

[10]Disponível em: https://drive.google.com/open?id=1zTTx11hjiGYwGAzvZ8IFJ7Ln7l4sKXwV

para encontrar a solução ótima.

5. Considerações Finais e Trabalhos Futuros

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. A contribuição deste trabalho foi avaliar o desempenho de uma biblioteca que possibilite o uso de algoritmos bio-inspirados para a resolução de problemas.

Neste artigo, o ganho de desempenho da biblioteca foi alcançado através da paralelização, usando diretivas OpenMP SIMD. Em praticamente todos os casos foi possível acelerar os algoritmos, atingindo um ganho de desempenho de até 5 vezes em relação à execução sequencial. Os testes dos algoritmos PSO, ABC e ACO foram feitos usando uma função e 10 tamanhos de população distintos.

Como trabalhos futuros pretende-se incluir novos algoritmos bio-inspirados à biblioteca, além de torná-la extensível. Também deseja-se validar a mesma testando os algoritmos em um problema real, como o da produção distribuída de energia.

Referências

Abraham, S., Sanyal, S., and Sanglikar, M. (2010). Particle swarm optimization based diophantine equation solver. *arXiv preprint arXiv:1003.2724*.

Ardeh, M. A. (2016). BenchmarkFcns Toolbox: A collection of benchmark functions for optimization. [Online; acesso em 10-Janeiro-2020].

Couto, D. C., Silva, C. A., and Barsante, L. S. (2015). Otimização de funções multimodais via técnica de inteligência computacional baseada em colônia de vaga-lumes. In *Proceedings of the XXXVI Iberian Latin American Congress on Computational Methods in Engineering*, Rio de Janeiro, RJ. CILAMCE.

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.

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

Silva, M. A. L., de Souza, S. R., Souza, M. J. F., and de Franca Filho, M. F. (2018). Hybrid metaheuristics and multi-agent systems for solving optimization problems: A review of frameworks and a comparative analysis. *Applied Soft Computing*.

Talukder, S. (2011). Mathematicle modelling and applications of particle swarm optimization.